Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

VoyContigo: Aplicación móvil de sistema de emisión y gestión de alertas para situaciones de riesgo y vulnerabilidad

Proyecto Tecnológico

Reporte del Proyecto de Integración

Trimestre 2022 Invierno

Moisés Aguirre Bautista
2163033692
al2163033692@azc.uam.mx

Asesora
Dra. Angeles Belém Priego Sánchez
Profesora Asociada
Departamento de Sistemas
abps@azc.uam.mx

Co-asesora
Dra. María Auxilio Medina Nieto
Profesora Titular
Universidad Politécnica de Puebla
maria.medina@uppuebla.edu.mx

26 de mayo del 2022

Declaratoria

Yo, Dra. Angeles Belém Priego Sánchez, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Dra. Angeles Belém Priego Sánchez

Yo, Dra. María Auxilio Medina Nieto, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Dra. María Auxilio Medina Nieto

Yo, Moisés Aguirre Bautista, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como el Repositorio Institucional de UAM Azcapotzalco.

Moisés Aguirre Bautista

Resumen

La aplicación móvil "***VoyContigo***" se enfoca como una herramienta, ya que proporciona apartados que sean funcionales para usuarios que pudiesen estar en una situación de riesgo u que deseen identificar la incidencia de este.

Los apartados que contempla la aplicación son la configuración y registro de usuarios, un sistema de emisión de alertas de riesgo, la configuración de contactos y un mapa virtual del conteo de las incidencias emitidas por los usuarios.

El apartado de configuración y registro de usuarios permitirá al usuario poder crear una cuenta para poder tener acceso a la funcionalidad de la aplicación, donde también podrá modificar sus datos personales.

En el apartado de emisión de alertas de riesgo se basa en él envió de mensajes de texto indicando la situación en la que pudiese encontrarse un usuario, basado a tres niveles (bajo, medio y alto). Dichos mensajes contendrán una leyenda predefinida de la situación en la que se encuentra el usuario, seguida de la ubicación en tiempo real donde fue accionada.

En el apartado de configuración de contactos se podrá acceder a los contactos del usuario donde se les podrá asignar una alerta, para que de esta manera los mensajes que serán enviados puedan ser recibidos por los contactos configurados.

El apartado del mapa virtual se podrá visibilizar un mapa virtual que identificara las alertas que han sido emitidas través de marcadores indicando en el mapa la ubicación de donde se acciono, mostrando el tipo de riesgo, la fecha y la hora que fueron emitidas.

De tal modo que la aplicación "***VoyContigo***" traerá como beneficio la disminución la interacción de un usuario con el dispositivo móvil para notificar a sus contactos que pudiese encontrarse en una situación de riesgo,

Para el desarrollo de la aplicación móvil, "***VoyContigo***" integra una arquitectura REST basada en el modelo cliente servidor, de tal manera que a través de las peticiones HTTP permitirá ser funcional la aplicación en el envío y recibo de información. Donde se contempla el desarrollo y configuración del Back-End y Front-End donde se integran técnicas para el desarrollo de componentes, implementando conceptos y uso de tecnologías que ayudan al desarrollo de las aplicaciones móviles.

Dicho desarrollo de Back-End se hará uso de Node.js con Express.js que servirán para la configuración del servidor que implementara una base de datos en MariaDB, donde se establecerá la conexión y manipulación de los datos a través de los métodos de las peticiones HTTP, para que de esta manera sean consumibles a través del Front-End.

Para el Desarrollo del Front-End se hará uso de un Framework de desarrollo de aplicaciones móviles para Android y IOS el cual será React-Native que proporciona una serie de componentes que pueden ser reutilizables, haciendo así el desarrollo más flexible y que permite el consumo e integración del Back-End.

Ambos desarrollos del Back-End y Front-End serán desarrollados en el lenguaje de programación TypeScript ya que permite seguir técnicas de desarrollo más flexible.

Índice

Índice de Figuras

Índice de Código

Índice de Tablas

Introducción

En el mundo se han presenciado una de las problemáticas más grandes en la sociedad como han sido las desapariciones forzadas, la intimidación, el acoso o verse expuesto en algunas circunstancias de riesgo, esto ha traído consigo la inseguridad de la población, donde en un mayor porcentaje se ve influenciada la mujer a ser víctima en estos escenarios en su día a día, en el trasporte público, calles e incluso su propio hogar.

Es por eso que en ocasiones la sociedad se ve preocupada ante esta exposición de los distintos escenarios en donde pueden convertirse en víctimas, dejándolos vulnerables ante una situación de riesgo, ya que no es fácil lidiar con los distintos escenarios. Actualmente, ha habido cambios y programas para la seguridad social. Sin embargo, no han tenido el impacto necesario para poder combatir estos acontecimientos de inseguridad, que involucren a la sociedad a ser partícipes de un apoyo mutuo como también focalizar los espacios de riesgo, llevando a cabo investigaciones para la prevención y eliminación de estas problemáticas.

El diseñar un sistema de alertas, eficiente y accesible, donde el usuario pueda configurar los contactos que las recibirán y podrán personalizar el tipo de alerta que quiera emitir, servirá como un mecanismo de apoyo para notificar la situación en la que pueda presentarse.

De la misma manera, el poder crear un mapa donde se han emitido las alertas y focalizar los puntos donde hay más incidencia de los tipos de casos presentados a partir de las alertas expuestas por los usuarios, lograran visibilizar puntos de riesgo y alertar a más usuarios, dando paso a crear la red para fortalecer la seguridad y el poder proceder a emitir la denuncia para la reducción de situaciones de riesgo.

Antecedentes

Referencias Internas

Sistema móvil para alertas de consumo de medicamentos y seguimiento de pacientes con diabetes [1]

Este proyecto desarrolla una aplicación móvil para generar alertas sobre el consumo de medicamentos, dando seguimiento a pacientes con diabetes, permitiendo dar respuesta sobre el control de consumo de medicamentos, expidiendo alertas a tutores del usuario, en caso omiso del consumo de los medicamentos registrados en el horario especificado.

Síntesis procedural aproximada de modelos urbanos 3D a partir de imágenes de mapa virtual de Google Maps [2]

Este proyecto desarrolla una aplicación para sintetizar automáticamente modelos de imágenes en 3D, a partir de imágenes de Google Maps, detectando los bordes de las imágenes por medio del algoritmo de Canny Edge, como reduciendo el ruido de estas a través de un filtro Gaussiano.

Referencias Externas

SISTER - La APP que te protege [3]
Es una aplicación de seguridad para mujeres con geolocalización en tiempo real, que te permite elegir a tus contactos de confianza y compartir la ubicación en tiempo real con ellos, busca y selecciona una ruta segura, emite alertas SOS en casos de emergencia a contactos como a otras personas.

En qroo no estoy sola [4]
Es una aplicación que mantiene informados a los contactos que personalices en caso de una emergencia, permitiendo el envío de mensaje de texto al presionar un clic.

Mujer segura alerta rosa [5]
"La aplicación que funciona de la mano con una pulsera Mujer Segura Alerta Rosa y ofrece un mecanismo de botón de pánico que envía una señal directa al C5 (Centro de Control, Comando, Comunicación, Cómputo y Calidad del Estado de México) y permite a las autoridades identificar el punto geográfico desde donde se emitió dicha alerta para atenderla con los servicios de seguridad correspondiente."

Sosmex (botón de pánico) #niunamenos [6]
Es una aplicación que ayuda a mujeres en situaciones de peligro, consiste en un botón de pánico el cual es activado por la aplicación, el sensor de movimiento.

Vive segura CDMX [7]
"App de uso personal diseñada para contribuir a la seguridad de las mujeres y niñas. Contiene una serie de funciones; mapa en línea, llamada de emergencia al 066, reporte de la sensación de seguridad, del estado del mobiliario y espacio público y de casos de acosos, información sobre derechos de las mujeres."

La Tabla 1 muestra una comparación de los trabajos relacionados con el proyecto propuesto.

Tabla 1 . Comparación cualitativa de los trabajos relacionados con el proyecto propuesto

Referencia	Similitudes	Diferencias
[1]	▪ Alarma de caso omiso de consumo de medicamentos implementado en java ▪ Aplicación móvil	▪ Alarma implementada en TypeScript [13] ▪ La aplicación móvil contiene una interfaz gráfica más amigable con el usuario
[2]	▪ Obtiene recursos digitales de un mapa virtual, para la extracción de imágenes.	▪ Obtiene recursos de localización de un mapa virtual
[3]	▪ Mapa de riesgo ▪ Alarma de riesgo	▪ Define el tipo de alertas emitidas en el mapa de riesgo
[4]	▪ Alarma de riesgo ▪ Selección de contactos	▪ Se pueden gestionar y personalizar cada alarma, asignando contactos distintos para cada situación si así lo desea el usuario.
[5]	▪ Botón de alarma	▪ Permite emitir la alarma enviándola por mensaje de texto
[6]	▪ Alarma de riesgo	▪ Logra emitir varios tipos de alerta, asignándole una situación
[7]	▪ Llamada de emergencia ▪ Alarma de riesgo	▪ Proporciona distintas alternativas de emergencia, como un apoyo comunitario de otros usuarios. ▪ Logra emitir varios tipos de alerta, asignándole una situación

Justificación

Actualmente han sido visibles muchos casos de violencia, acoso e intimidación, donde se han presenciado una gran incidencia de la inseguridad de la población, trayendo esto consigo grandes problemas como daños psicológicos, desapariciones, incluso la privación de la vida en los casos extremos. Sin embargo, lo que se pretende es alertar a los contactos de usuarios para mantenerlos al tanto de alguna de las situaciones de riesgo en las que el usuario pueda estar expuesto, de tal manera que el usuario notifique a los contactos que haya predefinido su ubicación en tiempo real y el posible riesgo en el que se encuentra,

El desarrollo de esta aplicación puede contribuir a mantener comunicados a usuarios con personas de confianza para notificarles de manera accesible que se ven expuestas ante una situación de riesgo o vulnerabilidad. Así mismo el tener un contador de las alertas emitidas por los usuarios que utilicen la aplicación, posibilita focalizar puntos de alta incidencia en un mapa virtual de alguno de los casos de acoso, intimidación, violencia, robo, etc. La incidencia de alertas en una zona permitirá al usuario tomar medidas y acciones de prevención para sus traslados.

Objetivos

Objetivo General

Desarrollar una aplicación móvil que permita gestionar y emitir alertas en tiempo real, visualizando lugares de riesgo a través de un mapa virtual de las alertas emitidas.

Objetivos Específicos

1. Implementar un módulo capaz de crear y configurar alertas
2. Implementar un módulo capaz de emitir alertas.
3. Implementar un módulo capaz de visualizar un mapa virtual de rastreo en tiempo real.
4. Implementar un módulo contador de las alertas emitidas en el mapa

Marco teórico

Para el desarrollo del proyecto se describirá los conceptos, tecnologías y herramientas para llevar a cabo la implementación de dicho proyecto en las cuales se hablarán a continuación.

Arquitectura de la aplicación

Para la creación de nuestra aplicación móvil se contará con la planificación de una arquitectura cliente-servidor donde se dividirá entre el desarrollo de la lógica de negocio y el desarrollo de la interfaz gráfica. A estos dos bloques se les describirá como Back-End y Front-End como se muestra en la Figura 1.

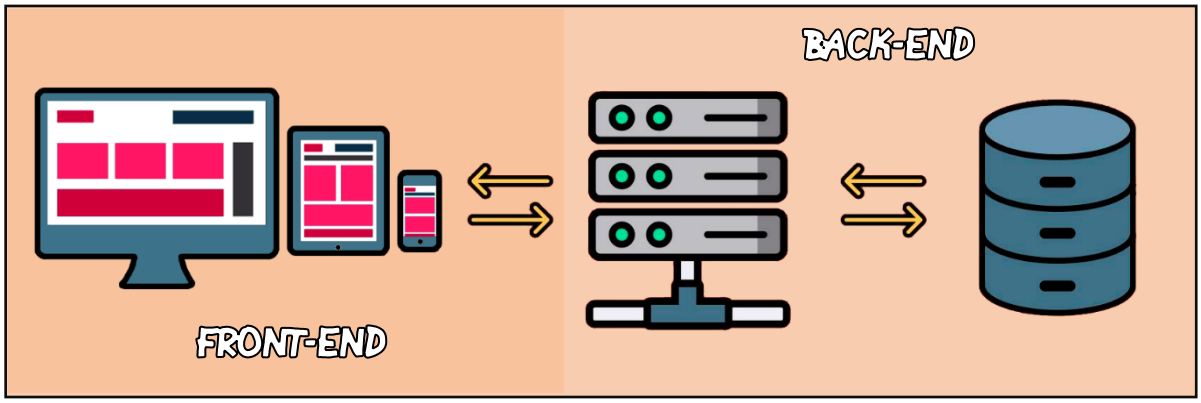


Figura 1 Modelo Front-End Back-End

Back-End

Es un término empleado para describir el desarrollo de la lógica de negocio tecnológica, que se hace para procesar la información del lado del servidor, es decir que se pueda crear una API que canalice información para que en el lado del Front End el usuario pueda consumir y realizar peticiones al servidor respondiendo esta con información, donde puede encontrarse una base de datos devolviendo esta información al cliente a través del Front End.

Front-End

Es el termino empleado para el desarrollo de las interfaces de usuario en cargada de darle la experiencia tecnológica comúnmente conocida “como lado del usuario”, podemos encontrar algunos elementos como son páginas, gráficos, botones, efectos visuales, enlaces, desplazamientos, menús, etc.

Una vez definidas las dos partes a desarrollar, se empleará una arquitectura de la web para poder comunicar ambas partes la cual será REST.

REST

"REST (Transferencia de Estado Representacional) creado por Roy Fielding. REST es un estilo de arquitectura de software para sistemas hipermedia distribuidos" [16], se utiliza para describir cualquier interfaz que trasmite datos sobre HTTP sin capa adicional.

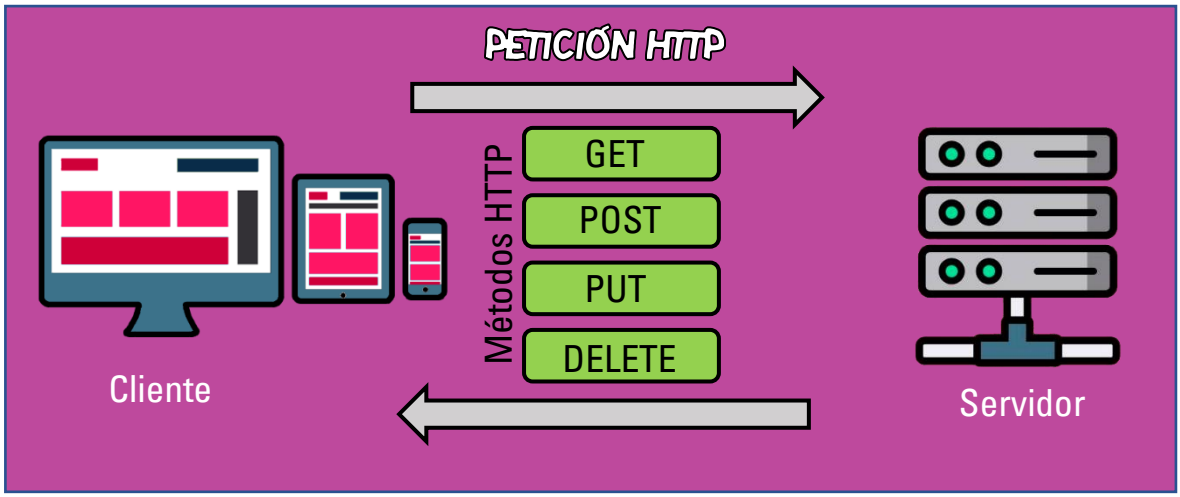


Figura 2 Arquitectura REST

Como se observa en la Figura 2 se necesitará crear una API REST que permita lanzar las peticiones HTTP para establecer una comunicación entre el servidor y el cliente, para la realización de las operaciones CRUD (*Create, Read, Update, Delete*).

API

Una Interfaz de programación de aplicaciones (API) es un conjunto de subrutinas, funciones, métodos, protocolos y definiciones que se utilizan para integrar software de aplicaciones, otorgando flexibilidad ya que simplifican el diseño, intercambiando datos a través de diferentes sistemas, el cual no necesariamente se necesita saber cómo están implementados.

Tecnologías Implementadas

Para el desarrollo de la ampliación móvil se utilizará *Framework* de trabajo para agilizar los procesos de implementación siguiendo algunas reglas que permiten realizar buenas prácticas.

Framework

Un *Framework* es un entorno de trabajo que permite agilizar los procesos de desarrollo, ya que está basado en conceptos estandarizados enfocados en prácticas para resolver problemas de referencias comunes en el desarrollo de software, ofreciendo un conjunto de criterios basados en sus herramientas y módulos que ofrecen.

Dicho de esta manera el *Framework* a utilizar será React Native.

React Native

"React Native [8] es un *Framework* de código abierto creado por *Mata Platfforms"* [18], que permite crear aplicaciones móviles nativas para Android y IOS, basadas en lenguaje de programación de JavaScript donde también permite su desarrollo en TypeScript.

Como bien se mencionó React-Native está basado en el lenguaje de programación Java Script.

JavaScript

"JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript" [19]. basado en prototipos, multiparadigma de un solo hilo, dinámico con soporte a la programación orientada objectos, imperativa y declarativa.

Sin embargo, en la búsqueda de agilizar el proceso del desarrollo de la aplicación móvil se encontró que React-Native acepta una ampliación del lenguaje JavaScript llamado TypeScrip [13].

TypeScript

TypeScript [13] es un lenguaje de programación tipado que se basa en JavaScript desarrollado y mantenido por Microsoft desde el 2012. Typescript es un lenguaje de programación libre construido un nivel superior que JavaScript, ya que es un super set expandido de las funcionalidades de JavaScript permitiendo escribir código más sencillo con menor errores al desarrollar ya que suele ser el código más limpio y coherente.

De esta forma al ser una ampliación del lenguaje JavaScript, el utilizar TypeScript permita que la sintaxis del código sea mucho más clara, ya que es un punto a favor en tiempo de desarrollo y lectura de la misma programación.

Si bien teniendo seleccionado la tecnología de React-Native es importante el contar con un entorno para JavaScript o bien TypeScript que permita ejecutar el código en un servidor la cual será Node.js.

Node.js

Node.js [20] es un entorno en tiempo de ejecución multiplataforma de código abierto controlado por eventos basado en JavaScript. Está diseñado para crear aplicaciones escalabres, teniendo un modelo asíncrono y una arquitectura orientada a eventos basado en el motor V8 de Google.

Así de esta manera el contar con Node.js nos permitirá crear un marco de aplicación para la configuración del Back-End y el servidor, es por eso que también se hará uso de Express.js.

Express.js

Express.js [21] es un marco de aplicaciones Back-End de código abierto multiplataforma y gratuito, que trabaja en tiempo de ejecución, diseñado para construir aplicaciones web y API's desarrolladas en el lenguaje JavaScript de manera mínima y flexible. Permitiendo desarrollar herramientas del lado del servidor proporcionando un conjunto solido de características para aplicaciones web y móviles.

Cabe mencionar que en la configuración del Back-End y el servidor se hará uso de un gestor de bases de datos para almacenar la información de los usuarios que utilizaran el sistema a desarrollar el cual será MariaDB [12].

MariaDB

MariaDB [12] es un sistema de gestión de bases de datos relacionales de código abierto, desarrollado por Michael Widenius, surgiendo a partir de MySQL derivando de esta manera la incorporación de las funciones ya conocidas por MySQL y creando mejoras para la ejecución de consultas complejas y almacenamiento directo en cache, posibilitando acceder a el *cluster* de datos.

Entorno de Desarrollo

Para poder implementar el proyecto será necesario configurar el ambiente de desarrollo por lo que es necesario la instalación de algunas herramientas que permitirán agilizar los procesos de codificación y tener un ambiente de pruebas que permita visualizar los avances de simulación de los resultados.

Para la codificación se utilizará un entorno de desarrollo llamado Visual Studio Code.

Visual Studio Code

Visual Studio Code [14] es un editor de código fuente gratuito, de código abierto, ligero basado en el Framework Electron que se utiliza para crear aplicaciones de escritorio, por lo que Visual Studio Code está disponible para aplicaciones de escritorio para Windows, Linux y macOS.

Visual Studio Code viene con soporte para JavaScript, TypeScript y Node.js incluyendo soporte para la depuración, resaltando la sintaxis con un ecosistema extensible para otros lenguajes de programación.

También se hará uso del entorno Android Studio Code que ofrece un emulador de dispositivos virtuales de Android.

Android Studio Code

"Android Studio [15] es un entorno de desarrollo integrado (IDE) para desarrollar aplicaciones para Android, basado en IntelliJ IDEA" [22], disponible para Windows, Linux, macOS y Chrome OS, remplazando a Eclipse como IDE oficial para desarrollo de aplicaciones Android con Java y Kotlin

Dispositivo Virtual de Android - *Android Virtual Device* (AVD)

"AVD es una configuración que define las características de un teléfono o una tablet Android, o de un dispositivo Wear OS, Android TV o Automotive OS, que desea simular en *Android Emulator*" [23].

Esta configuración permitirá hacer las pruebas necesarias en distintos dispositivos Android, configurando las dimensiones de las pantallas de los dispositivos móviles que ofrece, como las versiones de desarrollo de sistema operativo Android.

Gestor de Paquetes

Para la automatización de procesos como configuraciones que se requieran para el proyecto como instalaciones, actualización y eliminaciones de paquetes a utilizar se harán uso de algunos gestores de paquetes

NPM

NPM (*Node Package Manager*) [24] es un sistema de gestión de paquetes para Node js, el cual permite gestionar de manera óptima dependencias y módulos de la distribución de paquetes, sirviendo de ayuda en el desarrollo en Node. Sirviendo, así como un repositorio para publicar y compartir las herramientas desarrolladas en JavaScript, brindando la ayuda para poder gestionar, instalar y desinstalar paquetes de las dependencias para la ejecución de un proyecto.

NPX

NPX [25] es una herramienta de CLI que permite facilitar la instalación y ejecución de paquetes de npm. Npx está disponible y preinstalado con npm desde la versión 5.2.0. NPX surge con la finalidad de poder utilizar un paquete sin tener que instalarlo globalmente en el proyecto para poder ejecutarlo.

YARN

Yarn [26] es un instalador de paquetes JavaScript y gestor de dependencias lanzado por Facebook en colaboración con Google, enfocado en la seguridad y la velocidad. Yarn utiliza un registro de NPM permitiendo no realizar ninguna configuración adicional.

Herramienta de *Testing* de API REST

Se hará uso de una herramienta que permita hacer pruebas de testeo de la API a construir, para agilizar el proceso de verificación de la funcionalidad de nuestro Back-End, dicha herramienta será Postman [27].

Postman

Es una plataforma API para construir y usar API's, permitiendo realizar pruebas de testeo "HTTP *request*" simplificando y agilizando el proceso de testeo a través de crear peticiones REST, definir colecciones, establecer variables, crear *mockups* y ofrecer un entorno colaborativo para compartir las API's.

Conceptos y herramientas adicionales

Es importante destacar y describir algunos paquetes que servirán en el desarrollo del Back-End.

ORM (*Object Relational Mapping*)

Un ORM nos permite mapear estructuras de bases de datos relacionales con el objetivo de acelerar el proceso de desarrollo y brindar una cierta seguridad a la capa de acceso de los datos contra ataques. Dicho caso se hará uso para la API REST que vinculará las acciones de CRUD, en la seleccionar del ORM se hará uso de Sequelize [28].

Sequelize

Sequilize [28] es un ORM (*Object-Relational Mapping*) de Node.js basado en promesas permitiendo llamar funciones JavaScript, interactuando con distintas bases de datos como MySQL, MariaDB Microsoft SQL Server, Postgres y SQLite. Sequlize entrega característica solidas sin escribir consultas de bases de datos.

Así mismo como se hará uso de autenticaciones de usuarios, será necesario adicionar una protección de sus cuentas, dicho caso será en el cifrado de la contraseña a registrar, por lo que se hará uso de Bcryptjs [29] que permitirá encriptar la información que se considera delicada para poder almacenarla y de tal forma que no sea visible para cualquier persona aun siendo así el desarrollador de la aplicación del Front-End y Back-End.

Bcryptjs

Es un paquete escrito en JavaScript basado en el cifrado de *Blowfish*, Es una función de *hashing* de *passwords* incorporando sal para proteger ataques de *Rainbow Table*. bycript es una función adaptativa con el tiempo resistiendo a ataques de búsqueda por fuerza bruta.

También como buena práctica es recomendable hacer uso de JWT (*JSON Web Token*) [30] para establecer una cierta seguridad en la información, por lo cual describiremos en que consiste el JWT.

JWT

El JSON Web Token es un estándar (RFC 7419) que permite convertir la información de forma segura y compacta a través de convertirla en un objeto JSON utilizando un algoritmo HMAC o con claves publica/privada RSA o ECDSA es usualmente utilizada en el intercambio de información o bien en autorizaciones.

El JWT está conformado por tres partes el Header, Payload y la Firma como se muestra en la Figura 3.

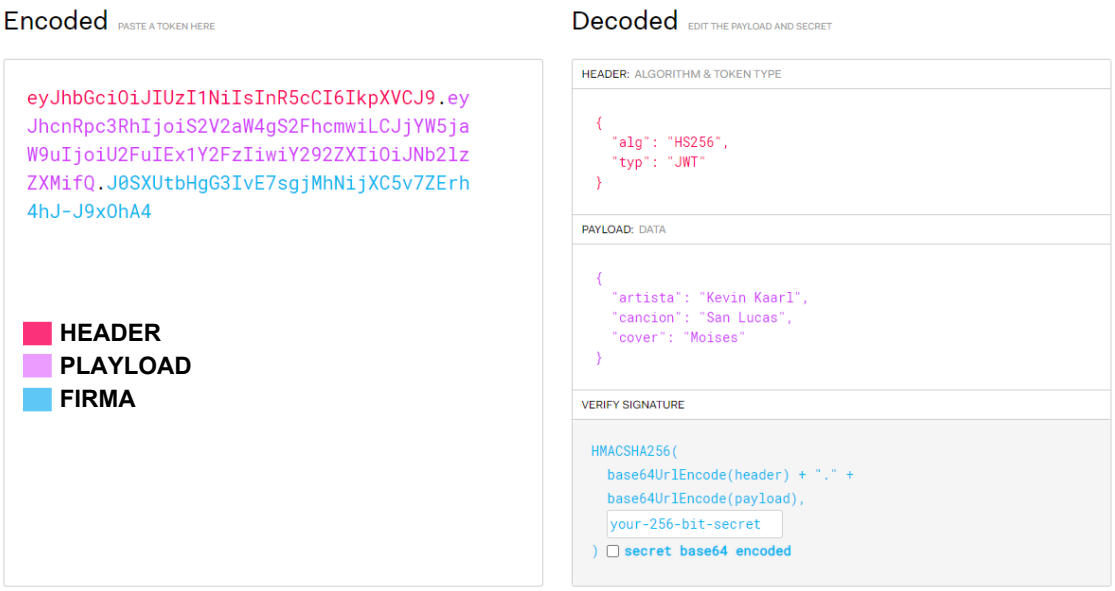


Figura 3 Estructura de un JWT

El **Header** contiene la información del algoritmo que utiliza para la encriptación con el tipo de token.

El **Payload** contiene la información que se quiere saber del token.

La **Firma** permitirá al verificador decidir si es válido o no el token.

Bibliotecas utilizadas

A continuación, en la Tabla 2 se mencionarán las bibliotecas utilizadas en el desarrollo de este proyecto.

Tabla 2 Bibliotecas utilizadas para desarrollo del proyecto

Biblioteca	Descripción
@react-navigation/native [31]	Crear la estructura de navegación en la aplicación, permitiendo navegar por varias pantallas.
react-native-safe-area-context [32]	Permite colocar el contenido de manera adecuado alrededor de la barra de estado, indicadores, elementos de la interfaz del disipativo y el sistema operativo.
native-screens react [33]	Expone los componentes del contenedor de navegación
@react-navigation/stack [34]	Proporciona hacer transiciones entre pantallas colocando en forma encimada una pantalla sobre otra a esta forma de transición también es conocida como de pila
react-native-gesture-handler [35]	Proporciona la gestión de gestos nativos para crear una mejor experiencia en las interacciones táctiles permitiendo que sean confiables y deterministas.
@react-navigation/drawer [36]	Proporciona un patrón de navegación mejor conocida como menú lateral para poder navegar entre pantallas
react-native-reanimated [37]	Proporciona una mayor flexibilidad en la interacción de gestos permitiendo crear animaciones e interacciones fluidas que se ejecutan en la interfaz del usuario.
@react-navigation/material-top-tabs [38]	Permite crear un navegador de pestañas permitiendo cambiar entre diferentes pantallas o deslizando, mostrando una transición animada montando las pantallas
react-native-tab-view [39]	Es un componente de vistas de pestañas multiplataforma

react-native-pager-view [40]	Proporciona el diseño y gestos para el desplazamiento entre las pantallas estilo carrusel
react-native-vector-icons [41]	Proporciona una biblioteca de iconos
react-native-permissions [42]	Proporciona una API de permisos unificados
react-native-responsive-screen [43]	Proporciona tres métodos para codificar los elementos de la interfaz de usuario para adaptar el contenido a distintos dispositivos
react-native-responsive-fontsize [44]	Permite mantener responsivo del tamaño de la fuente
@react-native-community/checkbox [45]	Proporciona un componente chekbox
react-native-maps [46]	Proporciona un componente de un mapa virtual
@react-native-community/geolocation [47]	Permite obtener la ubicación del dispositivo móvil con seguimiento
react-native-android-location-enabler [48]	Permite reconocer si la configuración de la ubicación del dispositivo esta habilitada
Axios [49]	Facilita el envió de solicitudes HTTP asíncronas a puntos finales REST y realiza operaciones CRUD
@react-native-async-storage/async-storage [50]	Es un sistema de almacenamiento clave-valor no cifrado, asíncrono y persistente.
react-native-element-dropdown [51]	Proporciona un componente de una lista despegable que permite seleccionar varios elementos

Desarrollo del Proyecto

Para el desarrollo del proyecto se comenzará con una estrategia basada en los objetivos, para seguir una metodología de implementación priorizando diversas actividades las cuales se dividen en 4 módulos como se muestra en la Figura 4.

Figura 4 Diagrama de secuencia de la aplicación VoyContigo

Módulo de creación y configuración de alertas.

Este módulo permitirá al usuario poder crear una cuenta, registrándolo en una base de datos, para dar el ingreso a la aplicación, donde podrá gestionar y configurar su usuario como sus alertas, dadas las acciones de:

- Registrarse: El usuario podar hacer uso de la aplicación accediendo con un usuario por el cual será necesario, registrarse con sus datos personales.
- Editar información de usuario: Se podrá modificar alguna información del usuario como correo electrónico, nombre, cambio de contraseña. etc.
- Agregar Alerta: El usuario podrá determinar qué tipo de alerta que quiera adicionar basado a tres colores (verde, amarrillo y rojo) que determinan un grado de peligro.
- Configurar Alerta: El usuario podrá gestionar y configurar los contactos referentes a el tipo de alerta que recibirán.
- Editar Alerta: El usuario podrá cambiar la configuración de la alerta.

A través de una sección de la interfaz gráfica. Se asignará un apartado específico para cada acción a realizar, haciendo así más amigable la interacción del usuario con la aplicación.

Módulo de emisión de alertas

Este módulo permitirá al usuario poder emitir las alertas a través de obtener su ubicación en el momento que sea accionada a través los botones ya configurados respectivamente con el asunto en el que se encuentra asignado en el tipo de alerta seleccionada hacia los contactos respectivos de dicha configuración.

Módulo de visualización del mapa virtual.

Este módulo estará encargado de poder visualizar un mapa virtual en un apartado de la aplicación, en el cual se llevará a cabo la acción de rastreo utilizando el mapa gráfico, en el que se asignaran algunos componentes de la identificación geográfica del usuario para el rastreo en tiempo real y la implementación de un mapa virtual de análisis para el módulo de contador de alertas.

Módulo contador de las alertas

Este módulo permitirá contar las alertas emitidas, adicionándolas en el mapa virtual de análisis de las zonas que mostrarán los contadores de alertas, dada la emisión de las dichas alertas dependiendo el asunto y el grado de peligro a partir de las alertas de los usuarios, teniendo un registro del acontecimiento en un intervalo de tiempo, dada la zona donde fueron emitidas las alertas y el asunto de emisión, haciendo así que más usuarios logren identificar la incidencia de alertas dependiendo el tipo de actos presentados, identificándolos en el mapa, para tomar medidas preventivas y de seguridad en su transcurso diario.

Una vez definido las tareas de los módulos correspondientes se seguirá la estrategia de desarrollo primordial para cumplir con dichas actividades en base al análisis de los módulos y los requerimientos del sistema. Se comenzará a diseñar el modelado de la planificación del sistema como son los apartados de la interfaz gráfica, componentes y navegaciones, del mismo modo diseñar el modelado he implementación de la base de datos a partir de la identificación de los requerimientos.

Planificación Interfaz Gráfica I

Para el desarrollo de la interfaz gráfica se desarrollará un bocetado del diseño prototipo como se muestra en la Figura 5.

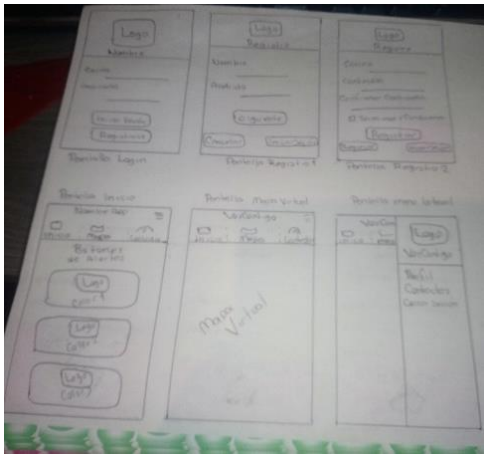


Figura 5 Bocetado de la interfaz grafica

Una vez basando en las pantallas del prototipo se genera un árbol de navegación como se muestra en la Figura 6 que servirá para desarrollar el código en React-Native.

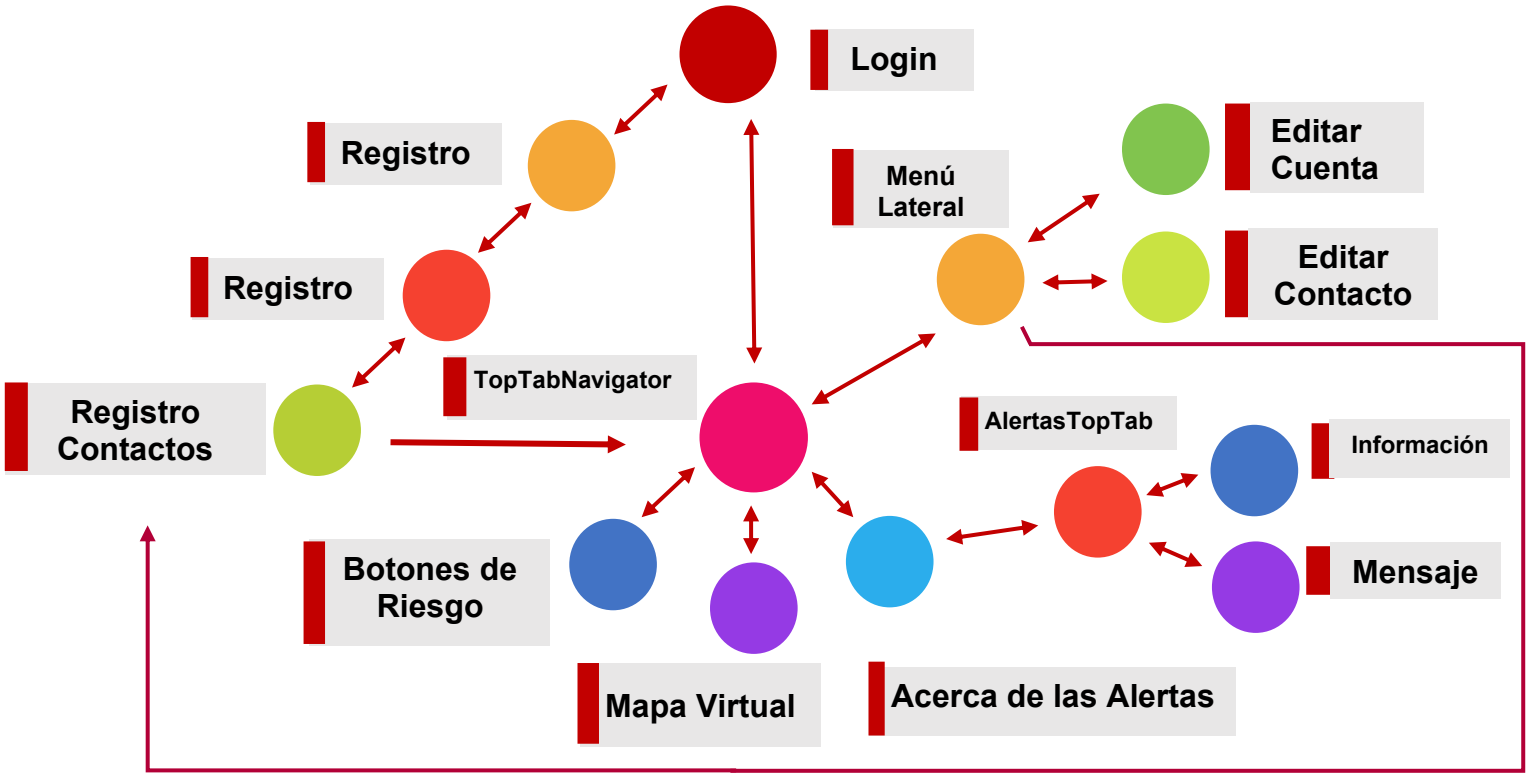


Figura 6 Diagrama de Navegación

Desarrollo de la base de datos

Antes de comenzar a implementar el Back-End de la misma forma se empleará el modelado de la base de datos, donde se identificará las tablas y atributos que describirá en la base de datos.

En la Figura 7 se muestra el modelo relacional que implementara la base de datos en MariaDB.

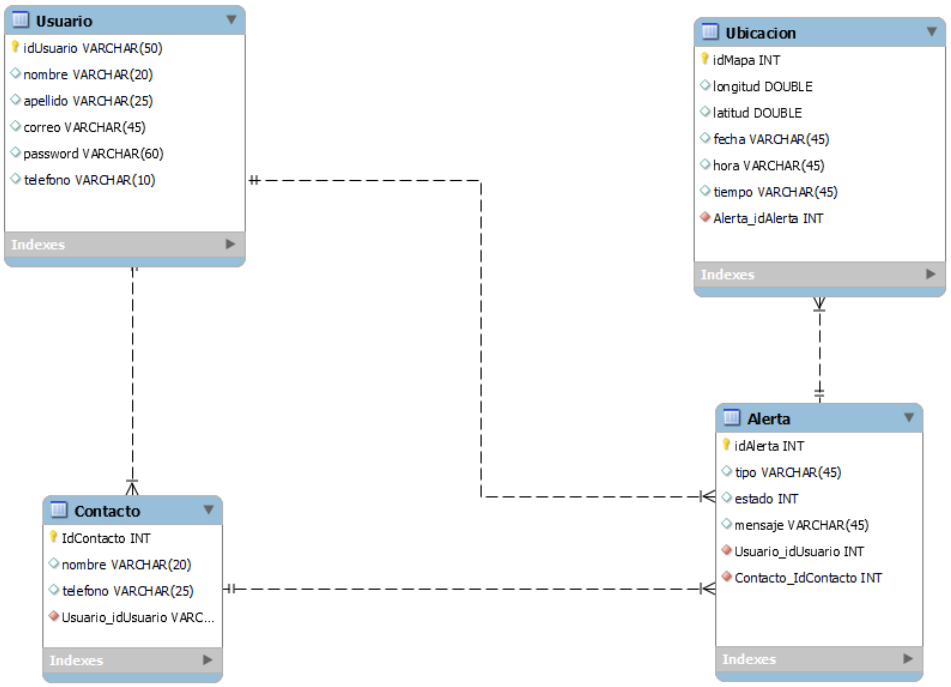


Figura 7 Diagrama del diseño relacional de la Base de datos

Una vez modelado la base de datos se procederá a la creación de la misma en MariaDB. A continuación, se mostrará implementación los siguientes scripts SQL basados en la Figura 7.

Script Creación de la base de datos

```
CREATE DATABASE IF NOT EXISTS `voycontigo` /*!40100 DEFAULT CHARACTER SET latin1 */;
USE `voycontigo`;
```

Código 1 Creación de la base de datos

Como primer paso se creará la base de datos con el nombre de "***voycontigo***" como se muestra en Código 1.

Posteriormente se crearán las tablas del modelo.

Creación de la tabla Alerta

```
CREATE TABLE IF NOT EXISTS `alerta` (
  `idAlerta` int(11) NOT NULL AUTO_INCREMENT,
  `tipo` varchar(50) DEFAULT NULL,
  `estado` int(1) unsigned NOT NULL DEFAULT 0,
  `mensaje` varchar(100) DEFAULT NULL,
  `Contacto_IdContacto` int(11) DEFAULT NULL,
  `Usuario_idUsuario` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`idAlerta`),
  KEY `Contacto_IdContacto` (`Contacto_IdContacto`),
  KEY `FK_alerta_usuario` (`Usuario_idUsuario`),
  CONSTRAINT `FK_alerta_contacto` FOREIGN KEY (`Contacto_IdContacto`) REFERENCES `contacto` (`idContacto`) ON DELETE CASCADE,
  CONSTRAINT `FK_alerta_usuario` FOREIGN KEY (`Usuario_idUsuario`) REFERENCES `usuario` (`idUsuario`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=112 DEFAULT CHARSET=latin1;
```

Código 2 Creación de la tabla Alerta

Se creará una tabla llamada ***Alerta*** con los atributos *idAlerta*, *tipo*, *estado, mensaje*, *Contacto_idContacto* y *Usuario_idUsuario*, el cual se le asignará a el atributo *idAlerta* la llave primaria, como también se les asignará las llaves foráneas a *Contacto_idContacto* y *Usuario_idUsuario* como se muestra en el Código 2.

Creación de la tabla Contacto

```
CREATE TABLE IF NOT EXISTS `contacto` (
 `idContacto` int(11) NOT NULL AUTO_INCREMENT,
 `nombre` varchar(20) DEFAULT NULL,
 `telefono` varchar(25) DEFAULT NULL,
 `Usuario_idUsuario` varchar(50) DEFAULT NULL,
 PRIMARY KEY (`idContacto`),
 KEY `FK_contacto_usuario` (`Usuario_idUsuario`),
 CONSTRAINT `FK_contacto_usuario` FOREIGN KEY (`Usuario_idUsuario`)
REFERENCES `usuario` (`idUsuario`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=74 DEFAULT CHARSET=latin1;
```

Código 3 Creación de la tabla Contacto

Posteriormente se creará una tabla llamada "***Contacto***" con los atributos *idContacto, nombre, telefono, idUsuario* y *Usuario_idUsuario*, el cual se le asignará a el atributo *idContacto* la llave primaria, así mismo se le asignará la llave foránea *Usuario_idUsuario* como se muestra en el Código 3.

Creación de la tabla Ubicación

```
CREATE TABLE IF NOT EXISTS `ubicacion` (
 `idUbicacion` int(11) NOT NULL AUTO_INCREMENT,
 `latitud` double DEFAULT NULL,
 `longitud` double DEFAULT NULL,
 `fecha` date DEFAULT NULL,
 `hora` time DEFAULT NULL,
 `tiempo` int(11) DEFAULT NULL,
 `Alerta_idAlerta` int(11) DEFAULT NULL,
 PRIMARY KEY (`idUbicacion`)
CONSTRAINT `FK_contacto_usuario` FOREIGN KEY (`Alerta_idAlerta`)
REFERENCES `alerta` (`idAlerta `) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=latin1;
```

Código 4 Creación de la tabla Ubicación

Una vez creada la tabla "***Contacto***" se creará la tabla llamada "***Ubicación"*** con los atributos *idContacto, nombre, telefono, idUsuario* y *Usuario_idUsuario*, en el cual se le asignará la llave primaria a el atributo *idContacto*, del mismo modo se le asignará la llave foránea *Usuario_idUsuario* como se muestra en el Código 4.

Creación de la tabla Usuario

```
CREATE TABLE IF NOT EXISTS `usuario` (
 `idUsuario` varchar(50) NOT NULL DEFAULT 'vy0',
 `nombre` varchar(20) DEFAULT NULL,
 `apellido` varchar(25) DEFAULT NULL,
 `correo` varchar(50) DEFAULT NULL,
 `password` varchar(60) NOT NULL,
 `telefono` int(10) unsigned DEFAULT NULL,
 PRIMARY KEY (`idUsuario`),
 UNIQUE KEY `correo_constrain` (`correo`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Código 5 Creación de la tabla Usuario

Como última instancia se creará la tabla llamada "***Usuario***" con los atributos *idUsuario, nombre, apellido, correo y password, telefono*, en el cual se le asignará la llave primaria a el atributo *idUsuario* como se muestra en el Código 5.

Una vez creada la base de datos con las respectivas tablas se procederá a implementar el Back-End.

Implementación del Back-End

Para poder hacer uso de un sistema persistente donde se puedan almacenar los datos de los usuarios, consultar, modificar o eliminar, es importante crear un sistema independiente a la aplicación móvil (Front-End) ya que se evitará hacer una sobrecarga del código y que el sistema sea vulnerable. Es importante lograr que toda la lógica del servidor como métodos del manejo de la base de datos sean solo administrados por el Back-End permitiendo ser así invisible para el usuario toda la arquitectura empleada.

Para el desarrollo del Back-End se desarrolla en Node.js utilizando Express.js para implementar una serie de servicios REST (*representational state transfer*) teniendo como resultado una pequeña API REST para el manejo del servidor de la base de datos generando un CRUD (*Create, Read, Update, Delete*) de las tablas para ofrecer los métodos que serán consumidos en el Front-End.

Para la implementación en Express.js se hace uso de un ORM (mapeo de objeto-relacional) que permitirá manipular la base de datos de manera más sencilla, para este proyecto se hará uso de Sequelize.

Como primer paso se hará la conexión a la base de datos creada. Para esto se creará una carpeta llamada **db** para colocar el método del Código 6 que permitirá conectarse a la base de datos e identificar de mejor manera donde se encontrará ubicado.

db > connection.ts > default

```
1  import { Sequelize } from 'sequelize';
2
3  /* Sc crea una instancia de Sequelize donde le asignamos el nombre de la base de datos
4  el usuario la contraseña y un objeto de configuración   */
5  const db= new Sequelize('voycontigo','root','toor', {
6        /* Donde se encuentra localizada mi base de datos */
7        host: 'localhost',
8        /* que base de datos se utilizara */
9        dialect:'mariadb',
10
11
12 });
13 export default db;
```

Código 6 Configuración de credenciales para la conexión de la base de datos con Sequelize

Al tener el primer método para acceder se procederá a crear una carpeta llama ***Models*** en donde se colocarán todos los modelos de como lucirán los métodos constructores referentes a las tablas creadas de la base de datos.

Se tomará el Modelo para la tabla Usuario especificando que atributos contiene y como lucirán definiendo el tipado de cada uno.

```
import { DataTypes, Model } from "sequelize";
import db from "../db/connection";



export interface usuarioModel extends Model{
    idUsuario:string,
    nombre:string,
    apellido:string,
    correo:string,
    password:string,
    telefono:string,
```

Código 7 Creación del modelado de la tabla Usuario con Sequelize

Al tener la interfaz como se muestra en el Código 7 se comenzará a hacer el mapeo de la tabla definiendo las características de las entradas, como llave primaria, foráneas, el tipado los parámetros, como así también algunas validaciones de la entrada de los datos como se presenta en el Código 8.

```
/* Crearemos un modelo que permitira decirle como lucira la base de datos
para evitar que alguien quiera inyectar codigo sql*/
const Usuario= db.define<usuarioModel>('Usuario',{
    idUsuario : {
        type:DataTypes.STRING,
        primaryKey:true,
    },
    nombre : {
        type:DataTypes.STRING,
        validate: {
            is: /^[a-z]+$/i,
            max: 12,
            min: 4,
          }
    },
    apellido : {
        type:DataTypes.STRING,
        validate: {
            is: /^[a-z]+$/i,
            max: 12,
            min: 4,
          }

    },
```

Código 8 Definición de la tabla Usuario con el tipado de sus atributos

Como se observa en el Código 8 el tener el modelo de esta manera nos permite tener un grado de seguridad de nuestro sistema, ya que evitaría la entrada de parámetros incongruentes o inyecciones SQL.

Al hacer los respectivos modelados de las tablas se procederá a crear otra carpeta llamada ***Controler*** donde se colocará todos los métodos de las peticiones del CRUD para cada modelo.

Método GET para obtención de todos los Usuarios

El Código 9 muestra la implementación para la obtiención de todos los usuarios con el método GET devolviendo como respuesta un objeto JSON de todos los usuarios encontrados

```
export const getUsuarios=  async (req: Request,res:Response) => {

    const usuarios = await Usuario.findAll();

    res.json(usuarios)
}
```

Código 9 Método GET para obtención de todos los Usuarios

Método GET para obtener un Usuario

El Código 10 muestra la implementación de la obtención de un usuario con el método GET a partir de su identificador

```
/* obtener un usuario */
export const getUsuario= async (req: Request,res:Response) => {
    const {id}= req.params;
    const usuario = await Usuario.findByPk(id);

    if(usuario){
        res.json(usuario);
    }else{
        res.status(404).json({
            msg:`No existe el usuario buscando`
        });
    }
}
```

Código 10 Método GET para obtener un Usuario a partir de su id

Método POST para la creación de un Usuario

El Código 11 implementará cuando se mande los parámetros en el *body* de la petición a través del método POST, se obtendrá los atributos para construir un nuevo usuario.

```
export const postUsuario = async( req: Request , res: Response ) => {

    const { body } = req;
    const {password} = body;
    try {
        const totalRegistro = await Usuario.count();
        /* encriptación de la contraseña */
        const salt = bcrypjs.genSaltSync(10);
        /* Encriptación de una sola via */
        body.password=bcrypjs.hashSync(password,salt);


        const fecha= new Date();
        const usuario =Usuario.build(
            {
            idUsuario: 'voy'+totalRegistro+fecha.getDay()+fecha.getHours()+fecha.getMinutes()+fecha.getSeconds(),
            nombre: body.nombre,
            apellido: body.apellido,
            correo: body.correo,
            password: body.password,
            telefono: body.telefono
           }
            );

        /* Guardamos el usuario */
        await usuario.save();

        /* Genera el JWT */
        const token = await generarJWT( usuario.idUsuario );
        res.json({
            usuario,
            token
         })
    } catch (error) {

        console.log(error);
        res.status(500).json({
            msg: 'Erro al Registrar el Usuario. Intentelo nuevamente o comuniquese con soporte técnico'
        })
    }
```

Código 11 Método POST para crear un Usuario

En esta descripción generaremos un id a partir del número total de registros, la fecha y el tiempo, teniendo una leyenda pequeña de ‘voy’ en un inicio.

Un punto importante a resaltar como se muestra en el Código 12 es que para guardar la contraseña se hace uso de **bcrypjs** que ayudara a encriptar la contraseña ya que es un hash de una sola vía permitiendo grabar la contraseña en la base de datos cifrada, esto con la finalidad de que ni el desarrollador del Back-End como el del Front-End puedan saber la contraseña de los usuarios ya que pudiesen hacer un mal uso de esta información, también cabe mencionar que siempre se generara diferentes encriptados a pesar de que usuarios puedan tener la misma contraseña

```
/* encriptación de la contraseña */
const salt = bcrypjs.genSaltSync(10);
/* Encriptación de una sola via */
body.password=bcrypjs.hashSync(password,salt);
```

Código 12 Encriptación de la contraseña

Método PUT para actualizar un usuario

El Código 13 implementará la actualización de un registro de un usuario, se pasará el identificador a través de la petición HTTP obteniendo los parámetros ingresados en el *body* de la *request*.

```
/* Actualizar */
export const putUsuario=async (req: Request,res:Response) => {
    const {idUsuario} = req.params;
    const { body } = req;
    try {
        const usuario = await Usuario.findByPk(idUsuario);
        if(!usuario) {
            return res.status(404).json({
                msg:'No existe el usuario'
            })
        }
        if(body.password!== undefined){
           /* encriptación de la contraseña */
           const salt = bcrypjs.genSaltSync(10);
           /* Encriptación de una sola via */
           body.password=bcrypjs.hashSync(body.password,salt);

        }
        await usuario.update(body);
        /* Guardamos el usuario */
        await usuario.save();
        /* Genera el JWT */
        const token = await generarJWT( usuario.idUsuario );
        res.json({
            usuario,
            token
        });
    } catch (error) {
        console.log(error);
        res.status(500).json({
            msg: 'Error al intentar Actualizar el Usuario. Intentelo nuevamente o comuniquese con soporte técnico'
        })
    }
```

Código 13 Método PUT para actualizar un Usuario

Método DELETE para eliminar un Usuario

El Código 14 implementará la eliminación de un registro, se utilizará el identificador recibido en la petición HTTP para hacer la búsqueda de dicho usuario y en caso de tener éxito se hará la eliminación física del registro.

```
/* Eliminación */
export const deleteUsuario=async (req: Request,res:Response) => {
    const {idUsuario}= req.params;

    const usuario = await Usuario.findByPk(idUsuario);
    if(!usuario) {
        return res.status(404).json({
            msg:'No existe el usurario'
        })
    }
    const usuarioAutenticado = (req as any ).usuario ;
    /* Eliminación fisica */
    await usuario.destroy();
      /* Eliminación logica */
      // await usuario.update();
    console.log(usuarioAutenticado);
    res.json({
        msg:'El Usuario ha sido eliminado con éxito',
        idUsuario,
        usuarioAutenticado
    });
}
```

Código 14 Método DELETE para eliminación de un Usuario

Al tener los métodos implementados de los controladores por cada tabla creada, se procederá a crear las rutas que nos permitirán hacer una serie de validaciones con la finalidad de en caso de que no se cumplan no ejecuten el código del controlador innecesariamente.

Para colocar las rutas de los modelos, se procederá a crear una carpeta llamada ***Routes*** como se muestra en la Figura 8, este tendrá la finalidad de identificar de mejor manera donde se encuentran las rutas de las peticiones.

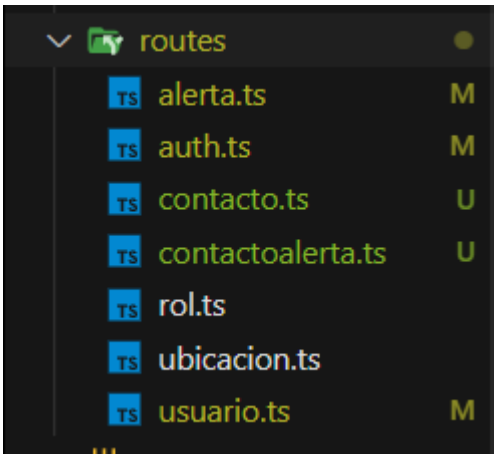


Figura 8 Creación de la carpeta Routes

Tomando como ejemplo las rutas del Código 15 para el modelo de Usuario

```
const router = Router();

router.get('/'    ,getUsuarios);
router.get('/:idUsuario',[
        check('idUsuario').custom(existeIdUsuario),
        validarCampos,
      ],getUsuario);
router.post('/',
        [
          check('nombre','El nombre es obligatorio').not().isEmpty(),
          check('password','El password debe ser mayor a 6 letras ').isLength({min:6}),
          check('correo','Eñ correo no es valido').isEmail(),
          check('correo').custom(correoExiste),
          validarCampos,
        ],postUsuario);
router.put('/:idUsuario',[
          check('idUsuario').custom(existeIdUsuario),
          validarCampos,
        ],putUsuario);
router.delete('/:idUsuario',[
          validarJWT,
          check('idUsuario').custom(existeIdUsuario),
          validarCampos,
        ],deleteUsuario);
export default router;
```

Código 15 Declaración de las rutas CRUD con middlewares

Se definen las rutas de nuestras peticiones, solo que en esta implementación se tienen *Middlewares* son funciones o métodos que nos permiten hacer validaciones antes de ejecutar las peticiones, en caso de que algunos de los atributos del *request* sean invalido arrojara un resultado con respecto a la situación en la que pueda encontrase la validación.

Como se ha mencionado para poder colocar los *Middlewares* será necesario crear los respectivos métodos para poder hacer los llamados en las rutas, es por eso que será necesario crear una carpeta llamada ***Middleware*** como se muestra en la Figura 9, para poder validar los campos.

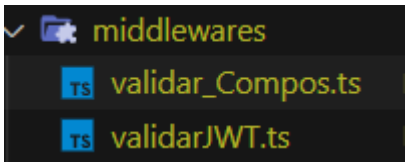


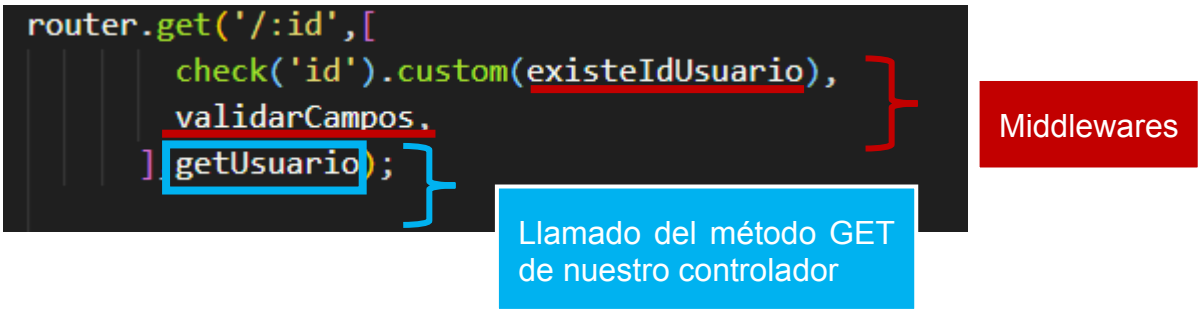
Figura 9 Creación de carpeta Middlewares

Para validar los campos se crea un archivo ***validar_campos*** donde se definirá todos los *Middlewares* disponibles para las *routes*, se mencionará algunos ejemplos como se muestra en el Código 16.

```
export const validarCampos = (req: Request,res:Response,next:any) => {
    const errors=validationResult(req);
    if(!errors.isEmpty()){
        return res.status(400).json(errors);
    }
    /* si llega a este punto sigue a otro middleware para verificar */
    next();
}
export const correoExiste = async(correo:'')=>{
    const existeEmail = await Usuario.findOne({
        where: {
            correo
        }
    });
    if (existeEmail) {
        throw new Error(`Ya existe un usuario con el correo : `+correo);
    }
}
```

Código 16 Declaración de Middlewares

Se puede ver en el Código 16 el método de validar campos, recibiendo los parámetros de la *Request* y *Response* como resultado de la acción en caso de que se cumpla o no la validación, verificando en este caso si los parámetros recibidos están vacíos o no, en caso de tener éxito se hace una función *next* permitiendo pasar a la siguiente *Middleware* hasta poder llegar a la ejecución de la petición.



Código 17 Estructura de la Ruta con Middlewares

Como se observar en el Código 17 la función GET de la ruta contempla la estructura de los *Middlewares* y en caso de haber cumplido las validaciones podrá ejecutar el método *getUsuario* para poder así mostrar el *Response* de la petición.

Configuración del Servidor

En la carpeta ***Models*** se generará una clase llamada *server*, en este archivo contendrá la lógica de la conexión a la base de datos como el acceso al llamado de las peticiones configurando las *Phats*.

```
/* Creacion de un servidor en express */
class Server {
  /* Application la obtenemos de express */
  private app: Application;
  private port: string ;
  /* Definimos nuesras rutas para los phats */
  private apiPaths = {
        usuarios: '/api/usuarios',
        alertas:  '/api/alertas',
        ubi:      '/api/ubicacion',
        rol:      '/api/rol',
        authPath: '/api/auth',
        conalerta:'/api/conalerta',
        contactos:'/api/contactos',
  }
    constructor() {
        this.app = express();
        this.port = process.env.PORT || '8000';
        /* Hacemos el llamado de la conexión de la base de datos */
        this.dbConnection();
        // Métodos iniciales
        this.middlewares();
        /* Hacemos el llamado del método routes para definir las rutas */
        this.routes();
    }
```

Código 18 Configuración de los Phats y declaración de nuestro método constructor

Como se puede observar el Código 18 se genera las *Paths* de acceso para hacer las peticiones HTTP, como también le método constructor donde se hace el llamado de las funciones de conexión con la base datos, los *middlewares*, las rutas que se han generado y el método para levantar el servidor.

.

Método constructor para conectarse a la base de datos

Se creará un método que permita acceder a la base de datos, como se mostró en el Código 6 se configuraron las credenciales que permitirán establecer la conexión, por lo que se utilizara la instancia de la configuración de sequelize para poder autenticarnos y establecer la conexión.

```
/* Se agrega un método que permita conectarse a la base de datos */
async dbConnection (){
    try {
        await db.authenticate();
        console.log('Base de datos online');
    }catch(error:any){
        throw new Error(error);
    }
}
```

Código 19 Método para conectarse a la base de datos a partir de la instancia de Sequeilze

Se utilizará una función asíncrona que permita establecer la conexión en caso de tener éxito la consola mostrara un mensaje, así mismo en caso de algún error mostrara que hubo algún problema.

Método middleware (Habilitación del CORS)

Como se ha mencionado anteriormente el *Middleware* permite ejecutar funciones antes de una ruta o procedimiento, en el Código 20 se mostrará la habilitación del CORS que nos permitirá hacer parseos del body de las rutas

```
/*Para parsear el body creamos un middleware*/
middlewares(){
    // CORS
    this.app.use(cors());
    // Parseo del body (lectura del body)
    this.app.use(express.json());
    // Carpeta publica
    this.app.use(express.static('public'));
}
```

Código 20 Método Middleware (habilitación del CORS)

Definiendo los Phats para acceso a nuestras rutas

Para poder utilizar las *Phat's* se tiene que vincular el *Phat* con las Rutas que corresponden a cada una, es por eso que es necesario crear un método indicando esa vinculación, así como se muestra en el Código 21

```
routes () {
    this.app.use(this.apiPaths.authPath,  authRoutes),
    this.app.use(this.apiPaths.usuarios,  userRoutes),
    this.app.use(this.apiPaths.alertas,   alertaRoutes),
    this.app.use(this.apiPaths.ubi,       ubicacionRoutes),
    this.app.use(this.apiPaths.rol,       rolRoutes),
    this.app.use(this.apiPaths.conalerta, conalertaRoutes),
    this.app.use(this.apiPaths.contactos, contactoRoutes)
}
```

Phat

Archivo de Configuración de la Ruta

Código 21 Definiendo los Phats para acceso a nuestras rutas

Método para poder escuchar el puerto del servidor

En el Código 22 se crea un método que permita detectar si el servidor esta levantando, mostrando el puerto de este.

```
/* Para levantar el servidor */
listen(){
    this.app.listen(this.port, ()=>{
        console.log('servidor corriendo en puerto ' + this.port);
    })
}
```

Código 22 Levantamiento del servidor

En el archivo principal de la aplicación en ***app.ts*** se hará el llamado del servidor como se muestra en el Código 23 accediendo al método listen para cuando se ejecute Express.js se levante el servidor.

```ts
app.ts > ...
1   import dotenv from 'dotenv';
2   import Server from './models/server';
3
4   /*Se hace el llamado para que lea el archivo
5   de configuración que contiene las variables de entrono */
6   dotenv.config();
7   /* Se hace una instancia del servidor */
8   const server = new Server();
9   /* accedemos al método listen*/
10  server.listen();
11
```

Código 23 Instancia de llamado del servidor

Levantamiento del servidor

Después del desarrollo del Back-End se levantará para confirmar que todo ha salido correcto.

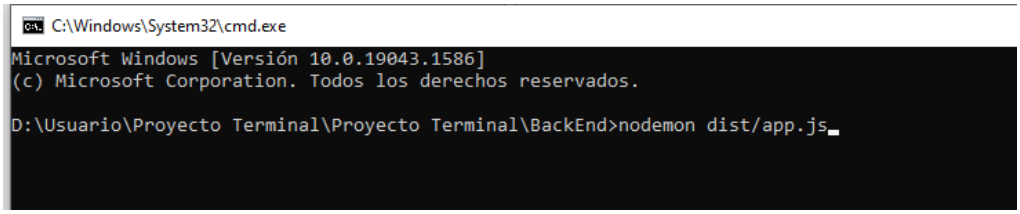


Figura 10 Comando para levantar el Back-End

Para eso se ejecutara el comando **nodemon *dist/app,js*** en la consola del símbolo del sistema, ubicándose en la carpeta del proyecto del Back-End como se muestra en la Figura 10.

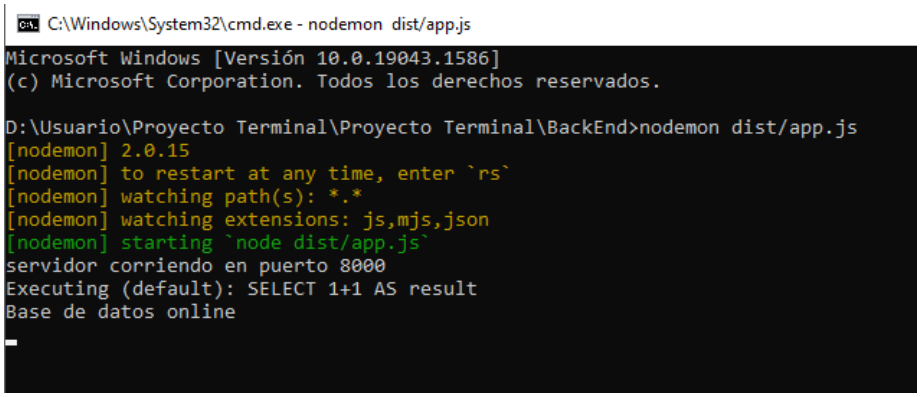


Figura 11 Servidor en ejecución

Una vez ejecutado el comando se logrará ver como en la Figura11 que el servidor se ha levantado, esperando a ser utilizado.

Pruebas de servicio REST

Para poder hacer pruebas del Back-End se hará uso de la herramienta de desarrollo de **Postman** ya que esta ayudara hacer las peticiones HTTP de los servicios que se han desarrollado.

Para poder realizar estas peticiones, será necesario configurar el ambiente para eso en la sección de “ambiente” se definirá los puertos para la conexión hacia el servidor

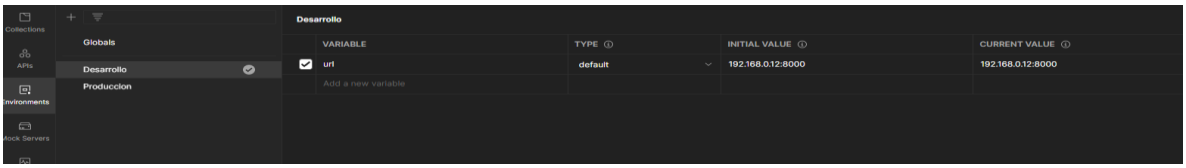


Figura 12 Configuración de puerto en Postman

Se genera unas variables globales en la cual le colocara el nombre de ***Desarrollo*** para identificar que se esta ejecutando el servicio en modo de desarrollo haciendo pruebas locales, para definir el nombre de la variable se colocara el nombre de ***url*** donde se asignara la dirección IP del dispositivo o la dirección del local host.

Una vez configurada la dirección local se generará un par de colecciones que describirán las peticiones HTTP en base a el CRUD de las tablas para determinar que funciona correctamente, para eso se definirá algunos nombres para identificar que petición u acción que se realice como se muestra a continuación en la Figura 13.

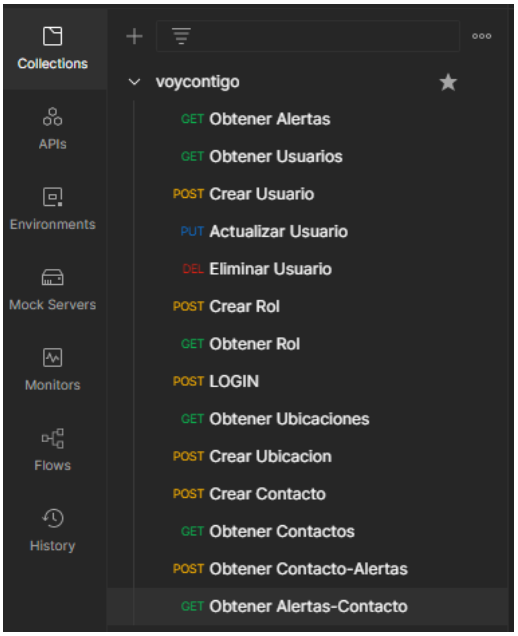


Figura 13 Colecciones CRUD

Se vera a continuación en la Figura 14 un ejemplo de una de las peticiones en Postman de una petición GET, como se dijo con anterioridad se escribe el *phat* de la petición HTTP, pero con la característica que **{{url}}** corresponde a la variable de la dirección IP del local host enseguida del *phat* que se configuro para las rutas en el Back-End

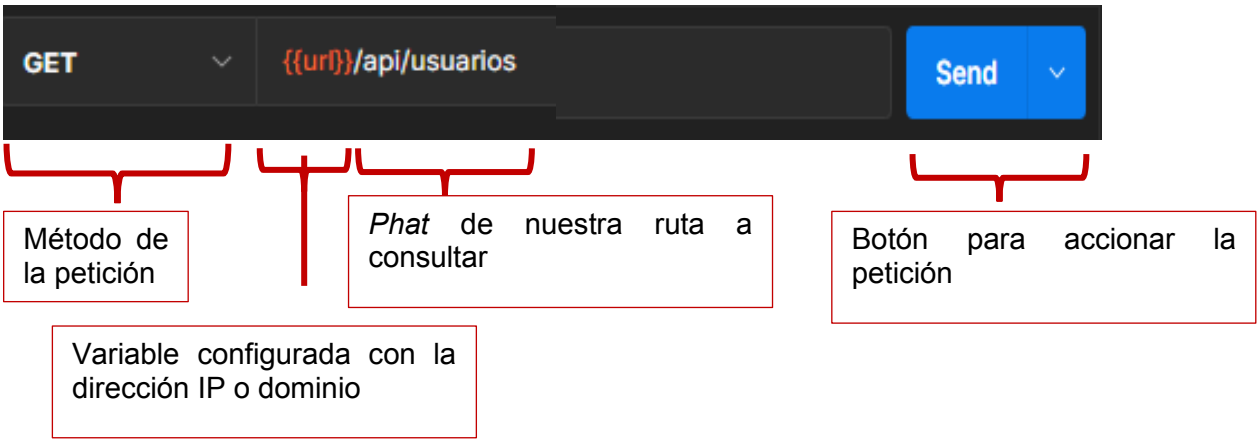


Figura 14 Phat de petición GET para Usuario

Una vez teniendo los *phats* configurados con el método de petición asignado al accionar el botón *send* se obtendrá una respuesta del servidor como se muestra a continuación en la Figura 15.

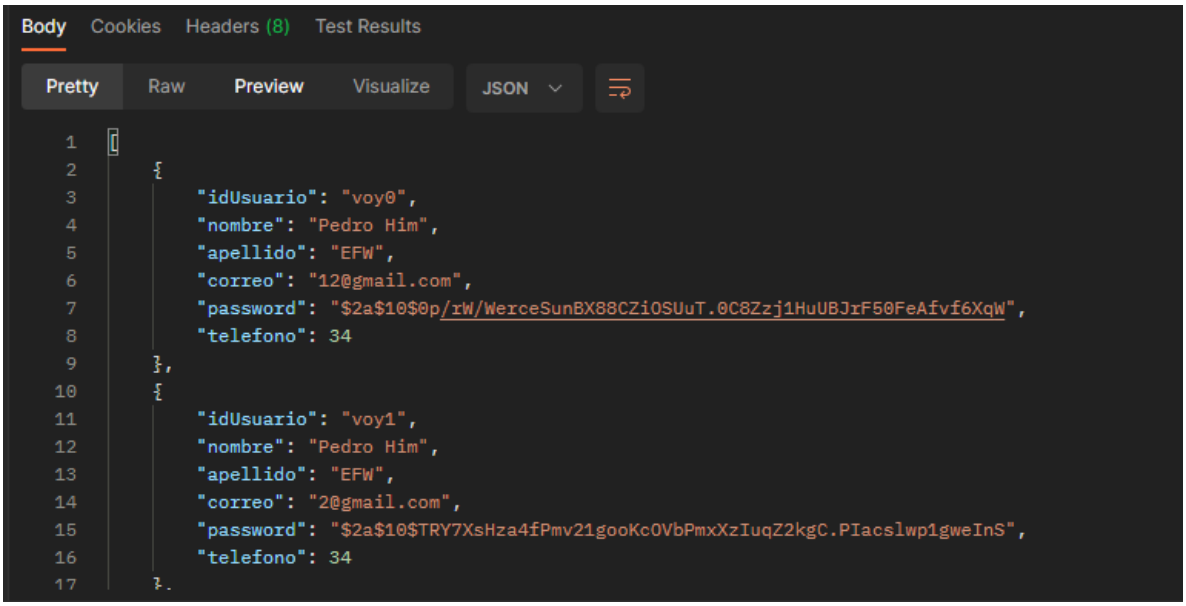


Figura 15 Respuesta método GET

Se logra ver los registros de los objetos de tipos en formato JSON

Así mismo se verá otro ejemplo en la Figura 16 con el método POST para poder crear un nuevo usuario, permitiendo mandar los parámetros a través del *body* para poder crear el usuario.

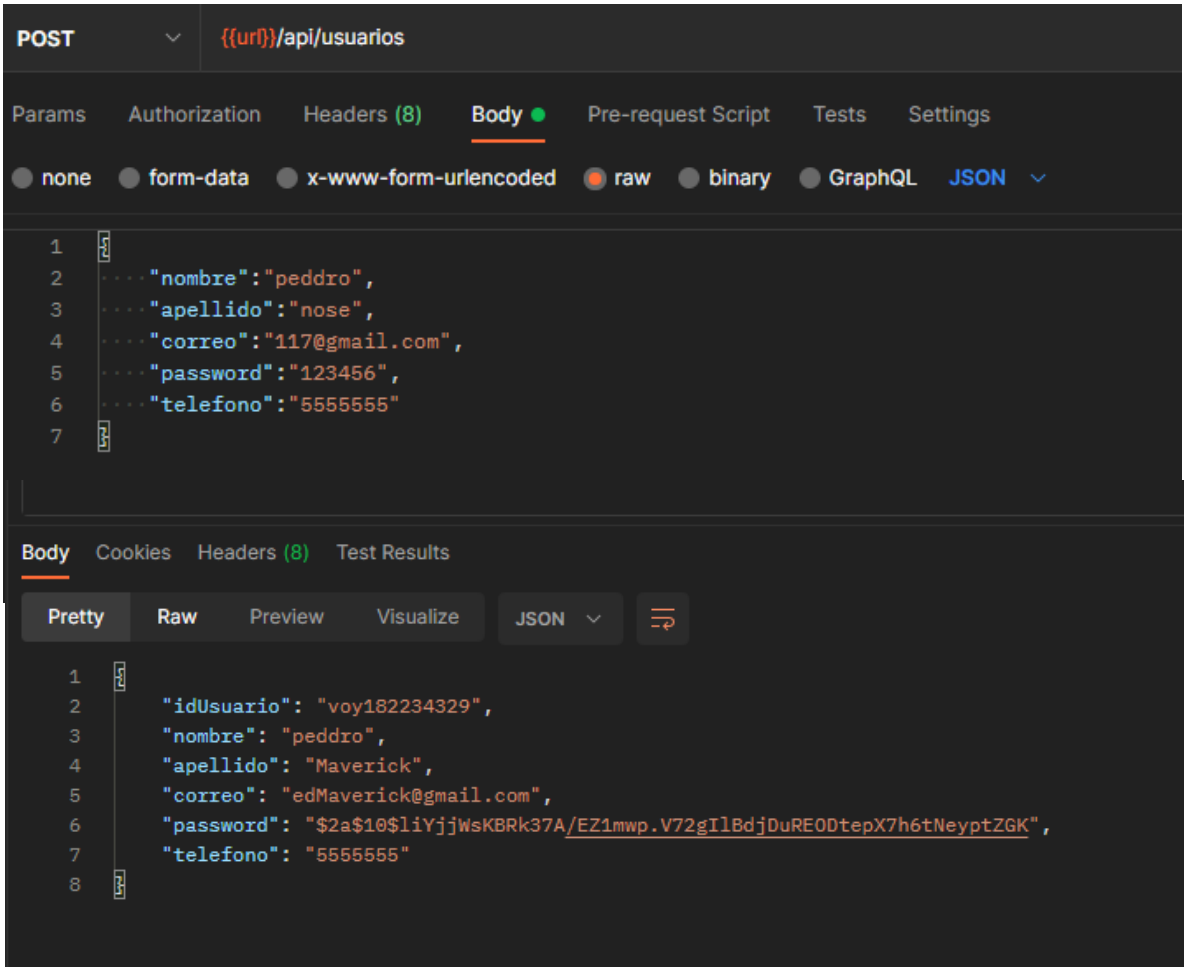


Figura 16 Petición POST para Usuarios

Donde se puede ver como se obtiene como resultado el Usuario ya regresado con su identificador y la contraseña encriptada, de esta manera también se logra corroborar si se ha registrado con éxito en la base de datos como se muestra en la Figura 17.

voycontigo.usuario: 19 filas en total (aproximadamente)

idUsuario	nombre	apellido	correo	password
voy0	Pedro Him	EFW	12@gmail.com	$2a$10$0p/rW/WerceSunBX88CZiOSUuT.0C8Zzj1Hu...
voy1	Pedro Him	EFW	2@gmail.com	$2a$10$TRY7XsHza4fPmv21gooKcOVbPmxXzluqZ2k...
voy10301634	sdfsdf	sdfsdf	12m@mtwo.com	$2a$10$tGqvqnxlx.dt/jC9aBYNkO/7Zxos.5AMM8coy...
voy11395132	sdfsdfsd	sdfsdf	asdas@gmail.com	$2a$10$tEVHJw.Ilcz6NarNnKk3WuPCiwjEuLtUQSJ6N...
voy126133417	eartyfgdd	fgdfg	mm@como.com	$2a$10$ieBci/gp5LNdwTZXxJEzaOuP..brcLxlR7etgfo...
voy130135553	asdasd	asdasdsa	ejemplo1@gmail.com	$2a$10$eEmjJY5RcGI0BTf..J59.J72S5bZn5bVetUQPp...
voy140135924	sdfsad	adfa	a1a@nose.com	$2a$10$xtDqG0Vje6nsYCBBV/JVY.VC9dyRpTcLQLloC...
voy156123621	Nose	Algo	aver@gmail.com	$2a$10$ziNhXN0iVuvLwrSQuwb2C.sQzj2eLcljNjlo5O...
voy160214711	Alexandere	Bape	alexa@gmail.com	$2a$10$lxNQ7X6ZRVlfCebMK3ogZeagTLR9KgfPYwB...
voy172185122	Moises	Juan	Moises@live.com	$2a$10$1E5bsS1HifsMthFPACO8vutn/qwXn0genUU
voy182234329	peddro	Maverick	edMaverick@gmail.com	$2a$10$liYjjWsKBRk37A/EZ1mwp.V72gIIBdjDuREODt...
voy2	jose jose	nose4	juan@gmai.com	$2a$10$AFNhGZyMzemBKKhxaCHPeCTV.qHaxmB...
voy3	peddro	nose	1@gmai.com	$2a$10$YsCZ7M5AcpX5mC.Ad2rDau8ZEzXxTye7HR/...
voy4	esd	sdf	o@gmail.com	123456
voy5	peddro	nose	1@gmail.com	$2a$10$Mrdm1K9ZQwAO7AZwN/9Dr.WGoKrL4vvP....
voy6	ahorasi	esmomento	juang@live.com	$2a$10$ORwdxulugdG.WhOteg2DpufAJHZa4dZt1U...
voy7	Junto	Jiménez	nose@live.com	$2a$10$YBRLj8f2Y09C/LJJ7qol.SrQw2W6QUBG5wxh...
voy8161156	peddro	nose	117@gmail.com	$2a$10$jHNQ9HQWPTv1hqgfGMJh1.WBPBd.Tok9EF...
voy930949	sgdgsd	sdgsdg	m2@gmail.com	$2a$10$J2zvEXt9Dx41UeC//HPEdev.Qg6152HaeQQ...

Figura 17 Verificación de registros de Usuarios

Como se puede ver en los registros de la base de datos se ha registrado con éxito.

Teniendo como herramienta Postman como base a hacer diversas pruebas para la verificación del servicio REST, donde también se puede ver el funcionamiento de las validaciones mostrando siempre una respuesta en caso de que alguna implementación haya sido incorrecta.

Como se logra ver en la Figura 18 se puede ver que se muestra la leyenda del problema que presento al intentar registrar un usuario.

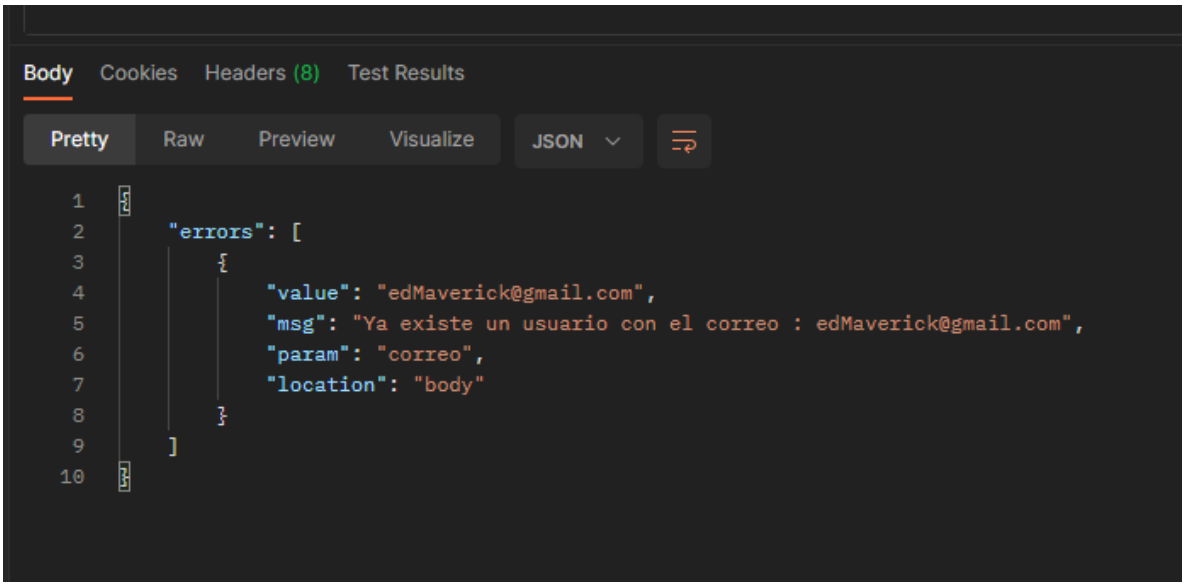


Figura 18 Respuesta de validación errónea

Implementación Front-End

Una vez obtenido el Back-End en funcionamiento se procederá a implementar el Front-End creando una aplicación móvil en React-Native, donde se configurarán los apartados en base al diseño implementado de los bocetos, secciones, navegaciones y contenido del diseño para el usuario.

Creación de la aplicación móvil con React Native

Para comenzar a desarrollar en React-Native con TypeScript comenzara con una plantilla de un proyecto generado a través de React-Native CLI por el cual ejecutara un comando en el símbolo de sistema como se muestra en la Figura 19, para poder crear la base del proyecto

npx react-native init VoyContigo --template react-native-template-typescript

Figura 19 Ejecución del comando para la creación de una aplicación en React-Native

Una vez ejecutado el comando se creará una aplicación con los componentes base para desarrollar en React-Native.

Para poder levantar la aplicación se hará uso del emulador que ofrece el IDE de Android Studio, que permite simular un dispositivo móvil Android en el cual previamente se debió haber configurado. De esta manera se ejecutará el comando siguiente para levantar la aplicación.

Npx react-native run-android

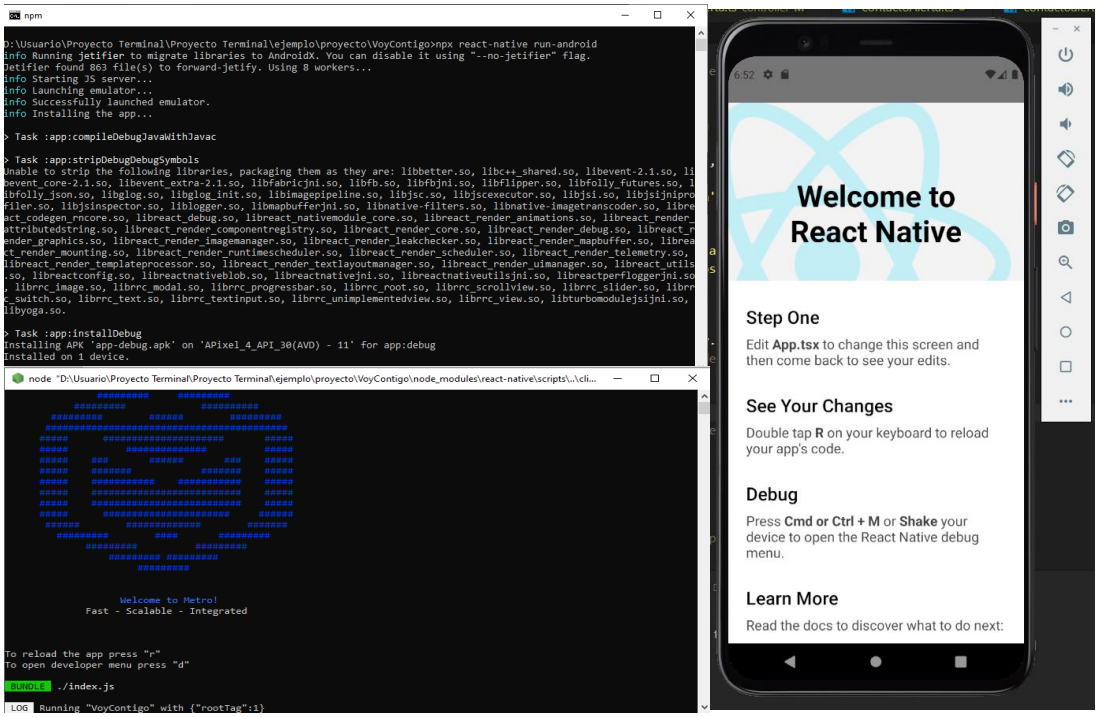


Figura 20 Resultado de la ejecición del comando npx react-antive run-android

Al ejecutar el comando se levantará Node y abrirá el emulador de Android mostrando la aplicación plantilla creada como se muestra en la Figura 20. De esta forma se puede verificar que se ha creado con éxito la base para poder desarrollar la aplicación en React-Native

Implementación de la interfaz grafica

Para la implementación de la interfaz gráfica se basará en el diseño propuesto en de los bocetos y la navegación de las secciones que contendrá la aplicación.

En este desarrollo se comenzará a armar los componentes de cada pantalla que contendrá la aplicación, donde se asignará un diseño responsivo, con la finalidad de que sea adaptable a cualquier tipo de dispositivo sin importar las dimensiones con las que cuente, como se muestra en la Figura 21.

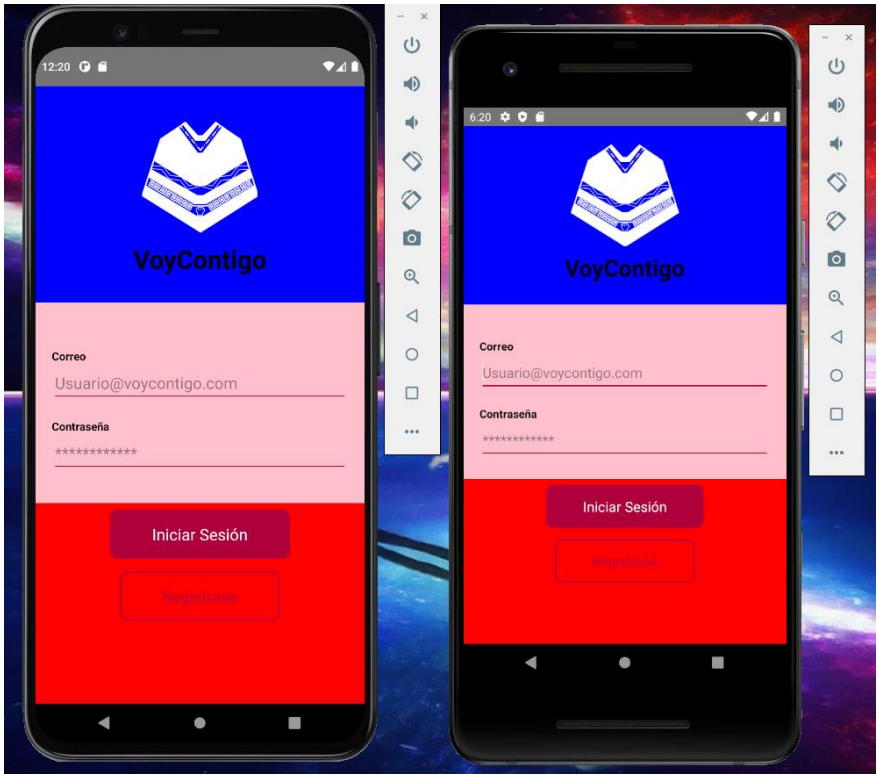


Figura 21 Visualización del diseño responsivo con dimensiones distintas de dispositivos

En el desarrollo de la aplicación se asignará una configuración del tema para un fácil diseño de los componentes reciclando algunas características determinando los colores, tamaño de la fuente, dimensiones de botones, espacios de trabajo, imágenes iconos, etc.

Así mismo se comenzará a configurar la navegación que contendrá la aplicación asignando los distintos efectos de navegación aplicándolas a las siguientes pantallas.

Stack Navigator

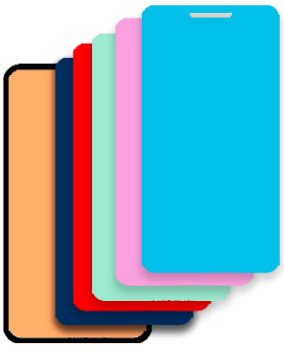


Figura 22 Ejemplificación modelo navegación tipo stack

Implementa la navegación en forma de pila (Figura 22) para la transición entre pantallas donde ofrece una serie de métodos para empujar o sacar pantallas, este tipo de navegación carga las pantallas conforme vallan siendo solicitadas cargando y manteniendo el contenido durante el uso de esta navegación de pantallas, sin embargo, también contiene métodos que permiten reemplazar la navegación de dichas pantallas.

Las pantallas de la aplicación que contendrán esta navegación son:

- *Login*
- Sección de Registro
- Menú Lateral Interno
- Configuración de Contactos
- Configuración de Cuenta

Material Top Tabs Navigator

Figura 23 Ejemplificación Tob Tabs Navigator

Este tipo de navegación implementa un menú en la parte superior o inferior del dispositivo como se muestra en la Figura 23 donde se pueden contener secciones permitiendo cargar el contenido de las pantallas en dichas secciones, permitiendo accionar una animación al deslizar la pantalla o tocar las pestañas de algunas de las secciones.

Esta navegación se implementa en nuestra página principal donde se mostrarán tres secciones donde se encontrarán las alertas, el mapa virtual y el apartado de “Acerca de las alertas”.

De la misma manera Se hará la implementación para mostrar la información de las características del contenido de las alertas como sugerencias de uso y el contenido del mensaje que se enviará.

Drawer navigation

Figura 24 Ejemplificación Drawer Navigator

Este tipo de navegación nos proporciona el cargar un contenedor de lado izquierdo o derecho sobre la pantalla cargada como se muestra en la Figura 24, usualmente este tipo de navegación se emplea para colocar un menú estilo hamburguesa donde se acciona y muestra el contenido accionando una animación de despliegue de algún costado de la pantalla de dispositivo

En la aplicación se hace al cargar el contenido del *Top Tab Navegador* para que muestre esa pantalla como principal y en la parte del Encabezado de la pantalla de inicio se colocara un icono de hamburguesa para que sea identificable como un menú, que será despegable al accionar dicho botón

Este tipo de navegación tiene la característica de mostrarse como un componente padre encapsulando las pantallas que integren su navegación.

Pantallas

Una vez definida la navegación y teniendo el diseño del tema de la aplicación, se comenzará a desarrollar la interfaz de pantallas correspondientes, las cuales serán las siguientes:

- *Login*
- Registro (nombre y apellido)
- Registro (correo y contraseña)
- Configuración de contactos
- Inicio (activación de alertas)
- Mapa virtual
- Configuración Perfil
- *Loading*

Login

Como primera instancia se desarrolla la pantalla de *Login* la cual contendrá un formulario que permitirá autenticar a un usuario registrado respectivamente en dicha aplicación, donde así mismo contendrá un botón que permitirá hacer la navegación hacia la pantalla de *Registro* como se muestra en la Figura 25.

Figura 25 Pantalla Login

Para dicho desarrollo de la Figura 25 en el proyecto de React-Native se creará una carpeta llamada *screens* como se muestra en la Figura 26.

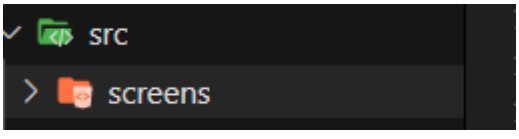


Figura 26 Creación carpeta screens

En dicha carpeta se creará un archivo *TSX* llamado *LoginScreen.tsx* (Figura 27) en el cual contendrá el código de la pantalla *Login.*

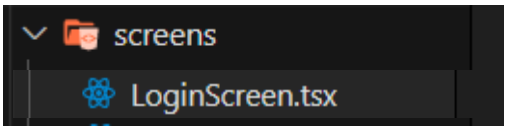


Figura 27 Creación archivo LoginScreen

En dicho archivo se desarrollarán los componentes de la interfaz, donde es importante identificar los elementos de la pantalla para un mejor desarrollo de esta como se muestra en la Figura 28.

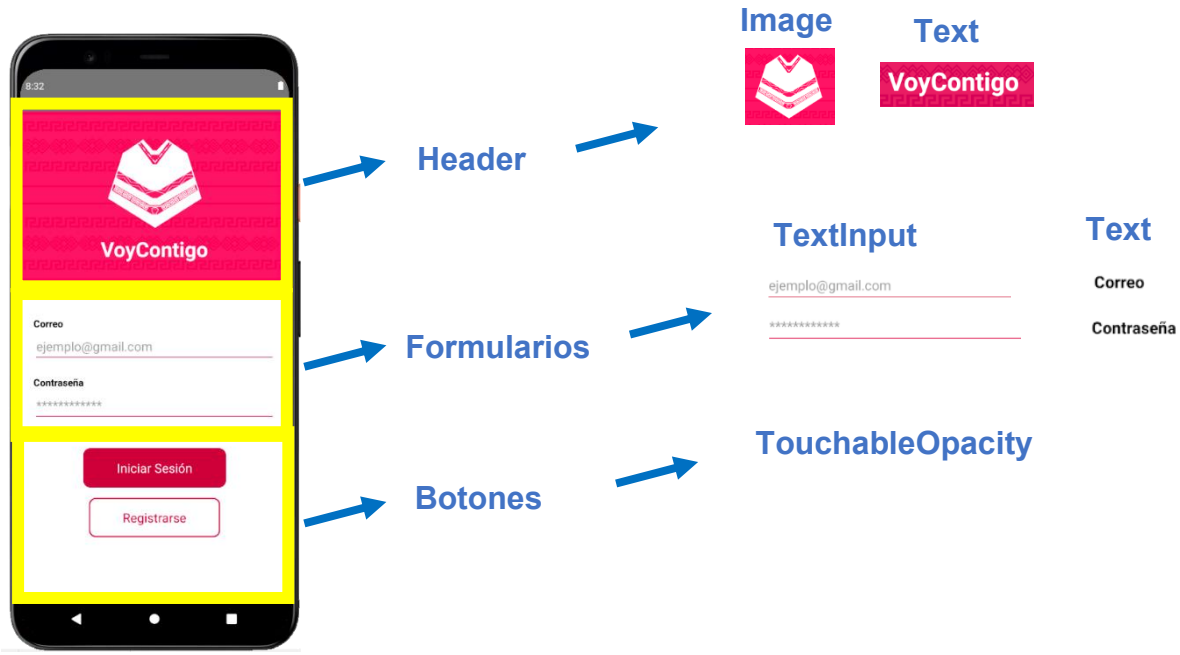


Figura 28 Identificación de componentes de la pantalla LoginScreen

Registro

Para la sección registro se dividirá en dos pantallas como se muestra la Figura 29, en las cuales seguirá una misma estructura con la finalidad de no saturar visualmente el contenido y la interacción con el usuario más amigable y entendible el requerimiento de los datos.

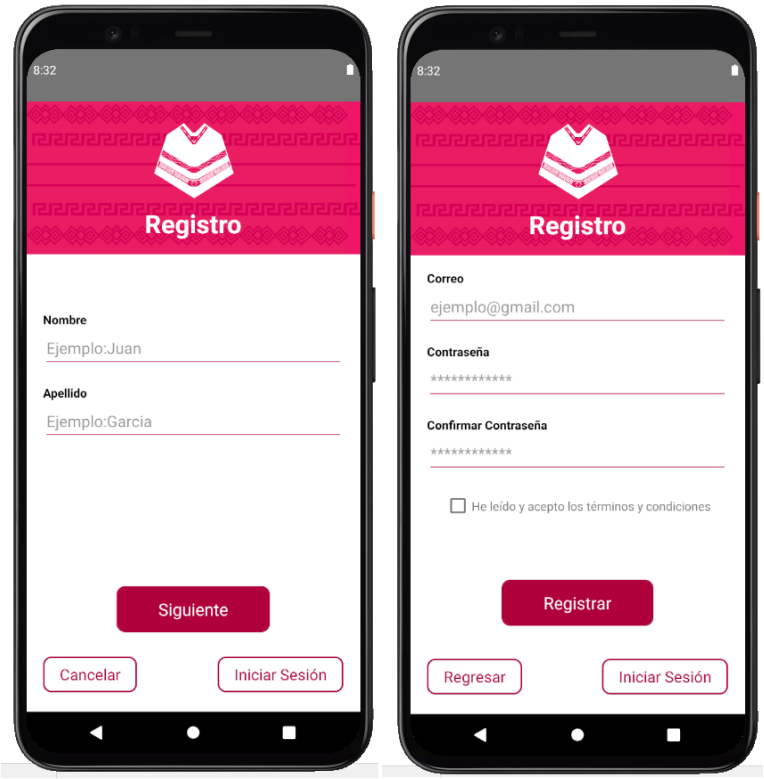


Figura 29 Pantallas Registro

Para la creación de las dos pantallas sera necesario crear dos atchivos *TSX* en la carpeta *screens* donde dichos archivos seran llamados *RegistroScreen.TSX* y *RegistroScreen2.tsx.*

Una vez creado los archivos se procedera a la identificación de los componentes como se hizo para la pantalla Login.

Configuración de Contactos

En la sección de configuración de contactos la pantalla (Figura 30) a desarrollar representara varios componentes donde se pretende mantener la esencia de la sección, sin embargo, representara un acceso distinto, ya que algunos componentes de la pantalla se mostrarán solo cuando el usuario se registre por primera vez, posteriormente a dicho registro solo mostrara la parte importante de configuración de contactos, añadiendo un apartado de navegación.

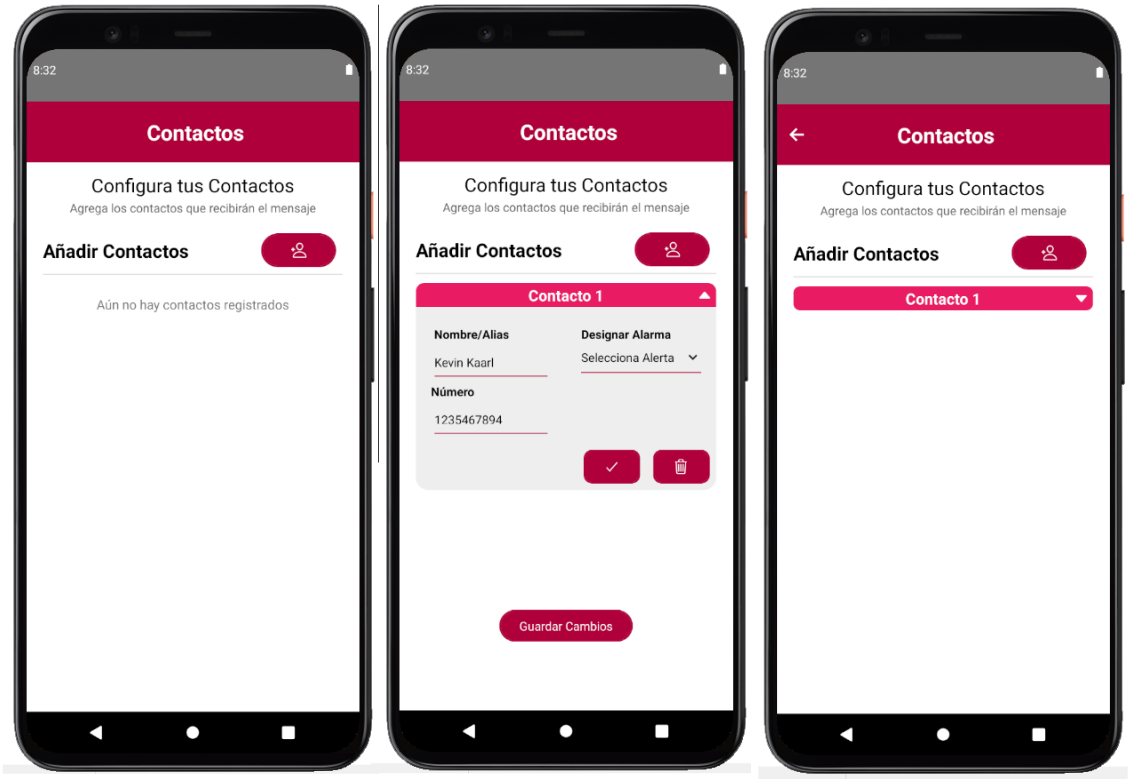


Figura 30 Pantalla Configuración de Contactos

Para esta pantalla se creará un archivo TSX llamado *ContactosScreen.tsx*, posteriormente se analizará los componentes que llevara, para de esta manera focalizar los elementos que se tendrán que ocultar o mostrar.

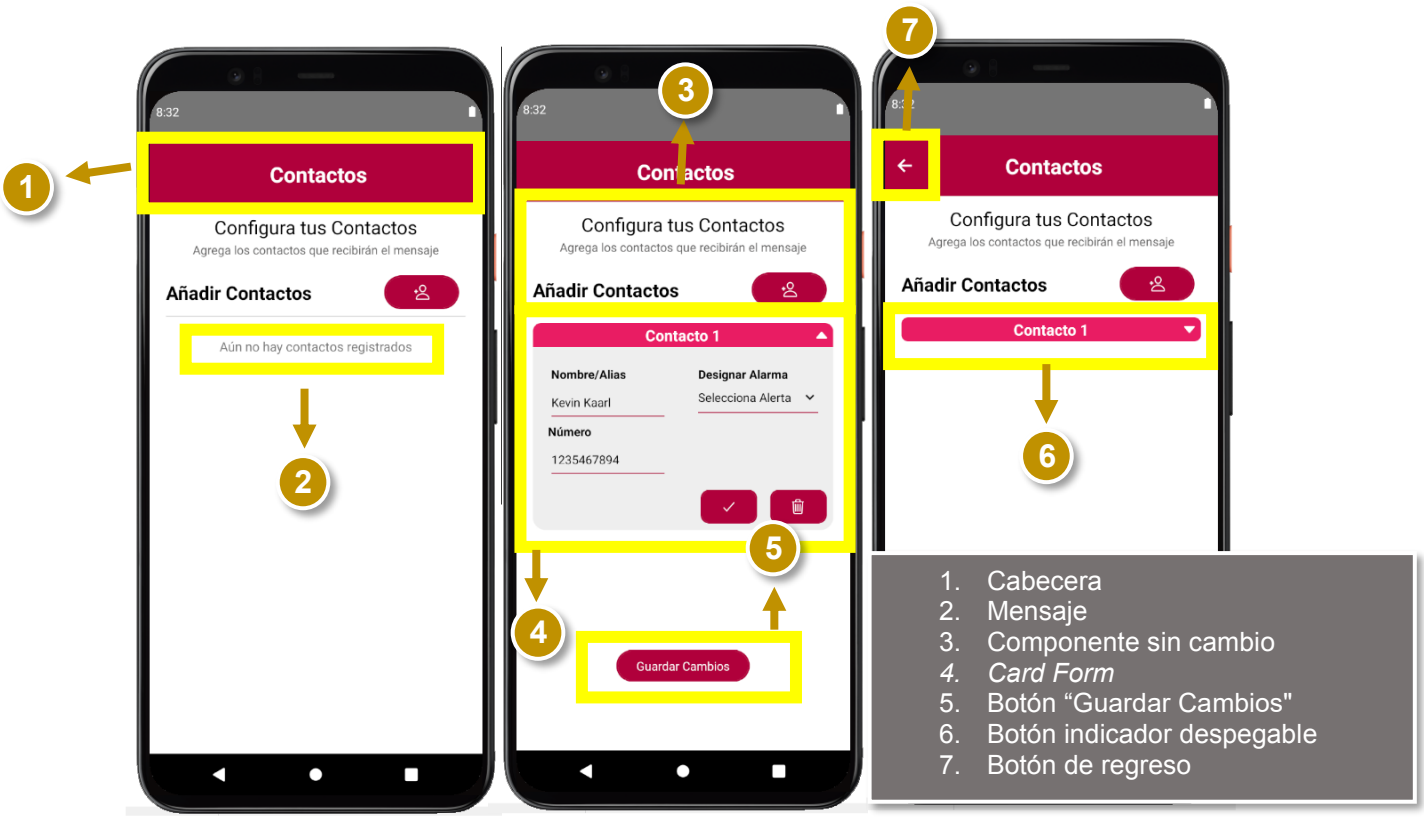


Figura 31 Ubicación de componentes pantalla Configuración de contactos

En la Figura 31 se puede destacar y ubicar los componentes importantes a desarrollar. Como primera instancia se ubicarán los componentes del despliegue posterior a el registro del usuario, donde habrá una primera configuración directa de los contactos.

Se puede observar en la **Cabecera** de la pantalla que tiene un cambio, el cual tiene un **botón** de navegación, este solo se mostrará cuando el usuario ya haya pasado por el proceso de registro por primera vez, ya que esa navegación permitirá regresar a la pantalla de **inicio**. También se observa que tiene un botón de "**guardar cambios"** ese botón permitirá salvar los cambios una vez agregado algún contacto teniendo la característica que dará paso a ocultar los componentes 5 dando paso a mostrar el componente 7.

En el componente 3 se mantendrá la misma información, ya que permite identificar como asignar un nuevo contacto, donde esa asignación se hará a través del botón con el icono de "agregar", donde se pretende abrir la agenda de contactos del dispositivo.

El componente 6 se desarrollará con la finalidad de hacer más sencilla la identificación de los contactos que se deseen agregar, donde se permitirá hacer un despliegue de la información (componente 4) de algún contacto asignado, el cual contendrá botones para la modificación, selección y eliminación de algún registro.

Encabezado – *Top Tab Navigator* y *Drawer navigation*

Antes de comenzar a desarrollar las pantallas faltantes, se crearán componentes de navegación que permitirán darle un diseño accesible a los apartados importantes, manteniendo un orden de contenido.

Como primera instancia se analizará el diseño para determinar que componentes desarrollar primero.

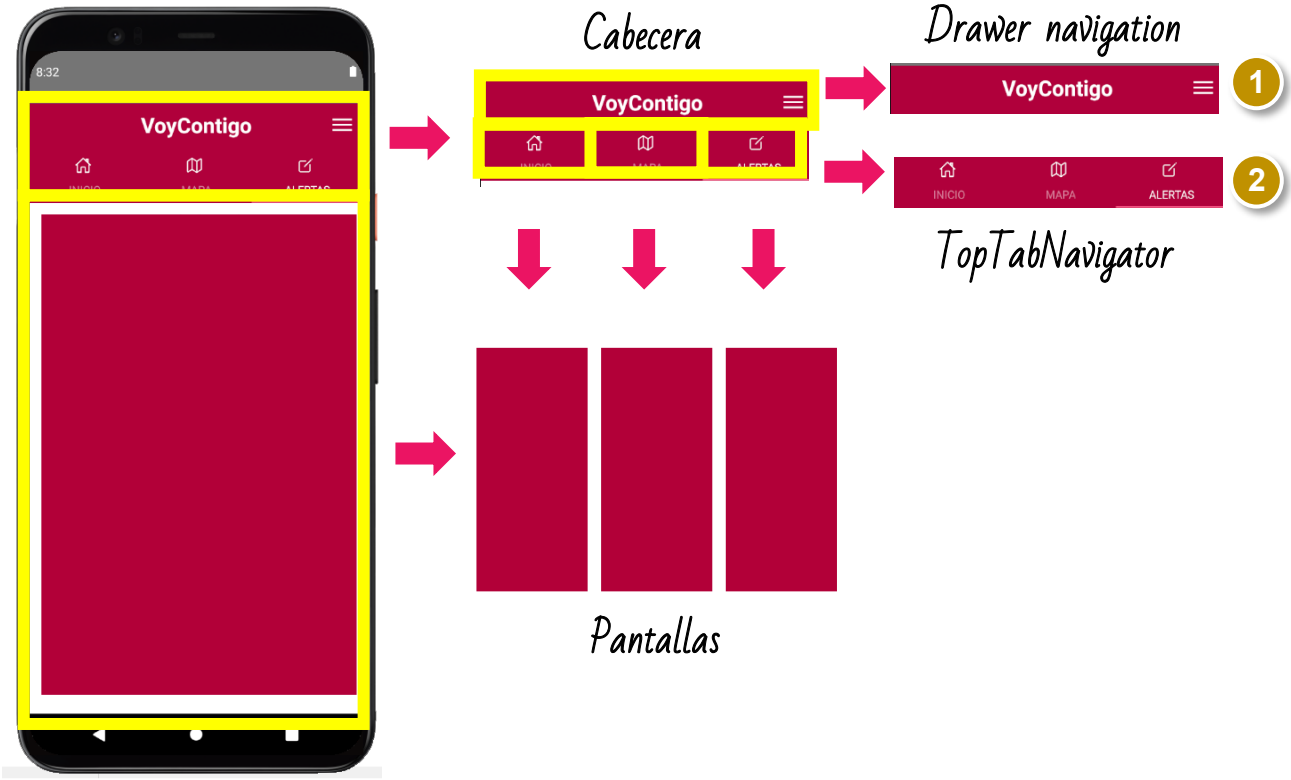


Figura 32 Análisis de navegación Drawer Navigation y Top Tab Navigator

Como se muestra en la Figura 32 en la parte superior se tiene una cabecera dicha cabecera se compone de el *Drawer Navigator* y *el Top Tab Navigator.* Como primera instancia se desarrollara el Drawer Navigator ya que este encapsulara el contenido permaneciendo como componente Padre, de tal manera que al ser cargado todo el contenido Hijo siempre permanecera el encabezado 1 permitiendo que se pueda mostrar un menu como se muestra en la Figura 33.

Drawer Navigator

Figura 33 Pantalla Menú Lateral

Siendo así para crear el *Drawer Navigator* se creará una carpeta llama ***Navigator*** donde en dicha carpeta se creará un archivo TSX llamado **MenuLateral.tsx** como se muestra en la Figura 34.

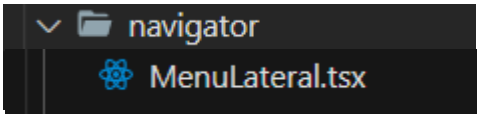


Figura 34 Creación de archivo Manu Lateral

Para poder desarrollar los componentes se harán las respectivas configuraciones para poder activar el menú lateral y a su vez crear el encabezado que se mostrara en la aplicación.


```
const Drawer = createDrawerNavigator();
export const MenuLateral = () => {
    const {width} =useWindowDimensions();
    return (
        <Drawer.Navigator
                        screenOptions={{
                            drawerType: width >= 768 ? 'permanent' : 'front',
                            drawerPosition:'right',
                            headerShown: false,
                            headerStyle: {
                              backgroundColor: colors.primaryDark,
                            },
                            headerTintColor: '#fff',                    //Asignamos
                            headerTitleStyle: {
                              fontWeight: 'bold',
                            },
                            headerTitleAlign:'center',
                            headerLeftContainerStyle:{ backgroundColor:'white'},
                        }}
  /* Le pasamos el componente que tendrá el menú lateral y mostrara el contenido */
  drawerContent={(props)=><MenuInterno {...props}/>}>
    {/* Determinamos la pagina que se mostrara por defecto */}
  <Drawer.Screen name="TopTabNavigator" component={TopTabNavigator}
      options={{
        title: 'VoyContigo', //Le asignamos el titulo que mostrara la pagagína en el Header

      }}
  />
```

Código 24 Creación del Menú Lateral

Como se muestra en el Código 24 para configurar el *Drawer Navigator* es necesario hacer una configuración para poder indicar sobre que componentes se establecerá la navegación, siendo así, se establecerán las pantallas como hijas en la hora de hacer el llamado del Menú Lateral.

La pantalla que se muestra al accionar el menú tipo hamburguesa se desarrollara como una pantalla normal, ya que lo único que cambia son las dimensiones en las que se mostrara como se puede observar en la Figura 33. Para la creación de la pantalla se declarará en el mismo archivo *MenuLateral.tsx* pero estableciendo un método llamado *MenuInterno.*

Top Tab Navigator

Figura 35 Creación Top Tab Navigator

Una vez configurado el *Drawer Navigator* se procederá a crear otra navegación que permitirá acceder a tres pantallas. Este tipo de navegación establece la misma idea de encapsular el contenido teniendo un Componente Padre a uno Hijo, es decir que el encabezado se mantendrá en toda la navegación por las pantallas.

Para la configuración de esta navegación requiere hacer del mismo modo hacer una configuración respectiva. En la carpeta *Navigator* se creará un archivo TSX llamado TopTabNavigator.tsx (Figura 36).

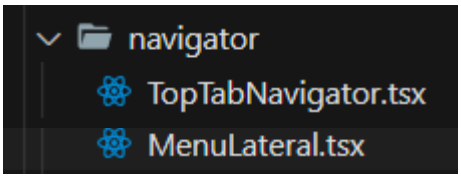


Figura 36 Creación del archivo TopTabNavigator.tsx

Una vez creado el archivo se procederá a hacer las configuraciones respectivas.

```
{/* Declaración del Top Tab Navigator que permite desplazar por las páginas
<Tab.Navigator  style={{paddingTop, backgroundColor:colors.primaryDark}}
                sceneContainerStyle={{backgroundColor:'white'}}

                screenOptions={({route}) => ({
                        tabBarPressColor: colors.primaryLight,
                        tabBarShowIcon: true,
                        tabBarIndicatorStyle: {
                            backgroundColor: colors.primaryLight,
                        },
                        tabBarActiveTintColor:'white',
                        tabBarStyle: {
                        borderTopColor: colors.primary,
                        borderTopWidth: 0,
                        elevation: 0,
                        backgroundColor:colors.primaryDark,
                        height:'10%',
                        },

                        style: {
                            backgroundColor:'#000',
                        },
```

Código 25 Configuración Top Tab Navifator

```
            tabBarIcon:({color,focused}) => {

                    let iconName:string='';
                    switch (route.name) {
                        case 'Inicio':
                            iconName='home-outline';
                            break;
                        case 'Mapa':
                            iconName='map-outline';
                            break;
                        case 'Incidencia':
                            iconName='speedometer-outline';
                            break;
                        case 'Alertas':
                            iconName='create-outline';
                            break;

                        default:
                            break;
                    }

                    return (  /* Se devuelve el icono */
                            <Text style={{color:'white'}}>
                            <Icon name={iconName} size={20} color='white' />
                            </Text>
                            );
                }
    })
    }
```

Código 26 Código 25 Configuración Top Tab Navifator asignación de Iconos

```
{/* Le indicamos los apartados que tendrá el menú de navegación del top tab */}
<Tab.Screen name="Inicio"  component={InicioScreen} />
<Tab.Screen name="Mapa"    component={MapaScreen} />
<Tab.Screen name="Alertas" component={AboutAlertasScreen} />
</Tab.Navigator>
```

Código 27 Código 25 Configuración Top Tab Navifator asignación de pantallas

En el Código 25 se asigna la configuración de los estilos que tendrá el encabezado, y como se puede observar en la Figura 35 en las secciones contiene un texto e iconos respectivos. Para lograr mostrar los iconos, se declarará una función al método *tobBarIcon* que dependiendo del nombre de la pantalla se asignará un icono. Por último, en el Código 27 se declara las pantallas que se mostraran en el encabezado que permitirá hacer la navegación.

Inicio

Una vez establecida las pantallas principales de la interacción del usuario, de *Login, Registro* y las navegaciones se procederá a mostrar la pantalla de inicio ya que en ambas pantallas mencionadas *Login, Registro* hacen la navegación hacia dicha pantalla.

Figura 37 Pantalla Inicio

Para la creación de esta pantalla se creará un archivo TSX el cual tendrá por nombre *InicioScreen.tsx* y como se ha mencionado con anterioridad se procede a realizar el mismo procedimiento para la creación del archivo.

En este desarrollo de la pantalla Inicio (Figura 37) en la identificación de los componentes, ya contendrá navegaciones y elementos adicionales el cual en la Figura 32 se puede observar y se muestra las distinciones a considerar.

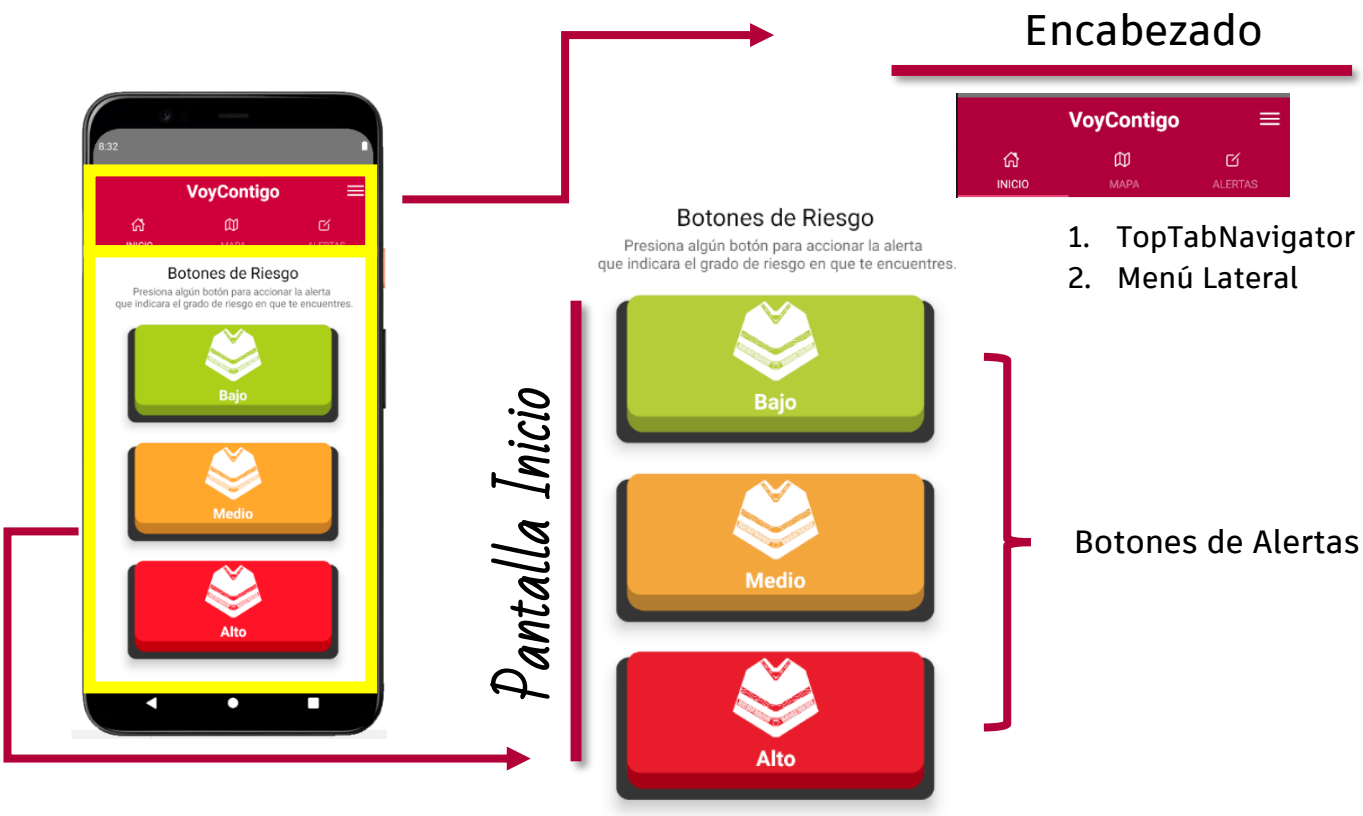


Figura 38 Identificación de la pantalla Inicio

En la Figura 38 se muestra la estructura de la pantalla de inicio, sin embargo, cabe mencionar que el archivo generado llamado *InicioScreen.tsx* corresponde a solo la parte inferior del encabezado, ya que el encabezado se mantiene como Componente Padre para tres pantallas, y donde contiene un menú lateral. Sin embargo, lo importante a resaltar es identificar es el desarrollo de la pantalla **Inicio** así que los componentes que se integran son: los botones que permitirán al usuario accionar las alertas que emitirán un mensaje de texto a los contactos que hayan configurado respectivamente, también se observan los componentes del título de la sección con una pequeña descripción que le indicara al usuario de que trata esta pantalla.

Mapa Virtual

Figura 39 Pantalla Mapa Virutal

Para el desarrollo de esta pantalla se seguirá la misma estrategia que la pantalla de Inicio, ya que pertenece a la navegación del *Top Tab Navigator*. Se creará un archivo TSX llamado *MapaScreen.tsx*, una vez creado el archivo en la carpeta *screens* se procera a realizar el desarrollo de la interfaz.

Considerando los componentes que integran la pantalla se observa la visualización de un mapa, sin embargo, antes de colocar dicho mapa se requieren hacer algunas configuraciones adicionales las cuales se hablaran en la sección **Implementación del Mapa Virtual** es por eso que solo se establecerá el espacio de trabajo para colocar el componente del mapa.

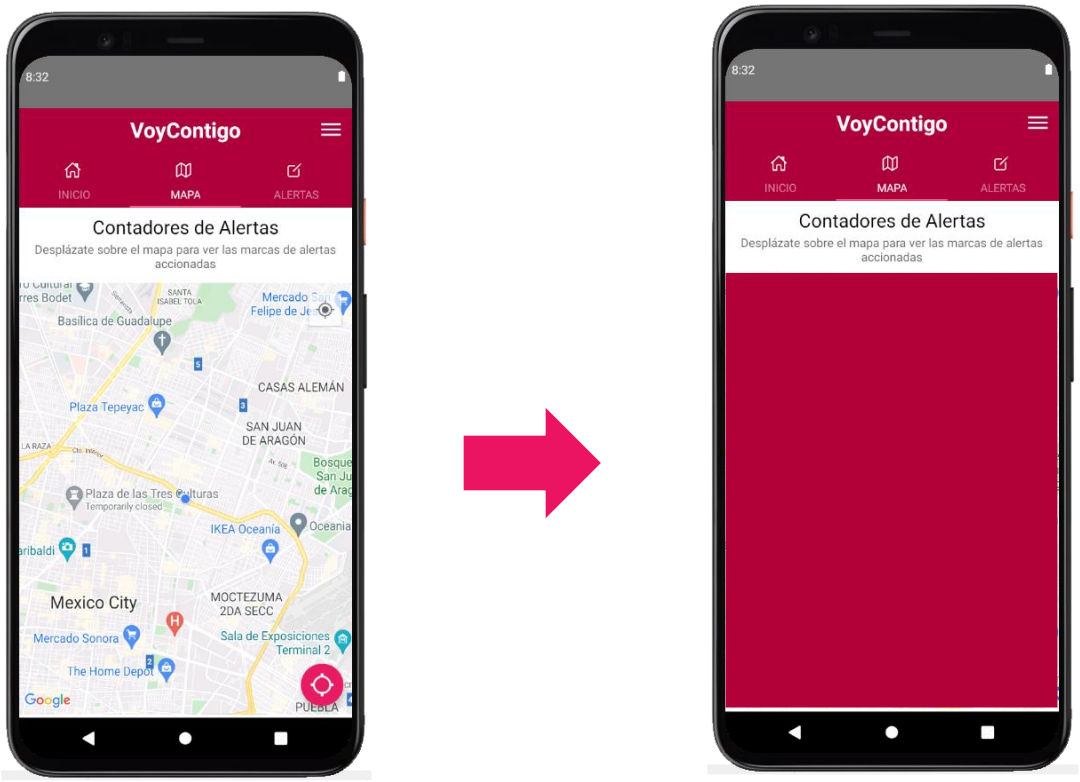


Figura 40 Desarrollo del área de trabajo para el mapa virtual

Acerca de las Alertas

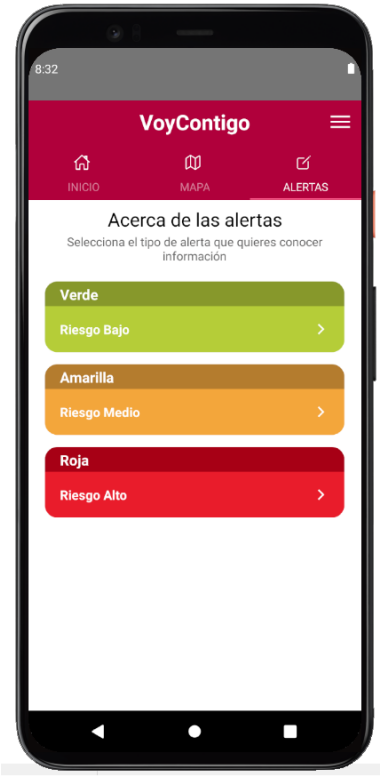


Figura 41 Pantalla Acerca de las Alertas

Para el desarrollo de esta pantalla se sigue con el mismo procedimiento como la pantalla de Inicio y Mapa Virtual, solo que en este caso se creara un archivo TSX llamado *AboutAlertasScreen.tsx.*

Una vez creado el archivo se procederá hacer el análisis de los componentes. En este análisis se observa que contiene tres botones que al ser accionados se navegara a otra sección respectiva a la información de alguna de las alarmas.

Información de la Alerta

Figura 42 Pantalla Información de Alerta

Para el desarrollo de esta pantalla se debe considerar que al ser otro apartado se necesita configurar previamente el encabezado que como se hizo en la sección del *Top Tab Navigator* para que de esta manera se coloque la pantalla Información en dicha sección como componente Hijo.

Alertas Top Tab

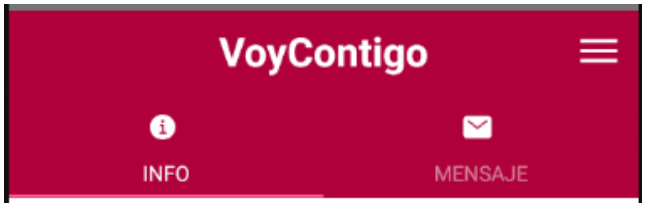


Figura 43 Sección Top Tab para información de las Alertas

Como se mencionó en la sección del *Top Tab Navigator* se procederá a realizar el mismo desarrollo solo que en este caso se creará con el diseño de la Figura 43.

Para el desarrollo de este, en la carpeta de *navigator* se creará una carpeta llamada "*aboutAlertas*" donde en dicha carpeta se creará a un archivo TSX llamado *AlertasTopTab.tsx* como se muestra en la Figura 44.

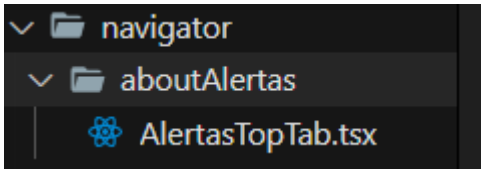


Figura 44 Creación de la navegación de información de alertas

Información de Mensaje

Figura 45 Pantalla Información del Mensaje

Para el desarrollo de esta pantalla se creará un archivo TSX llamado *MensajeConfigScreen*.tsx

Configuración de la Cuenta

Figura 46 Pantalla Configuración de la cuenta

Para el desarrollo de esta pantalla (Figura 46) se creará un archivo TSX llamado *CuentaScreen.tsx.*

Para esta pantalla contempla un delegue de contenido una vez que se accione algún botón que permitirá editar la información del Usuario.

Como se puede observar en la Figura 47 el botón emite un despliegue de información el cual mostrara un formulario que permitirá editar dicha información o cancelar la operación se pliegue nuevamente a su estado original.

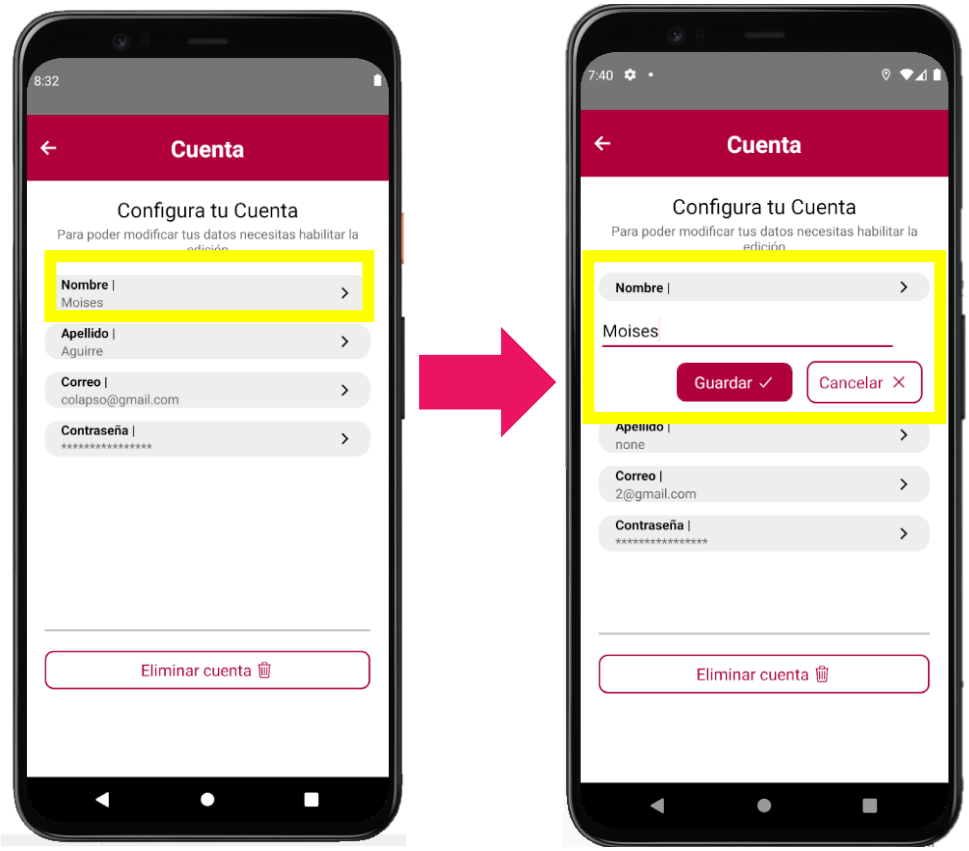


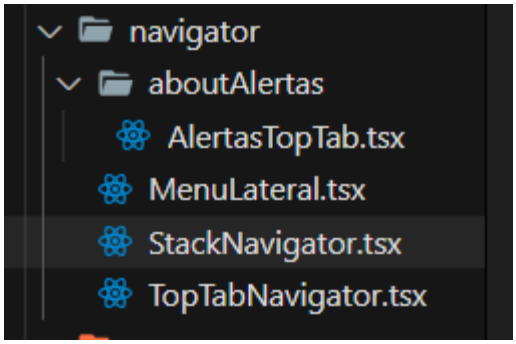
Figura 47 Pantalla Configuración de la cuenta despliegue de formulario

Configuración de la Navegación

En esta sección se implementará la navegación entre pantallas, dando la funcionalidad de activación de los botones para lograr la navegación. Cabe mencionar que para el desarrollo de algunas pantallas se implementaron con anterioridad dos tipos de navegación (*Top Tab Navigator y Drawer Navigator*) en las cueles fueron necesarias desarrollarlas para integrar el diseño, sin embargo, el tipo que se implementará en esta sección será Stack Navigator.

Implementación de Stack Navigator

Como primera instancia se procederá a crear un archivo TSX en la carpeta *navigator* el cual llevará por nombre *StackNavigator.tsx*



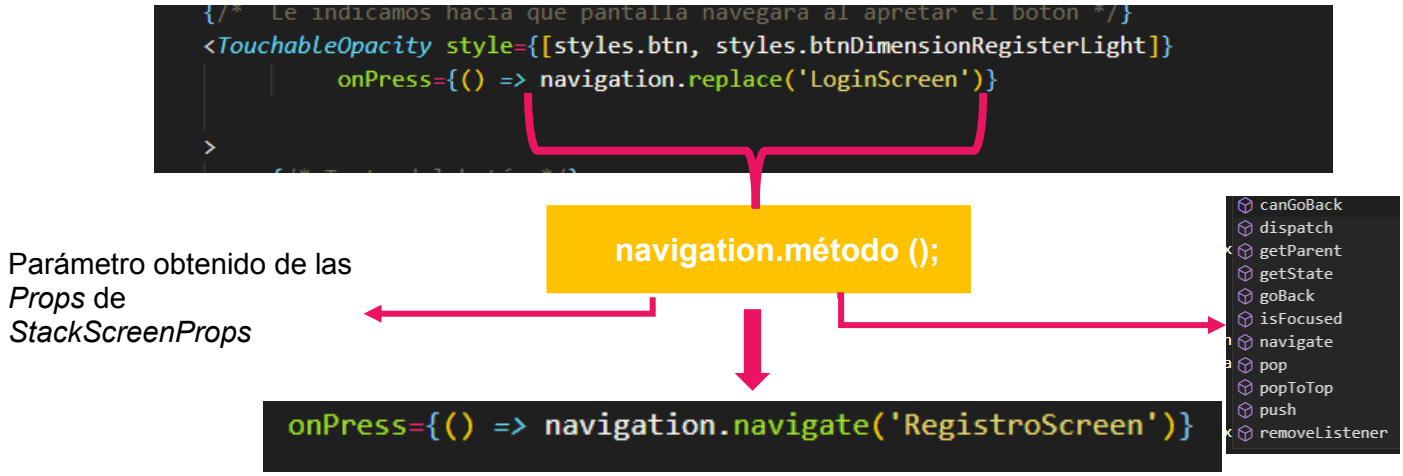
Para poder implementar la navegación se precederá a realizar la configuración como se muestra en el Código 28.

```
const Stack = createStackNavigator();

export const StackNavigator = () => {


  return (
    <Stack.Navigator  screenOptions={{
                                     headerShown:false,
                                     }}

      <Stack.Screen name="LoginScreen"    options={{title:'Login'}}    component={LoginScreen} />
      <Stack.Screen name="RegistroScreen" options={{title:'Registro'}} component={RegistroScreen} />
      <Stack.Screen name="RegistroScreen2"options={{title:'Registro'}} component={RegistroScreen2} />
      <Stack.Screen name="ContactoScreen" options={{title:'Registro'}} component={ContactoScreen}  initialParams={{primeraVez:true}}/>
      <Stack.Screen name="MenuLateral"    options={{title:'Inicio'}}   component={MenuLateral}  />

    </Stack.Navigator>
  )
}
```

Pantallas para Navegar

Código 28 Implementación del Stack Navigator

Una vez configurado el *Stack Navigator* a todos los botones que harán una navegación se les asignara la función como se muestra en el Código 29.



Código 29 Implementación de acción Onpress para navegación

Cabe mencionar que algunos botones que accionan formularios permiten la navegación a otra pantalla gracias a la previa configuración que se realizó en el archivo *StackNavigator.tsx*, el tipo de navegación de una validación exitosa del formulario se hará conforme al orden que fueron declaradas.

Configuración de Permisos del dispositivo

Para la configuración de permisos se contempla los requerimientos de la aplicación el cual se necesita las siguientes solicitudes para su funcionalidad:

- Acceso al GPS
- Lectura de Contactos
- Acceso a Internet
- Lectura y escritura de mensajes
- Envió de mensajes
- Lectura de números de teléfono

Se hará uso de un paquete que nos ayudara a implementar los permisos la cual es *API de React Native Permissions*. Esta API proporciona la ayuda para solicitarle a el usuario los permisos para el uso de la aplicación

Para configurar y habilitar estas funcionalidades se configurar el código para los permisos en el dispositivo Android.

Se procedera a ubicar el archivo **android -> src -> AndroidManifiest.xml** donde se agregara las siguientes líneas de código como se muestra en el Código 30.

```
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.READ_PROFILE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.READ_PHONE_NUMBERS" />
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.WRITE_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />
```

Código 30 Inserción de premisos para Android

Una vez instalado la API y colocado las líneas de código de los permisos que se necesitan, se implementara el llamado del uso de los permisos.

Llamado a los permisos del dispositivo

La manera de poder solicitar los permisos es muy similar, solo cambia en el tipo de permiso que se requiera, siguiendo la estructura siguiente.

```
request( PERMISSSIONS.ANDROID.----Permiso a solicitar------)
```

Código 31 Estructura de solicitud de permisos del dispositivo Android

Permisos activación GPS

Para esta implementación se tendrá una configuración distinta a las demás ya lo que se pretende es que se obtenga la ubicación en tiempo real del usuario, verificando si está activo el GPS y si los permisos no fueron denegados.

Como primera instancia se creará un *Context* que estará al pendiente del estado de la aplicación, por lo que se creará una carpeta llama *context* y un archivo llamado *PermissionsContext.tsx* como se muestra en la Figura 48.

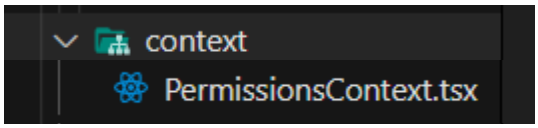


Figura 48 Creación del archivo PersmissionsContext

Una vez creado el archivo se creará una interfaz para indicarle un tipado del estado de los permisos como se muestra en el Código 32.

```
export interface PermissionsState {
    locationStatus: PermissionStatus;
}
```

Código 32 Interfaz PermissionsState

Posteriormente se creara un estado inicial y un contexto como se muestra en el Código 33.

```
export const permissionInitState: PermissionsState = {
            locationStatus: 'unavailable',
}

type PermisssionsContextProps = {
    permissions: PermissionsState;
    askLocationPermission: () => void;
    checkLocationPermission: () => void;
}

export const PermissionsContext = createContext({} as PermisssionsContextProps) ; //todo lo que exporta
```

Código 33 Creación del contexto PermissionContext

Se creará un componente funcional llamado *PermissionProvider* para así comenzar a desarrollar los siguientes códigos.

```
export const PermissionsProvider = ({children}:any) =>{
    const [permissions, setPermissions] = useState( permissionInitState);

    useEffect(() => {

       /* Lanzaremos la confirmación de los privilegios */
        checkLocationPermission();

        AppState.addEventListener('change',state =>{
            if(state !== 'active') return;
            checkLocationPermission();
        })

    }, [])
```

Código 34 Implementación del PermissionProvider parte 1

Como se muestra el Código 34 se declara un *useState* para mandar el estado en el que se encuentre el permiso de la aplicación.

A través del useEffect se mantendrá al pendiente de saber si el estado de la aplicación ha cambiado, la cual dentro del useEffect se declaran las funciones *chekLocaionPermission* (Código 35) y un *Listener.*

```
const checkLocationPermission = async () => {
    let permissionStatus: PermissionStatus;
    if(Platform.OS === 'ios'){
       permissionStatus = await check(PERMISSIONS.IOS.LOCATION_WHEN_IN_USE);
    }else {
       permissionStatus = await check(PERMISSIONS.ANDROID.ACCESS_FINE_LOCATION);
    }
    setPermissions({
        ...permissions,
        locationStatus: permissionStatus
    });
}
```

Código 35 Método chekLocationPermission

Como se observa el Código 35 se pide el estado de la ubicación del dispositivo, es decir si esta habilitado o no dicho permiso se manda a través del useState

```
const askLocationPermission = async () => {
    let permissionStatus: PermissionStatus;

    if(Platform.OS === 'ios'){
       permissionStatus = await request(PERMISSIONS.IOS.LOCATION_WHEN_IN_USE);
    }else {
       permissionStatus = await request(PERMISSIONS.ANDROID.ACCESS_FINE_LOCATION);
    }

     if(permissionStatus === 'blocked'){
         openSettings();
     }
    setPermissions({
        ...permissions,
        locationStatus: permissionStatus
    });
}
```

Código 36 Implementación askLocationPermission

En cambio, en el Código 36 preguntamos si se encuentra bloqueado los permisos para poder así verificar y mandar una ayuda hacia los ajustes del dispositivo para que pueda habilitar los permisos de manera manual.

```
return(
    <PermissionsContext.Provider value={{
        permissions,
        askLocationPermission,
        checkLocationPermission
    }}>
        {children}
    </PermissionsContext.Provider>
)
```

Código 37 Respuesta del PermissionContext

El Código 37 permitirá encapsular el contenido para que de esta manera las funciones permanezcan activas durante el uso de la aplicación.

Una vez configurado nuestro archivo, se configurará en nuestra entrada principal en el archivo App.tsx (Código 39) para envolver el componente de la aplicación colocándolo en el componente *AppState* que recibirá un componente hijo el cual será el *StackNavigator* (Codigo 39).

```
const AppState = ({children}:any) =>{
  return (
    <PermissionsProvider>
      <AuthProvider>
      {children}
      </AuthProvider>
    </PermissionsProvider>
  )
}
```

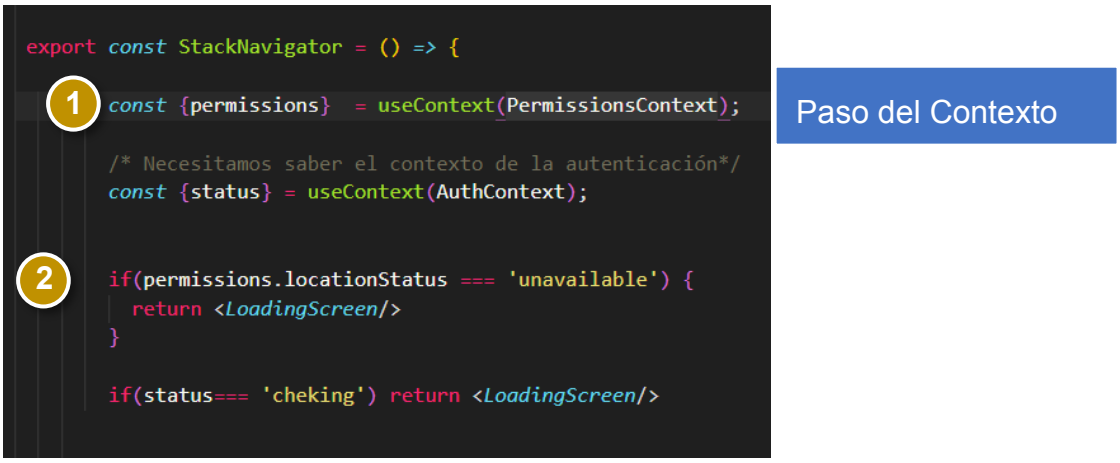
Código 39 Creación del AppState y declaración del PermissionsProvider

```
const App = () => {
  return (

    <NavigationContainer>
      <AppState>
      <StackNavigator />
      </AppState>
    </NavigationContainer>

  )
}
```

Código 38 Implementación del AppState envolviendo el StackNavigator

Adicionalmente se configurará el *Stack Navigator* con la finalidad de que determine si la aplicación cuenta con el permiso de acceder al GPS del dispositivo y en caso de que no sea así muestre una pantalla con una leyenda para que el usuario pueda identificar que es indispensable el uso del GPS para poder utilizar la aplicación.

.



Código 40 Configuración del Stack Navigator para validar el estado del GPS parte 1

El Código 40 nos muestra en el punto 1 la extracción del estado del contexto para saber si los permisos son cedidos o no. Para el punto 2 en el tiempo que la aplicación determine si los permisos están habilitados mostrara una pantalla que tendrá un componente *Loading,* donde para poder mostrar esta pantalla es necesario implementarla.

Pantalla *Loading*

Recordando el procedimiento para la creación de las pantallas se utilizará la misma estrategia para poder crear la pantalla *Loading,* es por eso que a este archivo TSX se llamara *LoadingScreen.tsx*.

Una vez creado el archivo se implementará la pantalla de la Figura 49.

Figura 49 Pantalla Loading

Ya desarrollada la pantalla *LoadingScreen,* se procederá a validar si los permisos del acceso al GPS están activados, así se implementará el Código 41.

```
return (
    <Stack.Navigator  screenOptions={{
                                headerShown:false,
                              }}
    >
      {
        (permissions.locationStatus === 'granted')?
            <>
              <Stack.Screen name="LoginScreen"     options={{title:'Login'}}     component={LoginScreen} />
              <Stack.Screen name="RegistroScreen" options={{title:'Registro'}} component={RegistroScreen} />
              <Stack.Screen name="RegistroScreen2"options={{title:'Registro'}} component={RegistroScreen2} />
              <Stack.Screen name="ContactoScreen" options={{title:'Registro'}} component={ContactoScreen}  initialParams={{primeraVez:true}}/>
              <Stack.Screen name="MenuLateral"     options={{title:'Inicio'}}    component={MenuLateral}  />
            </>
          :
          <Stack.Screen name="PermissionsScreen" component={PermissionsScreen} />
      }
    </Stack.Navigator>
  )
}
```

Verificación del permiso

Pantallas de acceso en caso de ser valido los permisos

Muestra la pantalla PermissionsScreen

Código 41 Configuración del Stack Navigator para validar el estado del GPS parte 2

Pantalla *Permissions*

Como se ha mencionado en el Código 41 se necesita crear una pantalla que permitirá al usuario activar los permisos de la ubicación del GPS

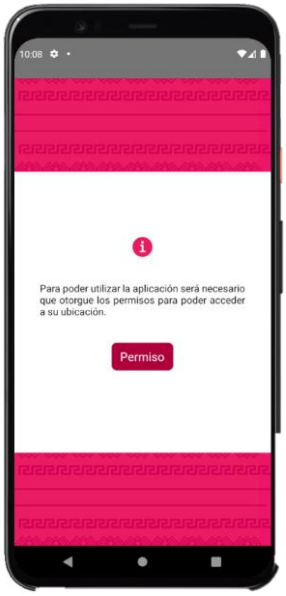


Figura 50 Pantalla Permissions

Para el desarrollo de esta pantalla se creará un archivo TSX llamado *PermissionsScreen.tsx* en el cual se desarrollará la parte importante del componente el cual será la activación del botón.

```
export const PermissionsScreen = () => {

   const {askLocationPermission} = useContext(PermissionsContext);
```

Código 42 Uso del context para acceder a los permisos de la ubicación

En el Código 42 se hará uso del *context* que se había creado con anterioridad para poder acceder al método PermissonContext, en el cual se extraerá el *askLocationPermission* para poder mostrar si el usuario desea activar los permisos.

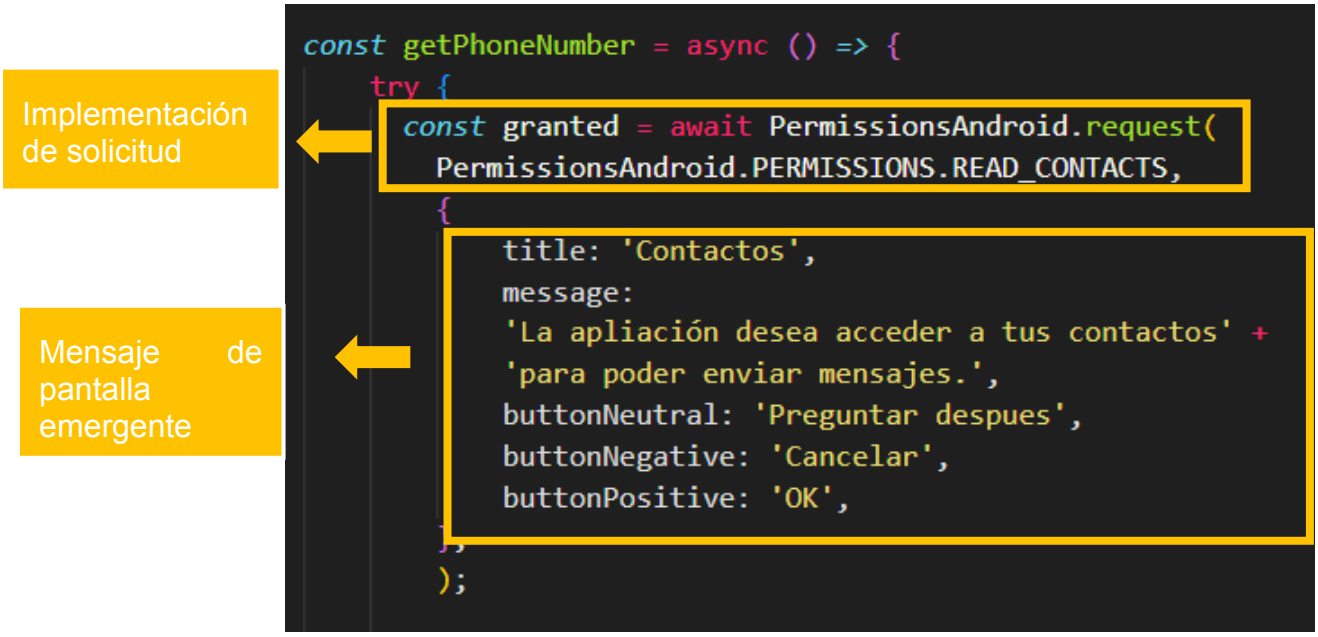
```
<View style={styles.containerBtn}>
  <TouchableOpacity style={styles.btn} onPress={askLocationPermission} >
                <Text style={styles.textBtn}>Permiso</Text>
  </TouchableOpacity>
</View>
```

Código 43 Activación de los permisos del GPS a través del onPress

En el código 43 se hará uso del *askLocationPermission* a través del *onPress* para accionar el método.

Permisos lectura de contactos

Como se vio en el Código 31 se implementará la misma estrategia para poder solicitar los permisos, solo que en este caso será para la lectura de los contactos.



Código 44 Implementación lectura de contactos

Como se ve en el Código 44 se tiene una parte adicional el cual se establece un mensaje de la pantalla emergente.

Permisos envió de mensajes de texto

Para él envió de mensajes para dispositivos Android seguirá siendo la misma estructura de solicitud de permisos, sin embargo, el tipo que se establecerá será para el envío de mensajes de texto.

```
const granted = await PermissionsAndroid.request(
PermissionsAndroid.PERMISSIONS.SEND_SMS,
{
    title: 'Envió de Mensajes',
    message:
    'Para poder enviar el mensaje es necesario que otorgue los permisos ' +
    '¿Desea que VoyContigo pueda enviar mensajes?.',
    buttonNeutral: 'Pregúntame Despues',
    buttonNegative: 'Cancelar',
    buttonPositive: 'OK',
},
```

Código 45 Implementación envío de mensajes de texto

Como se muestra el Código 45 es muy similar al Código 44 sin embargo cambia el tipo de permiso y las leyendas de la pantalla emergente.

Implementación del Módulo de visualización del mapa virtual

Para la implementación del mapa virtual se usará la API de **react-native-maps** y la API que ofrece Google el cual es de uso gratuito para versiones móviles, teniendo esto en cuenta se necesita registrase para poder configurar la API, enlazándola con la aplicación para poder hacer uso del mapa virtual

Se crara un proyecto en Google Cloud Platform (Figura 51) con el nombre ***VoyContigo***, donde al tener el proyecto se seleccionará y habilitará la API de **Maps SDK for Android**.

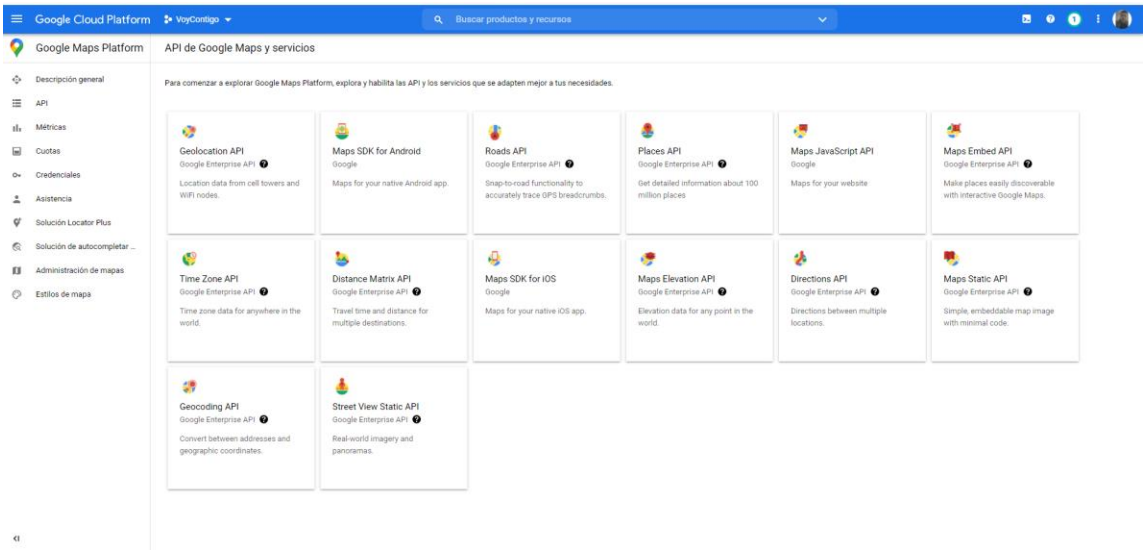


Figura 51Panel Google Platform

Una vez configurada la API, se necesitará configurar la API *Key* que permitirá configurar el mapa con la aplicación una vez ubicándose en el panel se muestra en la Figura 52.

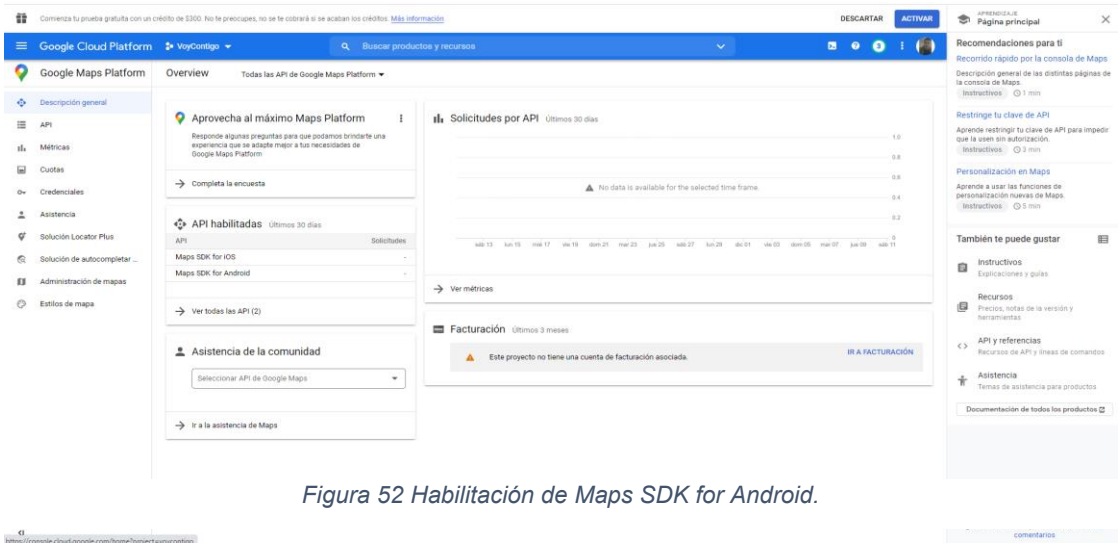


Figura 52 Habilitación de Maps SDK for Android.

Se redirigirá la sección de las “credenciales” para poder crear la Clave de la API y así también poder se restringir la llave, colocándole el nombre de ***VoyConrigo Maps*** asignando la configuración para Maps SKD for Android como se muestra en la Figura 53.

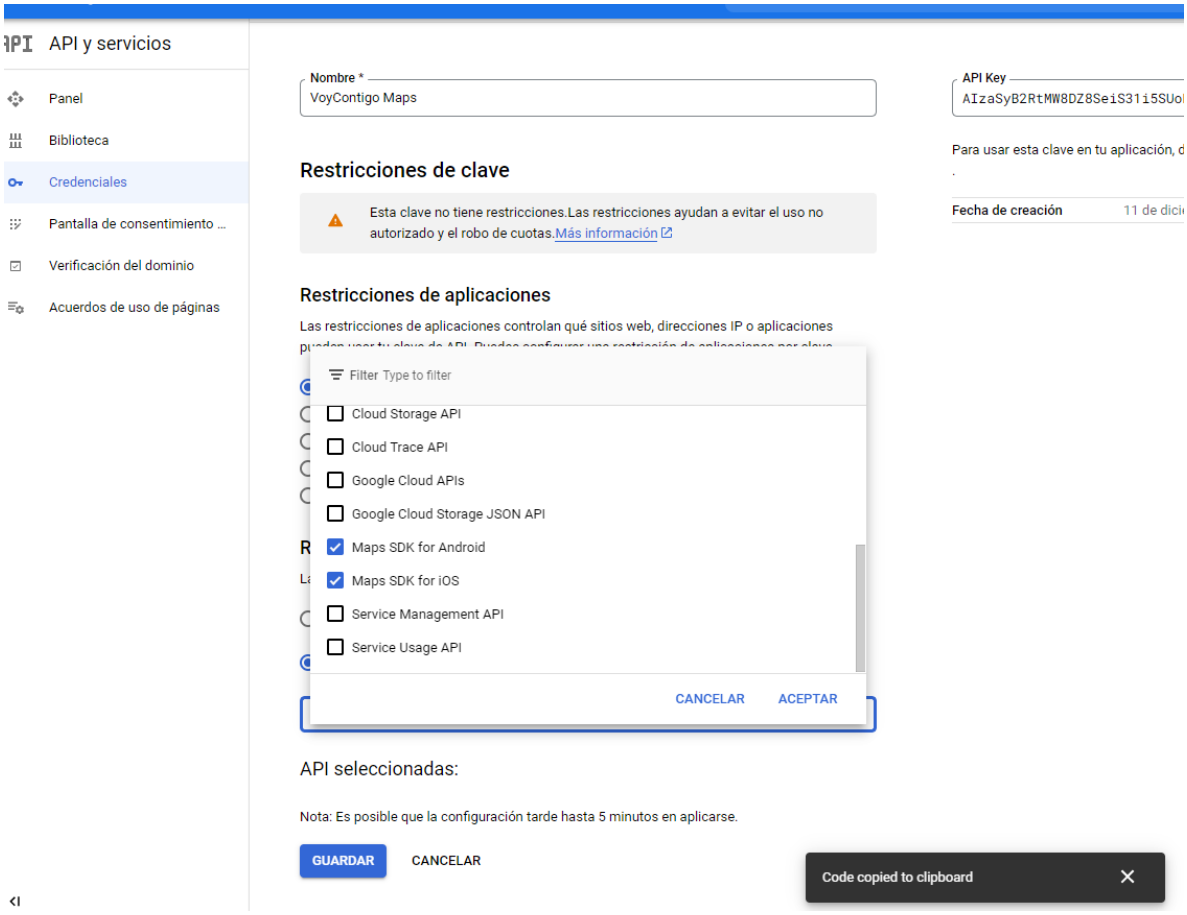
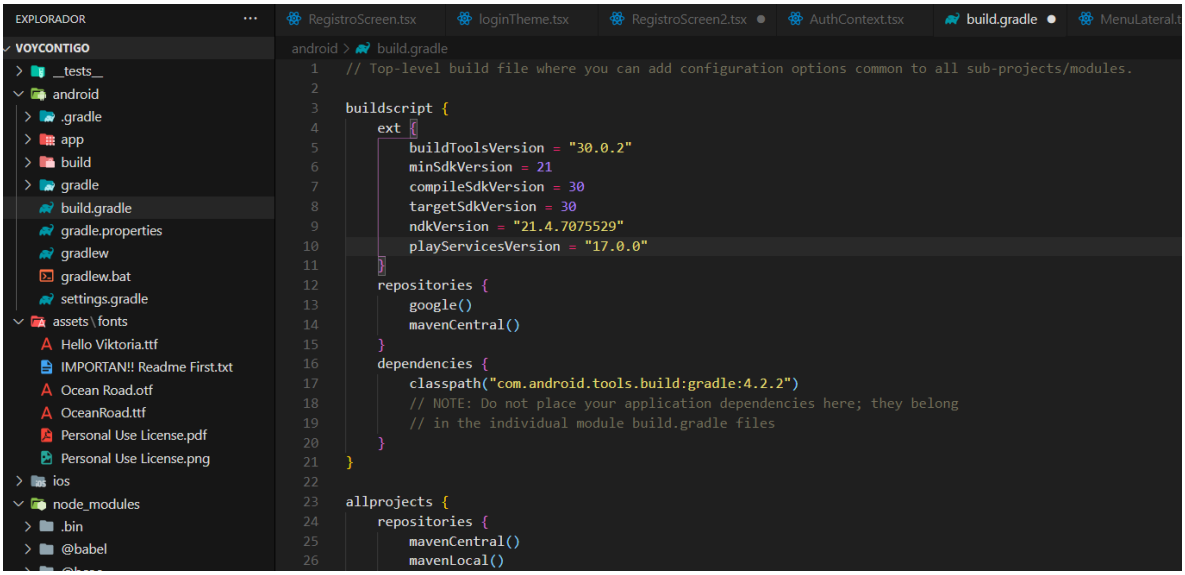


Figura 53 Selección de Maps SDK for Android

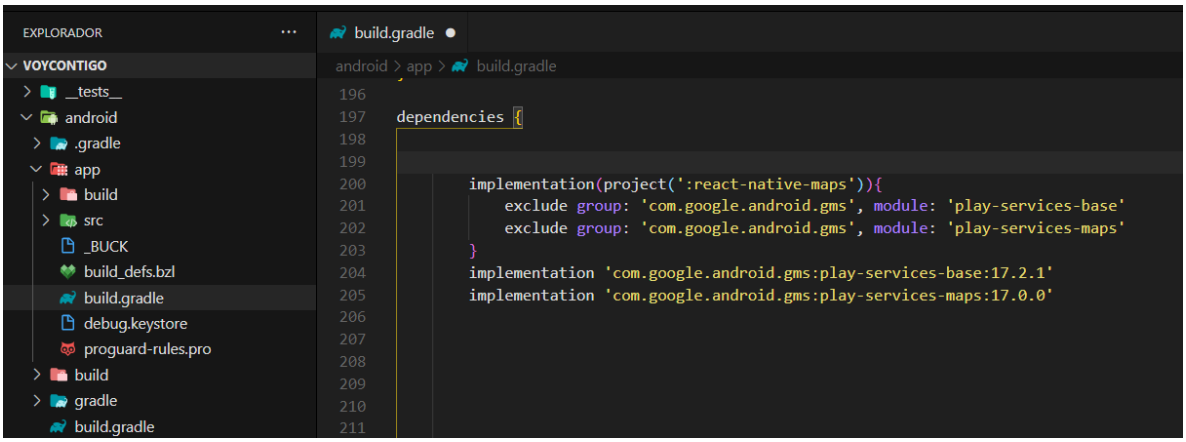
Una vez seleccionadas, se salvarán los cambios y se procederá a copiar la llave que se generó para poder llevarla a el proyecto.

En el proyecto de React-Native, se procederá a ubicarse en el archivo ***build.gradle*** donde en el ***buildscript*** se asignara la configuración requería como se muestra en la Código 46.



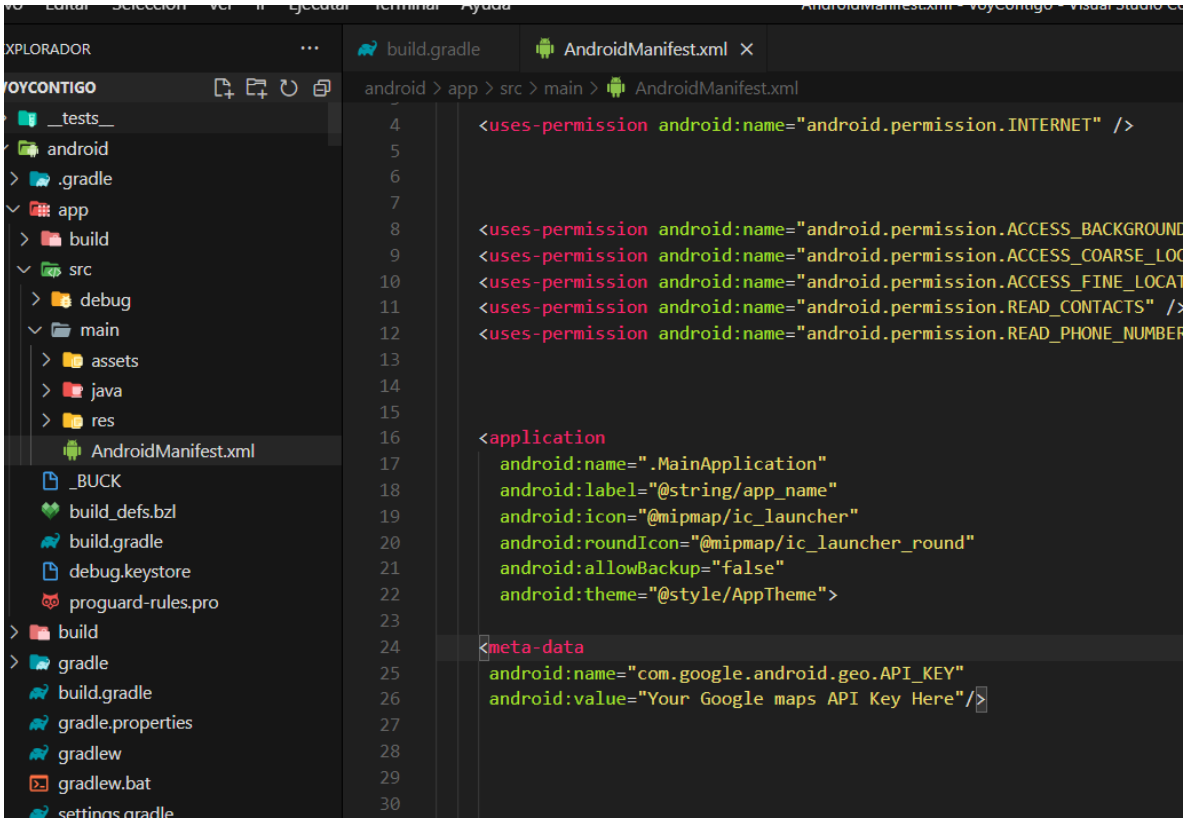
Código 46 Configuración del buildscript

En el ***build.gradle*** de la carpeta App se asignaran unas dependencias de Google para hacer la configuración del mapa virtual como se muestra en el Código 47.



Código 47 Asignación de las dependencias para el mapa virtual

Como ultima configuración en el archivo ***Android/app/src/main/AndroidManifest.xml*** en el ***meta-data*** se asignará la llave que se generó.



Código 48 Configuración del meta-data para la llave de la API de Google

Quedando asignada como se muestra en el Código 49 la configuración.

```
<meta-data
 android:name="com.google.android.geo.API_KEY"
 android:value="AIzaSyB2RtMW8DZ8SeiS31i5SUoDiIOnzt6S0S8"/>
<uses-library android:name="org.apache.http.legacy" android:required="false"/>
```

Código 49 Asignación de la llave generada

Una vez realizado las configuraciones para utilizar el mapa, se comenzará a implementar la asignación de este en la pantalla del mapa virtual que se había creado.

Asignación del Mapa Virtual en Pantalla

Para la asignación del mapa virtual en la pantalla se necesitarán realizar algunas configuraciones antes, las cuales permitirán obtener la ubicación del usuario en tiempo real conforme se valla moviendo de posición, pueda tener las coordenadas de dicho usuario, de este modo se basará en las coordenadas para dibujar su traslado en el mapa.

Creación del Hook useLocation

Se creará una carpeta llamada *Hooks* en dicha carpeta se creará un archivo TSX que se llamará *useLocation*, como se muestra en la Figura 54.

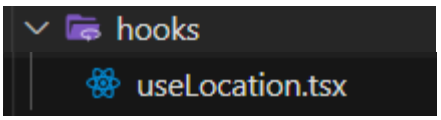


Figura 54 Creación del archivo useLocation.tsx

Posteriormente se crearán algunos *useState* para manejar los datos de mejor manera.

Ruta

Posición inicial

Posición del usuario

Verificará si la ubicación esta activa

```tsx
export const useLocation = () => {

    const [hasLocation,setHaslocation]=useState(false);
    const [routeLines,setRouteLines]=useState<Location[]>([]);
    const [initialPosition,setinitialPosition]=useState<Location>({
        longitude:0,
        latitude:0
    });
    const [locationDialog, setLocationDialog] = useState(true);
    const [userLocation,setUserLocation] = useState<Location>({
        longitude:0,
        latitude:0
     });
    const watchId = useRef<number>();
    const isMounted = useRef(true);
```

Código 50 Creación de useStates para obtención de la ubicación

En el Código 50 se muestra la declaración de los *useState* con los que se trabajaran.

Posteriormente de implementar un *useEffect* para que cargue todas las funciones que se implementaran y se logre cargar el estado de estas cuando se acceda al componente como se muestra en el Código 51.

```
useEffect(()=> {
        /* Se hace un llamado al getCurrentLocation  */
        getCurrentLocation().then( location => {
        /* Se verifica si está montado el componente  */
        if( !isMounted.current ) return;
        /* Se asignan las coordenadas principales,
         con la promesa que devuelve la posición actual  */
        setinitialPosition(location); /* posición inicial del mapa  */
        setUserLocation(location); /* posición inicial del usuario  */
        /* cuando se refresque la localización lo guardaremos en el
        arreglo, por lo que se des estructure   */
        setRouteLines( routes => [ ...routes, location ]);
        setHaslocation(true);
        });


},[]) ;
```

Código 51 Implementación del useEffect para el useLocation

En base el Código 51 se definirán las funciones que se lanzarán.

```
const getCurrentLocation = (): Promise<Location> =>{
        return new Promise ((resolve,reject) => {
            /* Se hace una petición para obtener las coordenadas actuales */
            Geolocation.getCurrentPosition(
                ({coords}) =>{
                    resolve({
                        latitude:coords.latitude,
                        longitude:coords.longitude
                    });
                },
                (err) => reject({err}), { enableHighAccuracy:true, showLocationD
            );
        });

}
```

Código 52 Implementación getCurrentLocation

En el Código 52 se creará una función del estilo de una promesa ya que obtendrá la latitud y la longitud de la ubicación, devolviendo así un tipado del tipo ***Location*** (longitud y latitud).

A continuación, en el Código 53 se implementará la función *followUserLocation* la cual permita seguir al usuario con la cámara del mapa virtual.


```
const followUserLocation = () =>{
    /* se quiere cancelar el watchposition, si se cierra la aplicación
      se mantendrá activo El GPS */
    watchId.current= Geolocation.watchPosition(
        ({coords}) => {
            /* Se verifica si está montado el componente  */
            if( !isMounted.current ) return;
            /* Se crean las variables para la latitud y longitud  */
            const location: Location = {
                latitude:coords.latitude,
                longitude:coords.longitude,
            }
            /* Cada vez que el usuario se mueve lo veremos con el useState   */
            setUserLocation(location);
            /* el estado actual se asigna a un arreglo de posiciones desestructu
            setRouteLines( routes => [ ...routes, location ]);
            /* Mensaje en cosnola de las coordenadas obtenida */
            console.log(location);
        },
        /* Cada vez que pasen 10 metros el usuario notificara con su ubicaión,
        pero con la cuestión de que cada vez que se utilice consumirá la baterí
        (err) => console.log(err), {enableHighAccuracy:true, distanceFilter:10,
    )
}
```

Código 53 Implementación de la función followUserLocation

Así mismo se desarrollará la función *stopFollowLocation* (Código 54) que permitirá detener el estado *watch* de la ubicación, es decir que ya no obtendrá la ubicación del usuario.

```
const stopFollowLocation = () => {
    if(watchId.current)
     Geolocation.clearWatch(watchId.current);
}
```

Código 54 Implementación de la función stopFollowLocation

Una vez implementadas las funciones se devolverán para que sean consumidas en otras secciones, porque se implementara el siguiente Código 55.

```
return {

        hasLocation,
        initialPosition,
        getCurrentLocation,
        followUserLocation,
        userLocation,
        stopFollowLocation,
        routeLines
}
```

Código 55 Respuesta del componente useLocation

Ya establecido el *useLocation* se procederá a implementar el componente del mapa.

Creación del componente del Mapa Virtual

Para implementar el componente del mapa virtual se creará una carpeta llamada *components* en dicha carpeta se creara una subcarpeta llamada *Maps* dentro de esa carpeta se creará un archivo TSX llamado *Map.tsx* como se muestra en la Figura 55.

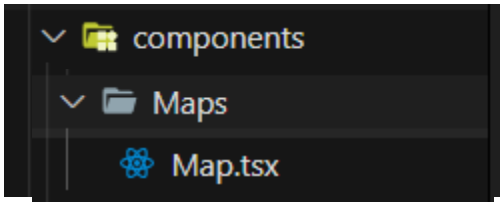


Figura 55 Creación del componente Map.tsx

Una vez creado el archivo se procederá a implementar el Código 56.

```
const{ hasLocation, initialPosition,getCurrentLocation,followUserLocation , userLocation,stopFollowLocation,routeLines} = useLocation();
/* Se hace una  referencia para el MapView*/
const mapViewRef = useRef<MapView>();
/* Se crea otra referencia para ver si el usuario mueve el mapa  */
const following= useRef<boolean>(true);
/* Si queremos monitorear al usuario siempre*/
useEffect  (() => {
      followUserLocation();
      return ()=>{
          /* hacemos la cancelación del seguimiento, para evitar el desbordamiento de la memoria */
          stopFollowLocation();
      }
},[])
```

Código 57 implementación del Mapa Virtual -Parte 1

En el Código 56 se hace el llamado a las funciones que se implementaron en el Hook de *useLocation* una vez extraídas se procederá a inicializarlas en el *useEffect*.

```
/* cada vez que el uselocation cambie se va disparar   */
useEffect  (() => {
        /*Se crea  una condición para evitar que el mapa nos mueva la cámara en caso de que el usuario desee explorar fura del seguimiento
          en caso de que el following sea falso la cámara se ira moviendo con el rastreo o */
      if(!following.current) return;
       /* Se desestructura las variables de Userlocation  para obtener las coordenadas del usuario */
       const {latitude, longitude} = userLocation;
       mapViewRef.current?.animateCamera({
           center:{latitude,longitude}
      });

},[userLocation])
```

Código 56 Implementación del Mapa Virtual - Parte 2

En el Código 57 se creará otro *useEffect* para poder realizar el seguimiento del usuario con la cámara del mapa virtual.

Una vez creado el *useEffect* se procederá a crear un método llamado *centerPosition* que permita centrar la posición de la cámara virtual en caso de que el usuario se salga del área de donde se encuentre al accionar este método regresara al punto de su ubicación.

```
const centerPosition = async () => {
    /*Se extraen los valores  */
    const {latitude,longitude} =await getCurrentLocation();
    /* Se coloca el seguimiento de la cámara nuevamente */
    following.current = true;
    /* Se anima la cámara con el centro de la latitud y longitud extraída */
    mapViewRef.current?.animateCamera({
        center:{latitude,longitude}
    });
}
```

Código 58 Método de centrado de la cámara virtual

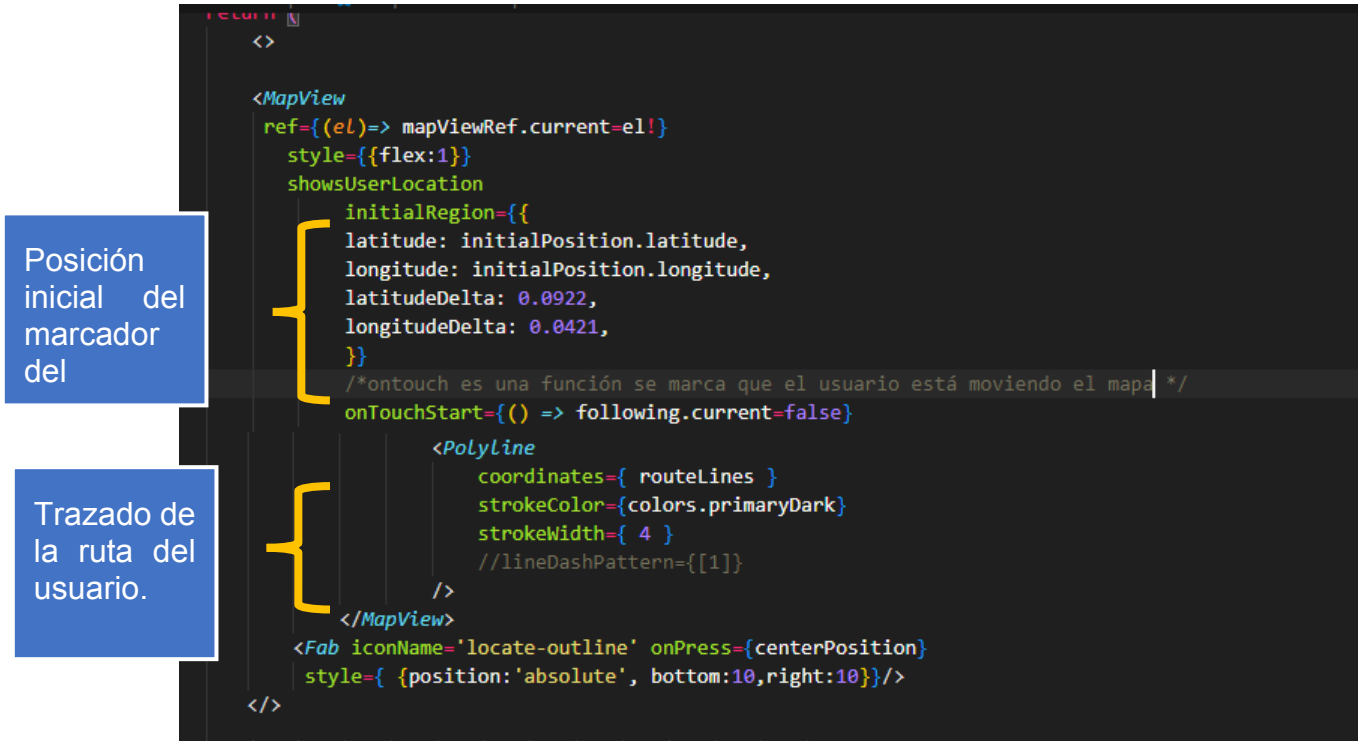
El Código 58 hace el llamado a obtener la ubicación en el instante que este para que pueda animar la cámara dirigiéndose al punto de donde se encuentra.

```
if(!hasLocation){
    return <LoadingScreen/>
}
```

Código 59 Verificación de la obtención de la ubicación

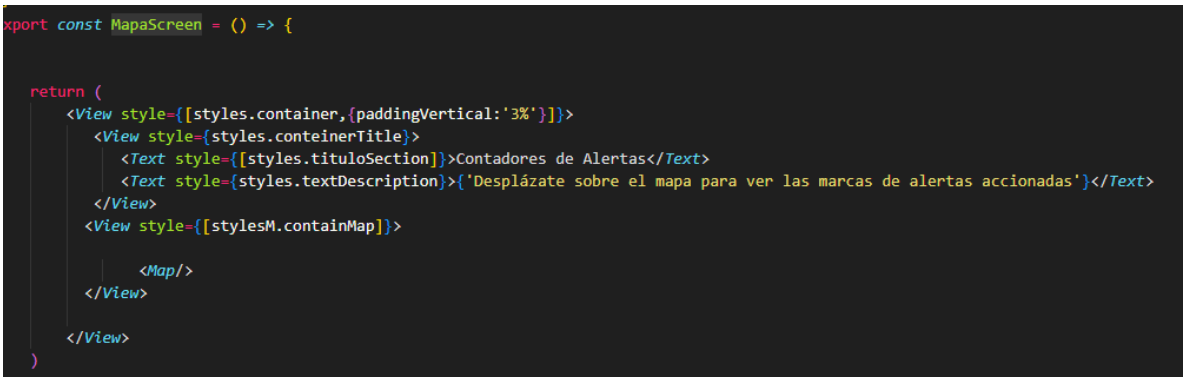
Así mismo mientras se obtenga la ubicación del usuario por primera vez se mostrará la pantalla de *Loading,* por lo que en código 59 se hace una validación para mostrar la pantalla o no.

Por último, se implementará el componente MapView como se muestra en el Código 60.



Código 60 Implementación del componente MapVIew

Ya obtenido el componente en el archivo *MapaScreen.tsx* se instanciará el llamado del MapView como se muestra en el Código 60.



Código 61 Instancia del MapView en la pantalla MapScreen

Implementación Módulo de creación y configuración de alertas

Una vez obtenida las bases necesarias para poder darle la funcionalidad a la aplicación se comenzará a realizar seccionada mente puntos estratégicos para lograr las características correspondientes al módulo creación y configuración de alertas, ya que es una pieza fundamental para la realización de los próximos módulos.

Implementación de Inicio de sesión

En esta sección se desarrollará la estrategia de implementación del *Login* de un usuario, en donde se harán validaciones y peticiones desde el Front-End al Back-End.

Como primera instancia al tener ya creado el Back-End se seguirá con la misma estrategia para poder mandar peticiones, creando una ruta específica para poder autenticar usuarios. Es así como en primera instancia se configurará el Back-End donde en las rutas se creará un archivo llamado ***auth.ts*** donde se definirá las siguientes rutas del Código 62.

```
router.post('/login',[
    check('correo','el correo es obligatorio').isEmail(),
    check('password','La contraseña es obligatoria').not().isEmpty(),
    validarCampos
],login);


router.get('/',[
    validarJWT
   ],validarTokenUsuario );
```

Código 62 Implementación de la ruta Login

En el método POST se definirá la ruta "/*login*" en donde se recibirá los parámetros de el correo y la contraseña. Del mismo se colocarán unos middlewares para hacer las validaciones correspondientes antes de ejecutar el método de *Login.*

Una vez definido el método *login* se tiene que crear en los controladores la definición de estos métodos para ser utilizados en la ruta de *Login*, a este archivo se le asignara el nombre ***auth.ts*** creándolo en la carpeta *controller.*

Una vez creado el archivo *auth.ts* se procederá a implementar el método *login* como se muestra en el Código

```
export const login=async (req: Request,res:Response) => {

    const {correo, password} = req.body;
    try {
        /* Verificación si el email existe */
        const usuario= await Usuario.findOne({where:{correo}});
        if(!usuario){
            return res.status(400).json({
                msg:'Usuario/ Password no son correctos - correo'
            })
        }
        /* verifica la contraseña */
          const validarPassword = bcrypjs.compareSync(password,usuario.password);
          if ( !validarPassword ) {
            return res.status(400).json({
                msg: 'Usuario / Password no son correctos'
            });
        }
        /* Genera el jwt */
        const token = await generarJWT( usuario.idUsuario );

        res.json({
           usuario,
           token
        })

    } catch (error) {
        console.log(error)
        res.json({
            msg:'Hable con soporte técnico.'
        })
    }

}
```

Código 63 Implementación método login

En el Código 63 se reciben los parámetros del *request* donde se desestructura los parámetros que se reciben para así poder validar si existe algún registro con el mismo correo y así poder mandar un mensaje en caso de existir.

Posteriormente se utilizará ***bcrypjs*** con el método que nos ofrece de poder comprar dos *passwords* decodificando la contraseña encriptada y validando ambos parámetros, en caso de que la respuesta sea falsa se mostrara un mensaje de que la contraseña fue incorrecta.

Como última instancia se generará un *Json Web Token* con el id del usuario pasando todo el objeto del usuario como respuesta,

Dada la situación de que las validaciones implementadas sean erróneas se mandará un mensaje de error y no permitirá el acceso.

Para la siguiente ruta del método *GET* es la validación del JWT, pero sin embargo para hablar del JWT se tiene que crear un token, de este modo se podrá mantener el estado de los datos del usuario sin realizar más peticiones y que nuestro sistema pudiese ser vulnerable a ser interceptado para saber los parámetros de manera trasparente, es por eso que la utilización de un *Json Web Token* permitirá codificar la información siendo así no tan visible la información que se obtenga.

JWT (JSON Web Token)

Cuando hay autenticaciones de inicios de sesión se creaban variables de sesión de usuarios donde el servidor crea la variable permitiendo que el usuario pueda realizar peticiones hacia el servidor, sin embargo estas variables de sesión han tenido problemas ya que cuesta mucho el poder gestionar estas, es por eso que se hará uso de JWT, que como se ha hablado brinda unca condición de seguridad para evitar que alguien intercepte las peticiones, aun que esta pueda ser vulnerada ya que la encriptación es de doble vía, sin embargo aun así nos permite una mejor gestión de la información de inicio de sesiones teniendo un tiempo de validación del JWT.

Ahora bien, se creará un JWT que nos permitirá tener la información del usuario que necesita mantener en la sesión donde se hará uso del paquete **jsonwebtoken**

Donde recordando que se tiene una función en el controlador del **auth** donde se define en el Código 63 se centrara en la función ***generaJWT*** como se muestra en el Código 64.

```
/* Genera el jwt */
const token = await generarJWT( usuario.idUsuario );
```

Código 64 Generación del JWT

Para esta función se tiene que crear, es por eso que en la carpeta de ***helpers*** se creara un archivo que se llamado **generarjwt.ts** donde se implementara el siguiente Código 65.

```
import jwt from 'jsonwebtoken';


export const generarJWT = ( uid = '' ) => {

    return new Promise( (resolve, reject) => {

        const payload = { uid };
        /* Firmamos un nuevo jwt */
        /*  */
        jwt.sign( payload, ""+process.env.SECRETORPRIVATEKEY, {
            /* Tiempo de expiracion */
            expiresIn: '30 days'
        }, ( err, token ) => {

            if ( err ) {
                console.log(err);
                reject( 'No se pudo generar el token' )
            } else {
                resolve( token );
            }
        })

    })
}
```

Código 65 Definición del método del JWT

Con el método de ***jwt.sing*** se firma un nuevo token donde se colocara el *payload* seguido de una llave privada o secreta ,que en caso de que alguien la conozca alguien más podrá firmar los token es por eso que en las variables de entorno se declarara una variable llamada **SECRETORPRIVATEKEY** donde se asignara un valor. Posteriormente se asignará el tiempo de expiración del JWT. Por último, se disparará el *call back* de la respuesta en caso de que salga todo correcto se mandara el token a través del resolve.

Una vez implementado el *auth* se procederá a comprobar que el servicio de autenticación sea correcto, el cual se hará uso de **Postman.**

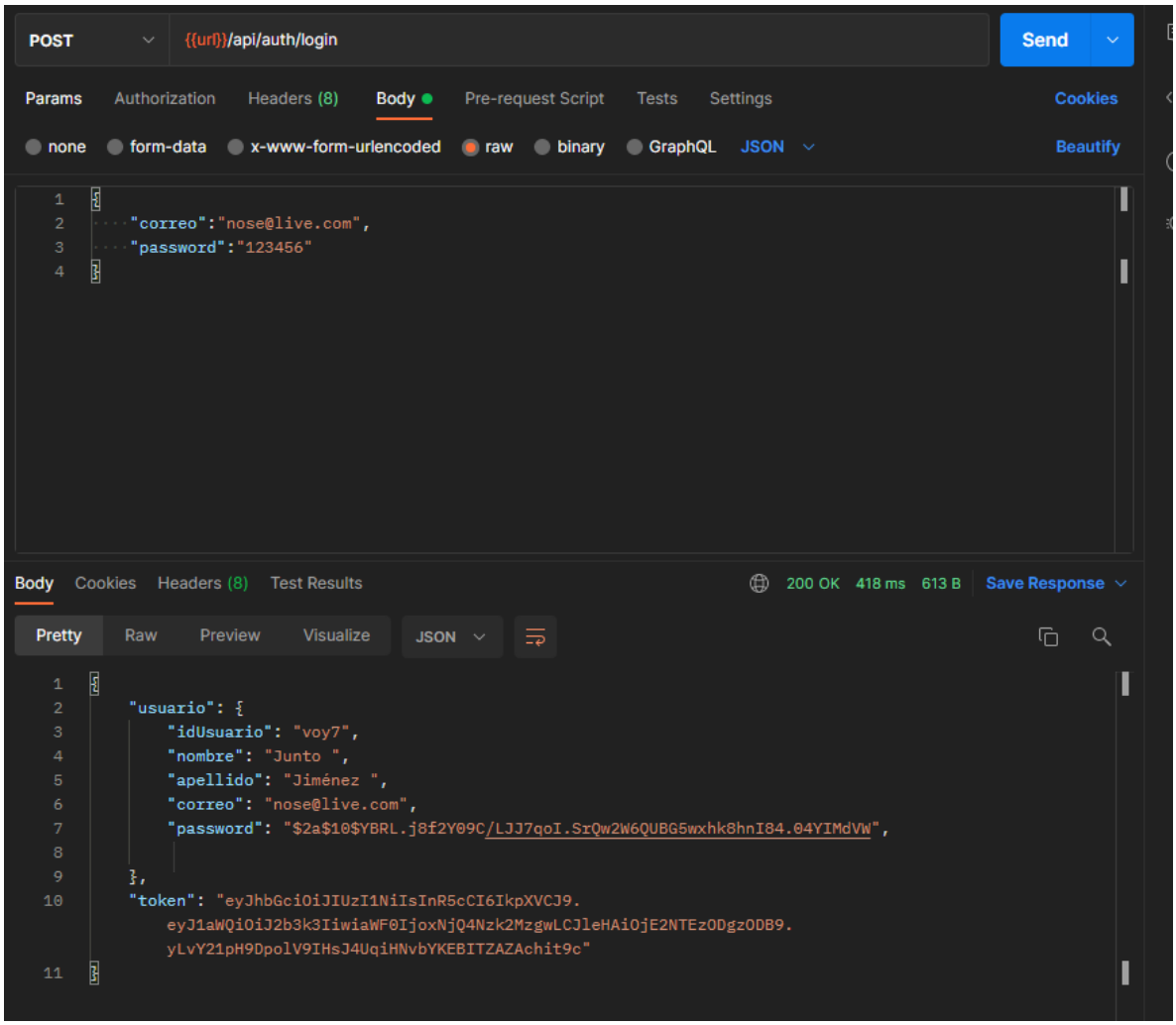


Figura 56 Comprobación del método Login a través de PostMan

Como se muestra en la Figura 56 se puede observar cómo al mandar el correo y la contraseña se obtiene como resultado el objeto de usuario y el token comprobando así la autenticación y generación del JWT.

Se implementará un middleware que verifique que sea válido el JWT el cual se creará un método llamado *validarJWT* en el archivo *validarJWT.ts* (Figura 9).

```
export const validarJWT = async( req: Request , res: Response, next:any ) => {
    const token = req.header('voy-token');
    if ( !token ) {
        return res.status(401).json({
            msg: 'No hay token en la petición'
        });
    }

    try {
        const payload  = (jwt.verify(token, process.env.SECRETORPRIVATEKEY+"") as payloadUser);
        const usuario = await Usuario.findByPk(payload.uid);
        if( !usuario ) {
            return res.status(401).json({
                msg: 'Token no válido - usuario no existe DB'
            })
        }
        (req as any).usuario=usuario;
        next();
    } catch (error) {
        console.log(error);
        res.status(401).json({
            msg: 'Token no válido'
        })
    }

}
```

Código 66 Implementación del método validarJWT

Como se muestra en el Código 66 se lee el token de la petición que validara si existe o no, en caso de que exista se verificara la información del token buscando en la base de datos si hay algún registro con dicho usuario.

Una vez genero del *middleware* se procederá en la ruta del *auth* del método *GET* para la función de *validarTokenUsuario* como se muestra en el Código 67.

```
router.get('/',[
    validarJWT
],validarTokenUsuario );
```

Código 67 Implementación del método GET para la ruta auth

Se procederá a implementar el Código 68 *validarTokenUsuario.*

```
export const validarTokenUsuario = async (req: Request,res:Response ) => {

    // Generar el JWT
    const token = await generarJWT( (req as any).usuario.idUsuario );

    res.json({
        usuario:  (req as any).usuario,
        token,
    })

}
```

Código 68 Implementación método validarTokenUsuario

Para el Código 68 se generará un nuevo JWT con la información recibida del usuario para generar un nuevo token de sesión.

Configuración de inicio de sesión en React-Native

Para poder hacer la confirmación del inicio de sesión es necesario consumir la API REST que se creó en el Back-End, es por eso que se describirá el proceso de consumir la API generada.

Consumo del Back-End

En el proyecto de React-Native se generará una carpeta llamada **API** dentro de la carpeta se creará un archivo TSX llamado **voyAPI.tsx** como se muestra en la Figura 57.

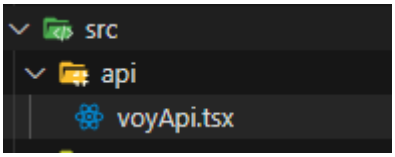


Figura 57 Creación del archivo voy Api.tsx

Para poder consumir el Back-End se hará uso de **axios** que traerá la dirección que corresponde a el servidor, en este caso es una dirección local como se muestra en el Código 69.

```
import axios from 'axios';
import AsyncStorage from '@react-native-async-storage/async-storage';

/* Se declara la ruta para hacer las peticiones */
const baseURL= 'http://10.0.0.4:8000/api';

const voyApi = axios.create({baseURL});

/* Se crea un middleware axios para obtener el header*/
voyApi.interceptors.request.use(
    async(config:any) => {
        const token = await AsyncStorage.getItem('token');
        if(token){
            config.headers['voy-token']=token;
        }
        return config;
    }
);

export default voyApi;
```

Código 69 Consumo de la API del Back-End

Como siguiente paso se creará en la carpeta *context* dos archivos llamados *AuthContext,tsx* y *authReducer.tsx.*

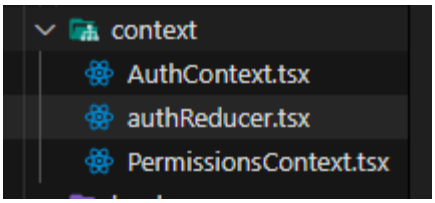


Figura 58 Creación de los archivos AuthContext y authReducer

En el archivo *authReducer* se creará un *reducer* que permitirá establecer varios estados, de tal manera que se puedan manipular por toda la ampliación.

```
export interface AuthState {
    status:'cheking'|'authenticado' | 'no-autenticado';
    token:string | null;
    errorMessage: string;
    user:Usuario| null;
    contacto:Contacto| null;
    contactos:Contacto[]| null;
    alertas:Alerta[]|null;
    oneSingup:boolean|null;
}


export type AuthAction =
            | {type: 'singUp', payload: {token:string , user:Usuario}}
            | {type: 'addError', payload:string}
            | {type: 'removeError'}
            | {type: 'noAutenticado'}
            | {type: 'logout'}
            | {type: 'loadContact',payload: { contacto:Contacto}}
            | {type: 'loadContacts',payload: { contactos:Contacto[]}}
            | {type: 'addErrorRegisterContact', payload:string}
            | {type: 'loadAlertas',payload: {alertas:Alerta[]}}
            | {type: 'removeOneSingUp'}
            | {type: 'OneSingUp'}
```

Código 70 Implementación del authReducer parte 1

En Código 70 se crea una interfaz para asignar el tipado de los datos que se utilizaran, así mismo se crea el ***authAction*** que determina que acción realizaremos y como lucirá

Como en última instancia para el *authReducer* asignara las acciones para cada estado como se muestra el Código 71.

```
export const authReducer = (state:AuthState, action:AuthAction) : AuthState =>{
        switch (action.type) {
            case 'addError':
                 return{
                     ...state,
                     user:null,
                     status:'no-autenticado',
                     token:null,
                     errorMessage:action.payload
                 }
             case 'removeError':
                  return{
                      ...state,
                      errorMessage:''
                  }
             case 'singUp':
                 return {
                       ...state,
                       errorMessage:'',
                       status:'authenticado',
                       token:action.payload.token,
                       user:action.payload.user,
                 }
             case 'OneSingUp':
                    return {
                        ...state,
                        oneSingup:true,
                    }
             case 'removeOneSingUp':
                 return {
                     ...state,
                     oneSingup:false,
                 }
             case 'logout':
             case 'noAutenticado':
                 return {
                     ...state,
                     status:'no-autenticado',
                     oneSingup:null,
                     token:null,
                     user:null,
                 }
             case 'addErrorRegisterContact':
                 return {
                     ...state,
                     errorMessage:action.payload,
                 }
```

Código 71 Implementación del authReducer parte 2

Posteriormente se desarrollará el *AuthContext,tsx.*

```
type AuthContextProps = {
    errorMessage:string; /* Cuando un error suceda se utilizará esta variable para alamacenarlos aquí */
    token: string | null; /* Nos permitira saber si el usuario esta autenticado */
    user:Usuario| null;
    contacto:Contacto|null;
    contactos:Contacto[]| null;
    alertas:Alerta[]|null;
    /* El estatus nos permite indicar si esta en login o si ya esta autenticado */
    status:'cheking'|'authenticado' | 'no-autenticado';
    oneSingup:boolean|'cheking'|null,
```

Código 72 Implementación del authContext parte 1

En el Código 72 se muestra el tipado de las *Props* dentro de estás se asignarán los estados y la forma de como lucen.

```
export const AuthContext = createContext({} as AuthContextProps);

const authInitialState:AuthState = {
    status: 'cheking',
    token:null,
    user:null,
    contacto:null,
    contactos:null,
    alertas:null,
    errorMessage:'',
    oneSingup:null,
}
```

Código 73 Asignación de valores a los estados iniciales

En el código 73 se asignan a las variables iniciales el valor correspondiente al cargar la aplicación.

```
export const AuthProvider = ({children} :any) =>{
```

Código 74 Declaración AuthProvider

Se declarará el método del *AuthProver* donde dentro de este se asignarán las funciones y métodos para lanzar las peticiones hacia el Back-End.

Una vez empleada la conexión y comenzado el desarrollo del *AuthProvider* se harán las peticiones, declarando los métodos correspondientes mandando los parámetros que se reciban del formulario.

Dentro del *AuthProveder* se declara el método *singIn* como se muestra el Código 75.

```
const singIn= async({correo, password}:loginData)=> {
    try {
        const {data} = await voyApi.post<LoginResponse>('/auth/login',{correo, password});
        dispatch({
            type:'singUp',
            payload:{
                token:data.token,
                user:data.usuario
            }
        });
        /* Se graba el token de manera persistente  */
        await AsyncStorage.setItem('token',data.token);
    } catch (error:any) {
        console.log(error.response.data);
        dispatch({type:'addError', payload: error.response.data.msg || 'Información incorrecta' })

    }
};
```

Código 75 Implementación del método singIn

Dentro del Código 75 al hacer uso de ***axios*** permitirá mandar las peticiones HTTP, en este caso se accederá a la ruta ***"/auth/login"*** en caso de tener éxito se hará el *dispatch* de la acción para cambiar los valores del estado, mandando los datos del usuario y el token, así mismo se guardará el token manteniéndolo de forma persistente en memoria para mantener la sesión activa.

En la configuración de la pantalla *login* se hará el llamado de los métodos y se creará la función que permitirá enviar los parámetros del formulario hacia la petición.

```
const {singIn,errorMessage,removeError} = useContext (AuthContext );

const onLogin  = () => {
        console.log({email,password});
        Keyboard.dismiss();
        singIn({correo:email, password});
}
```

Código 76 Uso del AuthCOntext y declaración de la función onLogin

Como se muestra en el código 76 se hace la importación de los métodos del *AuthContext* extrayendo la función *singIn* para la autenticación del usuario

```
<TouchableOpacity style={[styles.btn,styles.btnDark,stylesL.btnDimension]}
                  activeOpacity={0.08}
                  onPress={() => {onLogin() } }
>
```

Código 77 Implementación de la función onLogin en el botón

En el Código 77 se hace el llamado de la función cuando el usuario presiona el botón, al tener la implantación con la configuración de la navegación en caso de tener éxito nos podrá redireccionar a la pantalla del *Menu Lateral* en la que se encuentra principalmente la pantalla de *Inicio.*

Una configuración adicional se hará para que en caso de que la autenticación ya se haya realizado siempre muestre la pantalla de *inicio* y no el *login,* para eso se hará la configuración del archivo *StackNavigator.tsx*

```
{
  /* Verivicamos si los permisos de la ubicación estan habilitados */
  (permissions.locationStatus === 'granted')?
    (
      /* Verificamos si esta autenticado el usuario */
      (status!== 'authenticado')?
        (<>

        <Stack.Screen name="LoginScreen" options={{title:'Login'}} component={LoginScreen} />
        <Stack.Screen name="RegistroScreen" options={{title:'Registro'}} component={RegistroScreen} />
        <Stack.Screen name="RegistroScreen2" options={{title:'Registro'}} component={RegistroScreen2} />
        </>
        )
    :
        ((oneSingup===true)?
          <Stack.Screen name="ContactoScreen" options={{title:'Registro'}} component={ContactoScreen}  initialParams={{primeraVez:true}}/>
         :
         <>
          <Stack.Screen name="MenuLateral" options={{title:'Inicio'}} component={MenuLateral}  />
         </>
        )

    )
  :<Stack.Screen name="PermissionsScreen" component={PermissionsScreen} />
}
```

Código 78 Configuración de autenticación StackNavigator

En el Código 78 se muestran funciones ternarias que determinan el estado de la autenticación para poder mostrar el contenido dependiendo como se encuentre dicho estado.

Registro de un Usuario

Para el registro de un usuario se seguirá la misma estrategia para implementar el llamado de las peticiones, el cual será necesario declarar un nuevo método en el *AuthContext* llamado *singUp*.

```
const singUp= async({nombre, apellido, password,correo}:RegistroDatos)=> {
    try {
        const {data} = await voyApi.post<LoginResponse>('/usuarios',{nombre,apellido,correo, password});
        console.log(data);
        dispatch({
            type:'singUp',
            payload:{
                token:data.token,
                user:data.usuario
            }
        });
        await AsyncStorage.setItem('token',data.token);

    } catch (error:any) {
        console.log(error.response.data);
        dispatch({type:'addError', payload: error.response.data.msg || 'Verifique los compos' })
    }

};
```

Código 79 Implementación función singUp en el authContext

Como se muestra el Código 79 se sigue la misma estructura para mandar la petición, pero en este caso la ruta y el método que se envía es distinta, ya que en la previa creación del Back-End se generaron las rutas que permitirían acceder a las peticiones CRUD por lo que solo se hace uso de las mimas para la situación que se requiera.

Un detalle a considerar es que cuando el usuario se registre por primera vez se mostrara la pantalla de *configuración de contactos*, la cual al momento de accionar la función *onRegister* cambiara el estado de la aplicación asignándole una bandera que indique que es la primera vez que inicia sesión y necesita configurar sus contactos.

```
const onRegister  = () => {
    Keyboard.dismiss();
    if(validaForm() && confirma){
        singUp({
            nombre,
            apellido,
            correo,
            password
        });
        primeraVez();

    }else {
        setErrorGen(true);
    }


}
```

Código 80 función onRegister

Así que en el Código 79 después de establecer la función *singUp* se hará el llamado del método *primeraVez()* que se configurara en el *AuthContext.*

```
const primeraVez = async() =>{
    dispatch({
        type:'OneSingUp',
    });
    await AsyncStorage.setItem('oneSingup','true');
}
```

Código 81 Implementación de método primeraVez

En el Código 81 se habrá indicado que ha entrado por primea vez el usuario, cambiando el estado de la aplicación para el *dispatch OneSingUp.* De este modo en el *StackNavigator.tsx* del Código 82 se hará una función ternaria que determina si ha sido su primera vez o no a través de la variable ***oneSingup*** para mostrar la pantalla de configuración.

```
(permissions.locationStatus === 'granted')?
  (
    (status!== 'authenticado')?
      (<>
      <Stack.Screen name="LoginScreen" options={{title:'Login'}} component={LoginScreen} />
      <Stack.Screen name="RegistroScreen" options={{title:'Registro'}} component={RegistroScreen} />
      <Stack.Screen name="RegistroScreen2" options={{title:'Registro'}} component={RegistroScreen2} />
      </>
      )
:
      ((oneSingup===true)?
        <Stack.Screen name="ContactoScreen" options={{title:'Registro'}} component={ContactoScreen}  initialParams={{primeraVe
       :
       <>
        <Stack.Screen name="MenuLateral" options={{title:'Inicio'}} component={MenuLateral}  />
       </>
      )

  )
:<Stack.Screen name="PermissionsScreen" component={PermissionsScreen} />
)
```

Código 82 Función ternaria para el cambio de estado

Registro de Contactos

Para el registro de contactos se creará en el *AuthContext* el método *registraContacto* como se muestra en el Código 83.

```
const registaContacto= async({nombre, telefono,Usuario_idUsuario}:Contacto) => {
    try {
        console.log("Registrando ... ");
        console.log(nombre," | ",telefono, " | ",Usuario_idUsuario)
        const resp = await voyApi.post<Contacto>('/conalerta',{nombre,telefono, Usuario_idUsuario});
        console.log(resp.data);
        dispatch({
            type:'loadContact',
            payload:{
                contacto:resp.data,

            }
        });
        console.log(resp.data.idContacto);
        return resp.data.idContacto;
    } catch (error:any) {
        console.log(error.response.data);
        dispatch({type:'addErrorRegisterContact', payload: error.response.data.errors[0].msg || 'Es posible que el numero del contacto ya exis
        return 'error';
    }
};
```

Código 83 Implementación registraContacto

Para este apartado del Código 83 se hará la petición *POST* hacia ***"/conalerta***". Esta ruta tiene una característica la cual hará que registre al contacto en la tabla **Contacto** de la base de datos, pero a su vez que creara tres registros de las alertas (verde, amarilla y roja) que serán asociadas a dicho contacto indicando en los registros que se encuentran en un estado de no habilitado, esto con la finalidad de que sea más sencillo el cambio de estados de las alertas.

A su vez estando en el mismo archivo del *AuthContext* se creará un método llamado *actualizaAlertas.*

```
const actualizaAlertas  = async(idAlerta:number, estado:number) => {
    try {
        console.log("Actualizando Alertas ... "+idAlerta);
        const resp = await voyApi.put<Alerta[]>('/alertas/'+idAlerta,{estado});
        return resp.data;
    } catch (error:any) {
        dispatch({type:'addErrorRegisterContact', payload: error.response.data.errors[0].msg|| 'Es necesario seleccionar una alerta'});
    }

};
```

Código 84 Implementación del método actualizaAlertas

En el Código 84 se implementa el método permitirá actualizar el estado de la alerta que el usuario seleccione respectivamente ya sea para activarla o desactivarla.

Una vez creados los métodos del *AuthContext* se procederá a implementar el accionado de la funcionalidad en la pantalla Configuración de Contactos.

Recordando la pantalla de la Figura 30 se tiene un botón para agregar un contacto el cual cuando se acciona abre los contactos del dispositivo y extrae la información de dicho contacto como su nombre y el número de teléfono

Obtención de los parámetros

Llamado del método registraContacto

```
    .then(async selection => {
        if (!selection) {
            return null;
        }

        let { contact, selectedPhone } = selection;
        let snumber= selectedPhone.number.replace(/[^0-9]/g, '');
        let idContact=await registaContacto({idContacto:'',indice:0,nombre:contact.name, te
        if(idContact ==='error') return;
        sethabilitaCampoID(idContact);
        await getContact();
        setVacio(false);
        setVerificaCampos(true);
        return selectedPhone.number;
    });
}
```

Código 85 Implementación del método registraContacto en la funciono getPhoneNumber de la pantalla ContactoScreen

El Código 85 extraerá los datos del contacto para registrarlos a través de la petición POST, adicionalmente se cambiarán los estados de los *useStates* para habilitar el componente y mostrar la información de dicho contacto.

Cabe mencionar que para poder mostrar los datos del usuario en pantalla se creara un componente que será reutilizable, ya que este se asignara dinámicamente respecto al número de contactos que se obtengan, por lo cual se desarrollara el componente de la Figura 59.

Figura 59 Componente ContactoAdd

Componente ContactoAdd

Para este componente se realizará como las pantallas ya realizadas, solo que se pasaran los parámetros y se configuraran los botones de las acciones.

Como primera instancia se creará una carpeta en los componentes llamada *Contacto* en dicha carpeta se creará un archivo TSX llamado ConctactoAdd.tsx como se muestra en la Figura 60.

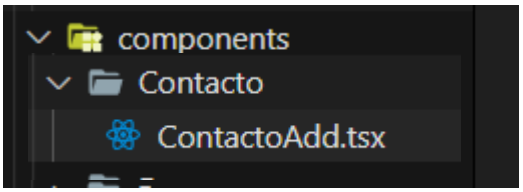


Figura 60 Creación del archivo ContactoAdd.tsx

En dicho archivo se creará una función que permita actualizar los datos de las alertas asociadas al contacto.

```
const actualizaCampos= async()=>{
    if(selected.length===0) return Alert.alert('Error al guardar','Es necesario asignarle una alarma al contacto');
    for(var i=0; i<alertasData.length;i++){
        let estado=0;
        for(var j=0; j<selected.length;j++){
            if(alertasData[i].tipo===selected[j]){
                estado=1;
            }
        }
        await actualizaAlertas(alertasData[i].idAlerta,estado)
        setVerificaCampos(false);

    }
    await actualizaContacto(idContacto,nombre,telefono);
    setHabilitaCampos(false);
    setSelectedPrincipal(true);

}
```

Código 86 Implementación de la función acutilizaCampos

En el Código 86 se verifica si se seleccionó alguna alerta antes de guardar los cambios, en caso contrario lanzara un mensaje de error, posteriormente se iterará sobre las alertas seleccionadas las cuales recibirán el estado 1, que significa que estarán activas, en cambio las que no se encuentren en el estado 1 se les asignara el estado 0 para poder así enviar el arreglo de las alertas a través del método *actualizaAlertas del AuthContext.* Del mismo modo se actualizará los datos del contacto en caso de que estos sean modificados a través del método *actualizaContacto.*

También para este código implementa algunos *useState* para ocultar, mostrar o editar el contenido.

Cabe mencionar que para poder cargar las alertas asociadas a el contacto del mismo modo se tiene que enviar una petición a la base de datos para extraer el estado de las tres alertas asociadas al contacto, por lo que se tendrá que crear un método que permita obtener dichas alertas, la cual en el *AuthContext* se creará un método llamado *obtenerAlertasContacto.*

```
const obtenerAlertasData = async ()=>{
    const a=await obtenerAlertasContacto(idContacto) as Alerta[];
    const n =[];
    const alertas:DataAlerta[] =[];

    for(let i = 0 ; i<a.length ; i++){
        let s='';
        switch(a[i].tipo){
            case 'verde':
                if(a[i].estado===1)
                n[i]='1';
                s='1';
                break;
            case 'amarilla':
                if(a[i].estado===1)
                n[i]='2';
                s='2';
                break;
            case 'roja':
                if(a[i].estado===1)
                n[i]='3';
                s='3';
                break;
        }
        alertas[i]={idAlerta:a[i].idAlerta,estado:a[i].estado,tipo:s};
    }
    const activos=[];
    let j=0;
    for(let i=0; i<n.length; i++){
        if(n[i]!==undefined){
            activos[j]=n[i];
            j++;
        }
    }
    setAlertasData(alertas);
    setSelectedPrincipal(activos.length===0? false: true);
    setSelected(activos);


    return n;
                                                              :s};
}
```

Código 87 Implementación función obtenerAlertasData

Una vez creado el método se implementará una función en el componente ContactoAdd.tsx llamada *obtenerAlertasData* como se muestra en el Código 87.

Dicha función iterara sobre los tres registros obtenidos que corresponden a las alertas asociadas al contacto y se guardaran en un arreglo las que se encuentren activas, para que de esta manera también a través del componente el usuario identifique que alertas tiene activas para ese contacto

Como última función a implementar en el componente *ContactoAdd* se creará una función llamada *deleteConfirm* la cual se encargará de eliminar el registro tanto del usuario como de las alertas asignadas a dicho contacto.

```
const deleteConfirm =async () => {
    await eliminaContactoAlerta(idContacto);
    await getContact();
    setVerificaCampos(false);
}
const eliminaItem =  () =>{
    Alert.alert('Se eliminara el contacto', '¿Estás seguro de querer eliminar el contacto?',
    [
        {
            text:'Si',
            onPress:()=>deleteConfirm()
        },
        {
            text:'No',

        }
    ]
    );

}
```

Código 88 Implementación de la función deleteConfirm

En el Código 88 se hace el llamado de la función *eliminaContactoAlerta* del *AuthContext* la cual se implementará como las anteriores peticiones solo asignándole la ruta de la función correspondiente de eliminación.

Cuando el usuario active la opción de eliminar el registro le lanzará un mensaje que le indique que se procederá a eliminar el registro, en caso de que el usuario este de acuerdo se llamar a la función *deleteConfirm* y hará la eliminación definitiva de la base de datos.

Una vez configurado el componente *ContactoAdd* se creará un *FlatList* que cargue el contenido de los registros de los contactos con sus respectivas alertas en caso de que exista alguno asignado.

Para la implementación del *FlatList* en la pantalla ContactoScreen.tsx se hará el llamado de la función del *AuthContext* llamada *obtenerContactos* esta función nos permitirá cargar los registros de los contactos ya configurados respectivamente, en caso contrario de que no existan se mostrara solo una leyenda de que no hay registros.

```
const getContact =  async() => {
    let m = await   obtenerContactos(idUsuario) as Contacto[];
    for(var i=0 ; i< m.length ; i++){
        m[i].indice=i+1;
    }
    m.reverse();
    (m.length===0)?setVacio(true):setContactos(m);
}
```

Código 89 Implementación de la función getContact

Como se muestra en el Código 89 se implementa la función *getContact* que hace el llamado de la petición *obtenerContactos* , donde estos se almacenaran en un arreglo enviándolos a través de un *useState*, en caso de que no exista un registro se marcara una bandera que indique que no hay registros.

```
{ (!vacio)?
<FlatList  data={contactos}  renderItem={({item}) =>
          renderContactsItem(item,verificaCampos, setVerificaCampos,
                             habilitaCampoID,
                             getContact)}
          keyExtractor={(item)=> item.idContacto}
          refreshControl={
              <RefreshControl
                refreshing={refresh}
                onRefresh={onRefresh}
              />
          }
 />:
```

Código 90 Implementación del FlatList

Una vez obtenido los datos se pasarán al *FlatList* como se muestra en el Código 90 el cual se encargará de enlistar los datos a través del componente AddContact, para eso se implementará una función llamada *renderContactsItem* donde se pasarán algunas funcionalidades independientes para darle la funcionalidad al componente.

```
const renderContactsItem = (contactItem:Contacto,verificaCampos:boolean,
                            setVerificaCampos:(bol:boolean) => void,
                            habilitaCampoID:string,
                            getContact:() => void
                            ) =>{
    return(
        <>
        <SafeAreaView style={{flex: 1}}>
         <ContactoAdd getContact={getContact}  habilitaCampo={(habilitaCampoID===contactItem.idContacto)?true:false}
                      verificaCampos={verificaCampos}
                      setVerificaCampos={setVerificaCampos} idUsuario={idUsuario}
                      idContacto={contactItem.idContacto}
                      noLabel={contactItem.indice}
                      alias={contactItem.nombre}
                      numero={contactItem.telefono}/>
        </SafeAreaView>
        </>
    )
}
```

Código 91 Implementación del renderContactsItem

Como se muestra en el Código 91 se puede ver el llamado del componente *ContactAdd* donde se pasa la información necesaria para cargar los datos del contacto.

Editar Información del Usuario

Para agregar la funcionalidad a la pantalla de *CuentaScreen.tsx* de la Figura 46 se realizará el mismo procedimiento de peticiones a través del *AuthContext* para eso se hará la configuración del llamado de las funciones para cargar los datos en los formularios.

```
const {user,actualizaUsuario} = useContext(AuthContext);
```

Código 92 Llamado de la función actualizaUsuario y el estado user del AuthContext

En el archivo *CuentaScreen.tsx* se creará un componente llamado *Formulario* el cual se implementará para crear un *FlatList*, dicho componente se utilizará para que cargue la información en cada campo, manteniendo el mismo diseño y acciones de edición.

Una vez en el componente *Formulario* declarará el *useContext* del *AuthContext* para extraer la función de *actualizaUsuario* que permitirá actualizar los datos del usuario como: nombre, apellido, correo y contraseña.

```
const actualiza = async(tipo:string, dato:string) => {
    if(validaForm()===true){
        await actualizaUsuario(idUsuario,tipo,dato);
        setHabilitaBtn(false);
        setTemp(etiqueta);
    }else{
        setErrorGen(true);
    }
};
```

Código 93 Implementación de la función actualiza para editar los datos del usuario

En el Código 93 se hace el llamado a la función de la petición para editar el dato que se desea actualizar, en donde así mismo se utilizan algunos *useState* para algunas validaciones del formulario y muestreo de los componentes.

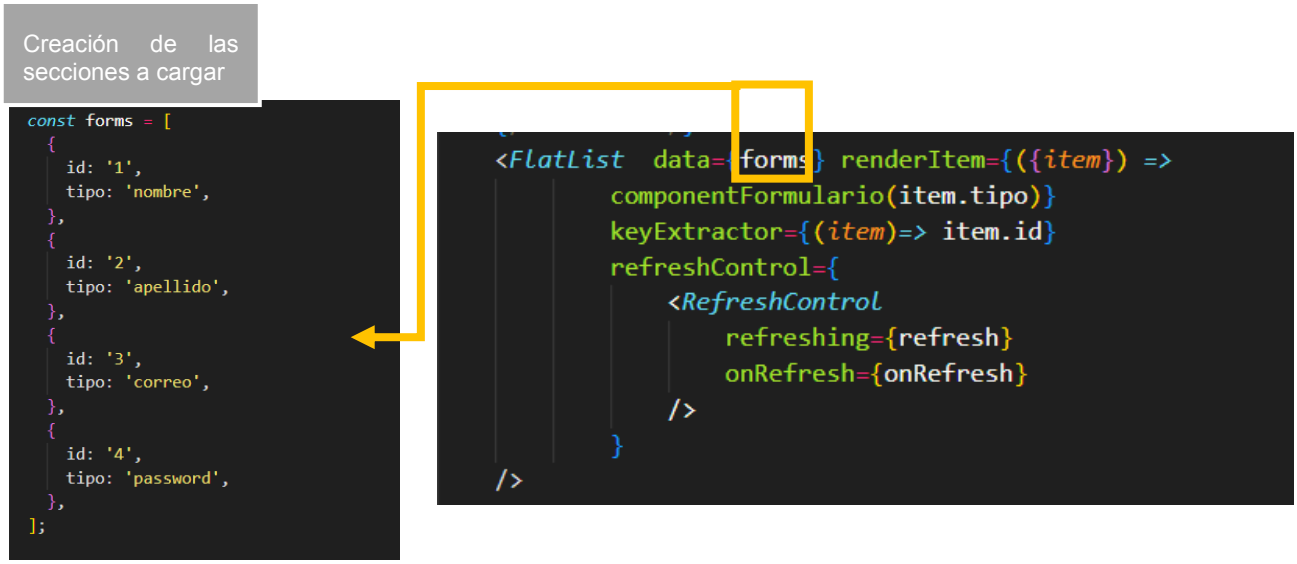
En el componente principal *CuentaScreen* se hará el llamado del *AuthContext* extrayendo los datos del usuario y la función *eliminaUsuario* como se muestra en el Código 94.

```
export const CuentaScreen = ({navigation}:Props) => {

    const {user,eliminaUsuario} = useContext(AuthContext);
    const usuar= JSON.stringify(user,null);
    const usuario= JSON.parse(usuar);
    const idUsuario= usuario.idUsuario;
```

Código 94 Llamado del AuthContext para la función elimnaUsuario

Así mismo se hará un parseo de la información del usuario como se muestra en el Código 94, ya que la variable ***user*** contiene la información del Usuario ya que cuando se autentica dicha información se establece, ya que fue configurada en el estado del *dispatch.*

Una vez declarados los componentes y las funciones se implementarán el *FlatList* Como se muestra en el Código 95.



Código 95 Implementación FlatList para los datos del Usuario

Una vez implementado los componentes, se asignarán a los botones las acciones del *onPress* para accionar las funcionalidades implementadas para editar la información y la eliminación definitiva de la cuenta del Usuario.

Implementación Módulo de emisión de alertas

En esta sección se desarrollará la funcionalidad de la pantalla de *Inicio,* la cual corresponde a la activación de los botones de riesgo, es por lo que se harán los llamados a las funciones del *AuthContext* y la obtención de la ubicación del usuario a través del *Hook useLocation*.

Como primera instancia se harán los llamados a las funciones del *AuthContext*, la cual se sigue la misma estrategia de solicitar las peticiones HTTP a través de las rutas con **axios**.

```
const {user,obtenerAlertas,obtenerContacto,
       registraUbicacionAlerta,errorMessage,
       removeError} = useContext(AuthContext);
```

Código 96 Llamado del AuthContext para la obtención de las funciones a implementar

En el componente *InicioScreen* se declarará el Código 96 para hacer el llamado de las funciones que nos interesaran utilizar.

```
const usuar= JSON.stringify(user,null);
const usuario= JSON.parse(usuar);
const idUsuario= usuario.idUsuario;

const [alertas, setAlertas] = useState<Alerta[]>([]);

const [alertaVerde, setAlertasVerde] = useState<string[]>([]);
const [alertaAmarilla, setAlertasAmarrilla] = useState<string[]>([]);
const [alertaRoja, setAlertasRoja]    = useState<string[]>([]);
```

Código 97 Implementación de useState y parseo de la información del usuario

En el Código 97 se declararán los *useState* ya que nos peritarán mandar los estados del almacenamiento de las respuestas del *AuthContext* una vez manipuladas en las funciones a implementar más adelante.

```
const obtenerAlertasData = async ()=>{
    const a=await obtenerAlertas(idUsuario) as Alerta[];
    setAlertas(a);
    const v: string[]=[];
    const y: string[]=[];
    const r: string[]=[];
   for(let i=0; i<a.length;i++){
       if(a[i].estado===1){
           console.log(a[i].Contacto_idContacto);
           const s=await obtenerContacto(a[i].Contacto_idContacto) as Contacto;
           switch(a[i].tipo){
               case 'verde':
                   v.push(s.telefono);
                   break;
               case 'amarilla':
                   y.push(s.telefono);
                   break;
               case 'roja':
                   r.push(s.telefono);
                   break;
           }
       }
   }
   setAlertasVerde(v);
   setAlertasAmarrilla(y);
   setAlertasRoja(r);
   return a;
}
```

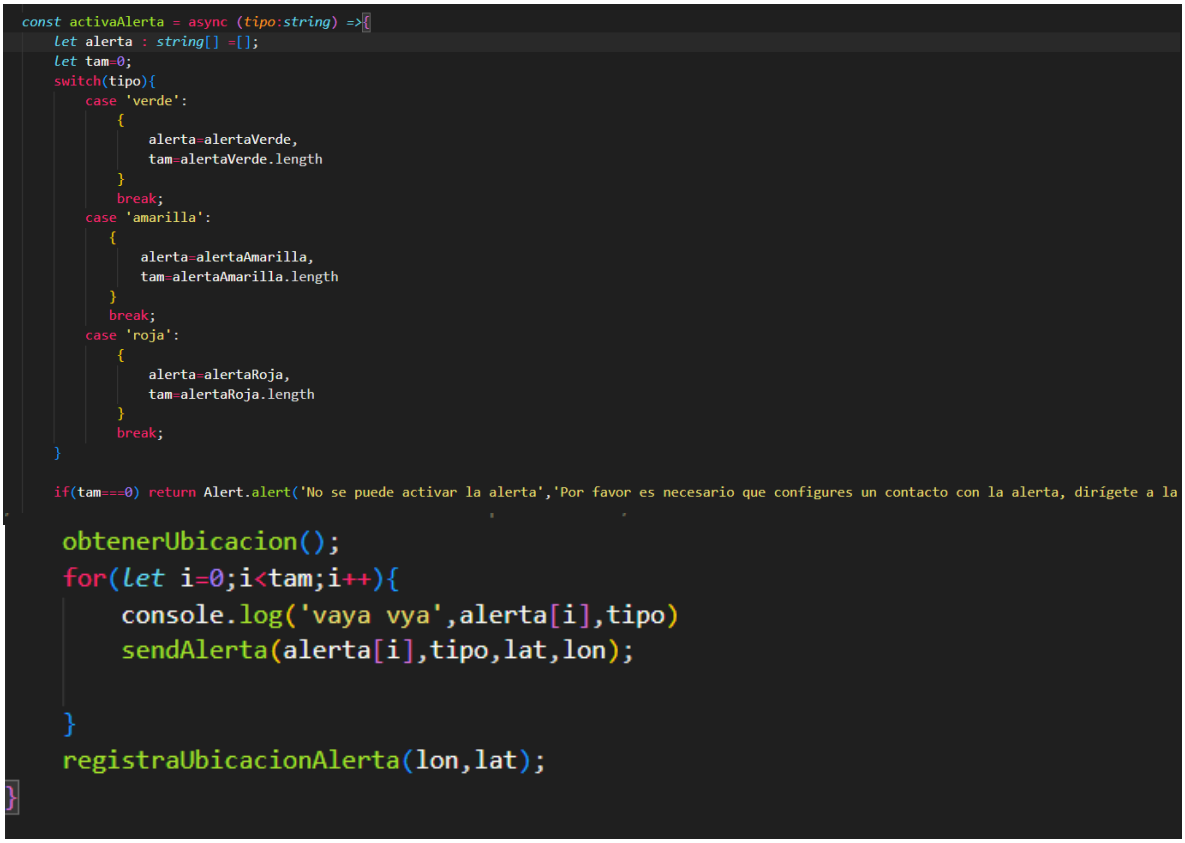
Código 98 Implementación de la función obtenerAlertasData

En el Código 98 se implementa la función que permitirá obtener las alertas que se han configurado respectivamente. Como primera instancia se manda la solicitud para obtener todas las alertas que estén asociadas al usuario, cuando se obtengan dichas alertas se iterara para obtener los datos del contacto mandando otra solicitud a la base de datos para extraer la información del contacto de dicha alerta, una vez cuando se obtenga la información se almacenara en un arreglo correspondiente a el tipo de alerta, ya sea verde, amarrilla o roja, es decir se asignara a un arreglo el teléfono del contacto en el *useState* del arreglo del tipo de alerta.

```
const{ getCurrentLocation} = useLocation();

const [lat, setLat] = useState(0);
const [lon, setLon] = useState(0);

const obtenerUbicacion = async () => {
   const {latitude,longitude} =await getCurrentLocation();
   setLat(latitude);
   setLon(longitude);
}
```

Código 99 Implementación de la función obtenerUbicacion

En el Código 99 se hace el llamado del *getCurrentLocation* del Hook que se implementó en secciones anteriores, el cual permitirá obtener la longitud y la latitud de las coordenadas del usuario el cual al momento de desestructurar se mandaran a través del *useState* para que sean utilizadas al momento de activar la alerta.



```
const activaAlerta = async (tipo:string) =>{
    let alerta : string[] =[];
    let tam=0;
    switch(tipo){
        case 'verde':
            {
                alerta=alertaVerde,
                tam=alertaVerde.length
            }
            break;
        case 'amarilla':
            {
                alerta=alertaAmarilla,
                tam=alertaAmarilla.length
            }
            break;
        case 'roja':
            {
                alerta=alertaRoja,
                tam=alertaRoja.length
            }
            break;
    }

    if(tam===0) return Alert.alert('No se puede activar la alerta','Por favor es necesario que configures un contacto con la alerta, dirígete a la

    obtenerUbicacion();
    for(let i=0;i<tam;i++){
        console.log('vaya vya',alerta[i],tipo)
        sendAlerta(alerta[i],tipo,lat,lon);

    }
    registraUbicacionAlerta(lon,lat);
}
```

Código 100 Implementación de la función activarAlerta

En el Código 100 se implementa la función *activaAlerta* la cual determinara el tipo de alerta que se accionara, para que de esta manera obtenga el arreglo correspondiente a la aleta de los teléfonos obtenidos. Una vez determinado el tipo de alerta con el arreglo se llamará la función de *obtenerUbicacion* para así saber las coordenadas exactas del momento de la activación la alerta, una vez llamada la función se iterará sobre el arreglo de la alerta para mandar el mensaje a los números y esto se logrará a través de la función *sendAlerta*. Por último, se registrará la ubicación de la activación de la alerta para marcar en el mapa virtual ese punto de donde se emitió.

Para poder configurar la función *sendAlerta*, se creará una carpeta llama *Alerta* y un archivo TSX llamado *sendAlerta.tsx* como se muestra la Figura 61.

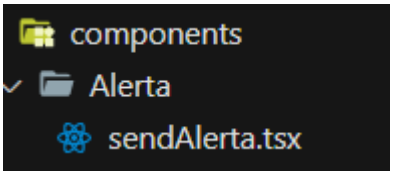


Figura 61 Creación del archivo sendAlerta.tsx

Una vez creado el archivo se creará un componente llamado *sendAlerta* en el cual se asignará una función llamada *sendDirectSms*.

Enlace para ubicar a el usuario

Permisos de envio de mensaje de texto

```
export const sendAlerta = async (numero:string, alerta:string,latitude:number,longitude:number) => {


    let mensaje= '';
    let ubicacionTxt='http://maps.google.com/maps?q='+latitude+','+longitude;


    const sendDirectSms = async () => {
        if (numero) {
        try {
            const granted = await PermissionsAndroid.request(
            PermissionsAndroid.PERMISSIONS.SEND_SMS,
            {
                title: 'Envió de Mensajes',
                message:
                'Para poder enviar el mensaje es necesario que otorgue los permisos ' +
                '¿Desea que VoyContigo pueda enviar mensajes?.',
                buttonNeutral: 'Pregúntame Despues',
                buttonNegative: 'Cancelar',
                buttonPositive: 'OK',
            },
            );
```

Código 101 Implementación de la función sendDirectSms parte 1

Cómo se muestra el Código 101 se implementará la configuración de los permisos para el envío de mensajes de texto que se describió en la sección Permisos envió de mensajes de texto.

Código 102 Implementación de la función sendDirectSms parte 2

En la implementación del Código 102 se determina el mensaje se enviará para que posteriormente a través de la función de ***DirectSms.sendDirectSms*** se envié el mensaje de texto a el número correspondiente con el mensaje predeterminado asignado de la alerta. Una vez mandado con éxito el mensaje saldrá una pantalla emergente que indicara que se ha mandado con éxito el mensaje.

Cabe mencionar que para poder accionar la función ***DirectSms.sendDirectSms*** que hará el en vio del mensaje, se tendrá que hacer una configuración para dispositivos Android.

Configuración del envió directo de mensajes para Android

Inicialmente en el proyecto de React-Native en la ruta "***Android/app/scr/main/java***" se crearán dos archivos de Java llamados *ModuloSendSms.java* y *PaqueteSendSms.java* como se muestra en la Figura 62.

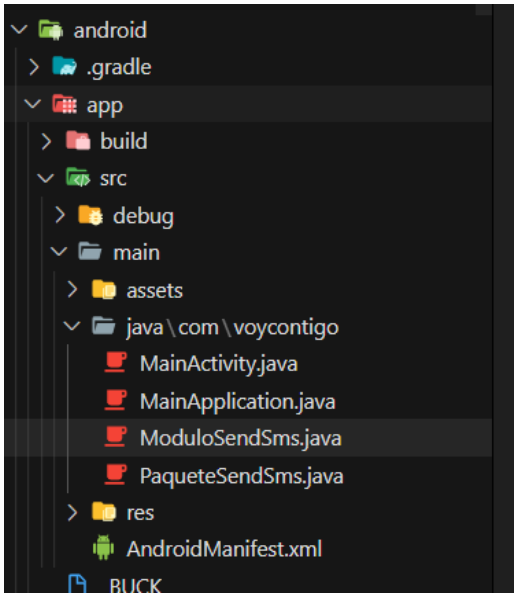


Figura 62 Creación de los archivos Java ModuloSendSms.java y PaqueteSendSms.java

Una vez creado los archivos se procedería a implementar el desarrollo del código de estos.

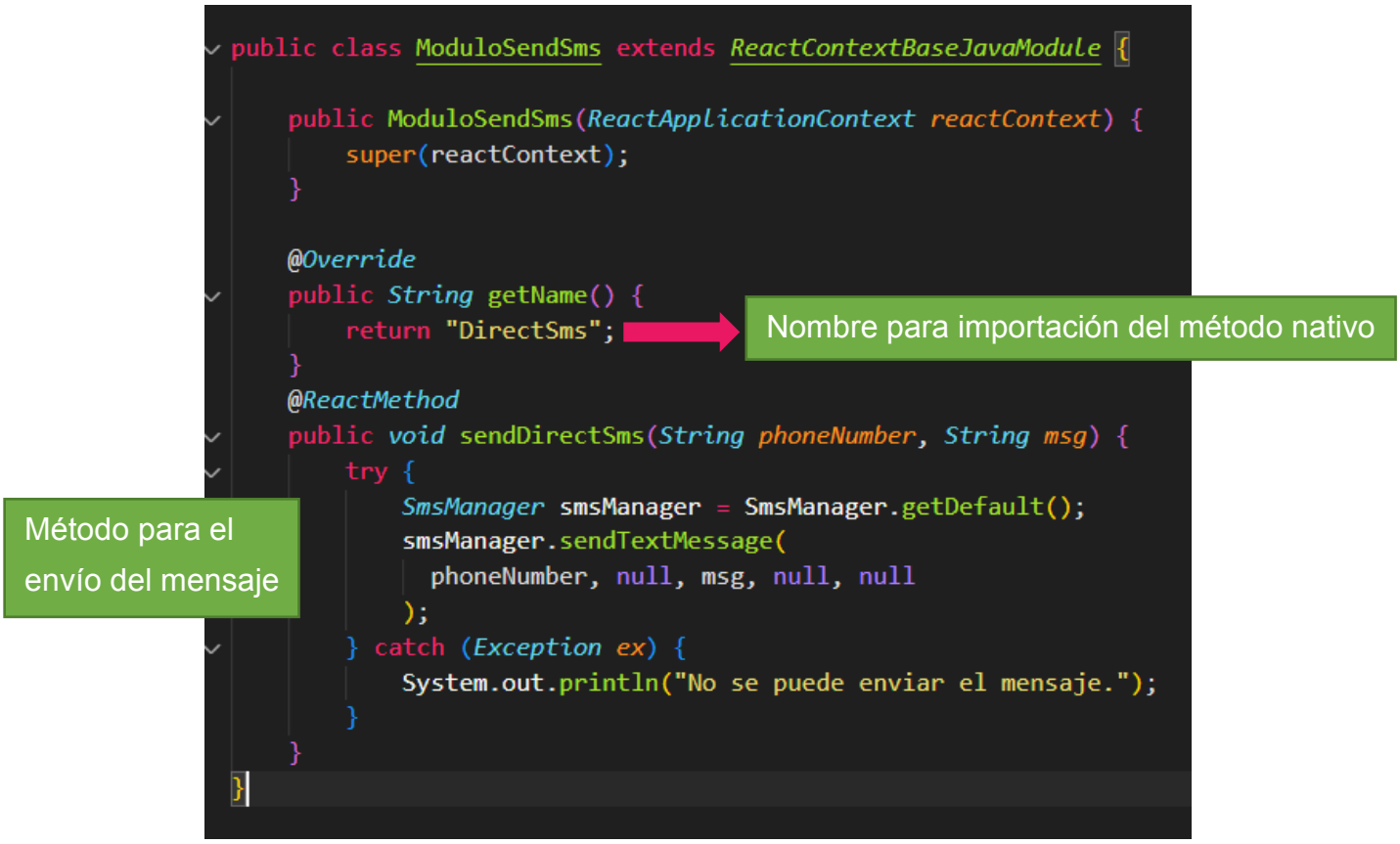
Implementación ModuloSendSms.java

Como primera instancia se procederá a hacer las importaciones de las bibliotecas a utilizar como se muestra en el Código 103.

```
package com.voycontigo;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.Callback;
import com.facebook.react.bridge.ReactMethod;
import com.facebook.react.uimanager.IllegalViewOperationException;

import android.telephony.SmsManager;
```

Código 103 Importación de las bibliotecas para la clase ModuloSendSms.java

La importación para el funcionamiento del envío del mensaje será la de "*import android.telephony.SmsManger*"




```
public class ModuloSendSms extends ReactContextBaseJavaModule {

    public ModuloSendSms(ReactApplicationContext reactContext) {
        super(reactContext);
    }

    @Override
    public String getName() {
        return "DirectSms";
    }
    @ReactMethod
    public void sendDirectSms(String phoneNumber, String msg) {
        try {
            SmsManager smsManager = SmsManager.getDefault();
            smsManager.sendTextMessage(
              phoneNumber, null, msg, null, null
            );
        } catch (Exception ex) {
            System.out.println("No se puede enviar el mensaje.");
        }
    }
}
```

Código 104 Declaración de la clase ModuloSendSms

Una vez realizado las importaciones se implementará el Código 104 para la declaración de la clase, donde se observa los métodos para hacer el llamado nativo y el método que permitirá enviar los parámetros del mensaje.

Implementación PaqueteSendSms.java

Como primera instancia de harán las importaciones respectivas de las bibliotecas a utilizar y el módulo creado como se muestra en el Código 105.

```
package com.voycontigo;
import com.facebook.react.ReactPackage;
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.uimanager.ViewManager;
import com.voycontigo.ModuloSendSms;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
```

Código 105 Importación del módulo Send Sms

```java
import com.voycontigo.ModuloSendSms;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class PaqueteSendSms implements ReactPackage {

    @Override
    public List<ViewManager> createViewManagers(
      ReactApplicationContext reactContext
    ) {
        return Collections.emptyList();
    }

    @Override
    public List<NativeModule> createNativeModules(
            ReactApplicationContext reactContext) {
        List<NativeModule> modules = new ArrayList<>();
        modules.add(new ModuloSendSms(reactContext));
        return modules;
    }
}
```

Código 106 Adición del Módulo SendSms

En el Código 106 se añade una lista del tipo *NativeModule* donde se añadirá el módulo creado *ModuloSendSms.*

Una vez creado las clases y declarado los métodos, se configurará el archivo *MainApplication.java* en el cual se hará la importación del paquete creado y se añadirá a la lista de los paquetes como se muestra en el Código 107.

```java
import com.facebook.react.ReactPackage;
import com.facebook.soloader.SoLoader;
import java.lang.reflect.InvocationTargetException;
import java.util.List;
import com.voycontigo.PaqueteSendSms;

public class MainApplication extends Application implements ReactApplication {

  private final ReactNativeHost mReactNativeHost =
      new ReactNativeHost(this) {
        @Override
        public boolean getUseDeveloperSupport() {
          return BuildConfig.DEBUG;
        }

        @Override
        protected List<ReactPackage> getPackages() {
          @SuppressWarnings("UnnecessaryLocalVariable")
          List<ReactPackage> packages = new PackageList(this).getPackages();
           packages.add(new PaqueteSendSms());
          return packages;
        }
```

Código 107 Configuración del MainApplication

Por último, para hacer el llamado del método nativo se hará la importación de *NativeModules* de “react-native” el cual contendrá el método *DirectSms* como se muestra en el Código 108.

```
import {NativeModules,PermissionsAndroid, Alert} from 'react-native';
var DirectSms = NativeModules.DirectSms;
```

Código 108 Llamado del método NativeModules

De esta manera al tener configurado las funciones antes mencionadas, se procederá a hacer el llamado de la función *activaAlerta()* para accionar el *onPress* de los botones de las alertas, para que de esta manera se mande un mensaje de texto hacia los contactos configurados, en caso de que no exista ningún contacto asignado no se podrá accionar el botón.

```
<BtnAlerta tipo={'verde'}    onPress={() => activaAlerta('verde')}    tam={alertaVerde.length} />
<BtnAlerta tipo={'amarilla'} onPress={() => activaAlerta('amarilla')} tam={alertaAmarilla.length}/>
<BtnAlerta tipo={'roja'}     onPress={() => activaAlerta('roja')}     tam={alertaRoja.length}/>
```

Código 109 Paso de parámetros al componente BtnAlerta

En Código 109 se pasarán los parámetros al componente que se creara *BtnAlerta*, el cual corresponde a los tres botones de las alertas a accionar.

```
<View style={[stylesI.containerBotones,]}>
 <TouchableWithoutFeedback disabled={tam===0 ? true : false} onPress={onPress} onPressIn={bajaBtn} onPressOut={levantaBtn}>
        <View style={stylesI.btnArea}>
            <View style={[stylesI.btnBase]}>
            <Animated.View style={[stylesI.btnAltura,
                                   {backgroundColor: (tipo=== 'verde') ? colors.aLverdeDark : (tipo=== 'amarilla')? colors.aLamarilloDar
                                    alturaStyle]}>
                <Animated.View style={[stylesI.btnSuperior,internoBtn ,
                                       {backgroundColor:  tam===0 ? colors.secondaryDark : (tipo=== 'verde') ? colors.aLverde : (tipo===

                                       ]}>
                <Image source={require ('../img/logo.png')}
                       style={ stylesI.imgLogo }
                />
                 <Text style={stylesI.textBtn}> {(tipo=== 'verde') ?'Bajo' : (tipo=== 'amarilla')? 'Medio' : (tipo=== 'roja')? 'Alto'
                </Animated.View>
            </Animated.View>
            </View>
        </View>
 </TouchableWithoutFeedback>
 </View>
)
```

Código 110 Implementación del componente BtnAlerta

En el Código 110 se implementan funciones ternarias para asignar el contenido correspondiente a las tres alertas.

Implementación del Módulo contador de las alertas

Para este módulo se adicionará algunas configuraciones al componente *Map* para que de esta manera se asigne un contador de alertas que permita visualizar donde se han accionado los botones.

```
const obtenerUbicaciones = async() => {
    try {
        console.log("Obteniendo ubicaciones ... ");
        const {data} = await voyApi.get<UbicacionDescrip>('/ubicacion');
        const {ubicaciones, alertas} =data;
        let marker : UbicacionMarker[] = [];
        let j=0;
        let tam= ubicaciones.length;
        for(let i=0; i<tam; i++){
            if(ubicaciones[i].Alerta_idAlerta===alertas[i].idAlerta){
            marker[j]={
                        id:j,
                        longitud:ubicaciones[i].longitud,
                        latitud: ubicaciones[i].latitud,
                        fecha:ubicaciones[i].fecha,
                        hora:ubicaciones[i].hora,
                        tipo:alertas[i].tipo
                    };
                    j++;
            }
        }
        return marker;
    } catch (error:any) {
        console.log(error.response.data);
        dispatch({type:'addErrorRegisterContact', payload: error.response.data.errors[0].msg || 'Intentelo nuevamente' });
        return [];
    }
}
```

Código 111 Implementación de la función obtenerUbicaciones en el AuthContext

En el Código 111 muestra la implementación del método GET para accionar la petición a través del PHAT configurado en el Back-End para obtener los registros de las ubicaciones de las alertas accionadas.

Una vez definida la función se hará el llamado de la función en el componente *Map* como se muestra en el Código 112.

```
const {obtenerUbicaciones} = useContext(AuthContext);
const [ubicaciones, setUbicaciones] = useState<UbicacionMarker[]>([]);
const obtenerUbicacion = async () =>{
        const datos=await obtenerUbicaciones() as UbicacionMarker[];
        (datos.length===0)? null :
         setUbicaciones(datos);
        ;
}
useEffect( () =>{
    obtenerUbicacion();
}
,[]);
```

Código 112 Llamado de la función obtenerUbicaciones

Así mismo se hará uso del *useEffect* para que al cargar el componente se lance la petición y obtenga las ubicaciones y de paso a marcar las alertas que se han activado.

Una vez dicho esto se implementará las ubicaciones de las alertas que han sido emitidas, para eso se modificará el Código 60 y se añadirá los marcadores.

```
<MapView
 ref={(el)=> mapViewRef.current=el!}
   style={{flex:1}}
   showsUserLocation
        initialRegion={{
        latitude: initialPosition.latitude,
        longitude: initialPosition.longitude,
        latitudeDelta: 0.0922,
        longitudeDelta: 0.0421,
        }}
        /*ontouch es una funcion quen os marca que el usuario esta moviendo el mapa  */
        onTouchStart={() => following.current=false}
    >
        {ubicaciones.length!==0 ? ubicaciones.map((marker, id) => (
                <Marker
                key={id}
                coordinate={{latitude:marker.latitud ,
                            longitude: marker.longitud}}
                title={'Alerta '+marker.tipo}
                description={'Se acciono el día : '+ marker.fecha + ' a las '+marker.hora}
                pinColor={ (marker.tipo==='verde')? colors.alerta2
                                                  : ((marker.tipo==='amarilla')?
                                                     colors.alerta5
                                                    : (marker.tipo==='roja')? colors.alerta8
                                                    : 'pink')
                        }
```

Código 113 Implementación del conteo de las alertas

En el Código 113 se implementa una condición ternaria que determinara si existen registros de las ubicaciones de las alertas emitidas, en caso de que existan a través del componente *Marker* se marcaran con los colores de las alertas las ubicaciones donde fueron accionadas indicando el tipo de riesgo, la fecha y la hora en que se emitieron.

Resultados

Una vez desarrollada la aplicación se mostrará su funcionamiento mediante un video el cual se encuentra en el siguiente enlace.

Figura 63 Enlace al video del funcionamiento de la aplicación

Como en primera instancia en el video de la Figura 63, se realiza la interacción con la aplicación móvil simulando el ingreso de un usuario contemplando los casos de errores que pudiesen cometer más usuarios, de esta manera se observa el buen funcionamiento del Back-End, ya que a través de la aplicación móvil el Front-End reacciona ante los casos que se configuraron.

Del mismo modo se observa la navegación entre pantallas, la habilitación del contenido de cada componente y el accionado de los botones que se encuentran en cada una.

Así mismo la aplicación muestra el contenido de la información a solicitar, cargándola en los componentes de las secciones.

Análisis y discusión de resultados

Una vez desarrollada la aplicación móvil se realizaron pruebas del funcionamiento del Back-End y Front-End, de tal manera del poder comprobar los distintos escenarios en el que un usuario pudiese encontrarse ante el uso de la aplicación móvil.

Para comprobar el funcionamiento del Back-End se hizo uso de la herramienta de *Postman*, la cual ayuda a realizar pruebas de los métodos HTTP incorporados en los servicios implementados a través de las rutas que se describieron en el desarrollo del proyecto.

Funcionamiento del Back-End

Recordando un poco en la sección de desarrollo del proyecto se habían creado los métodos a través de *Postman* que definían las rutas y acciones de las peticiones como se muestra en la Figura 63.

Figura 64 Colección de métodos de las rutas del Back-End

Como se muestra en la Figura 65 se realiza una prueba de autenticación de un usuario registrado, como consecuente el Back-End responderá con un mensaje de error.

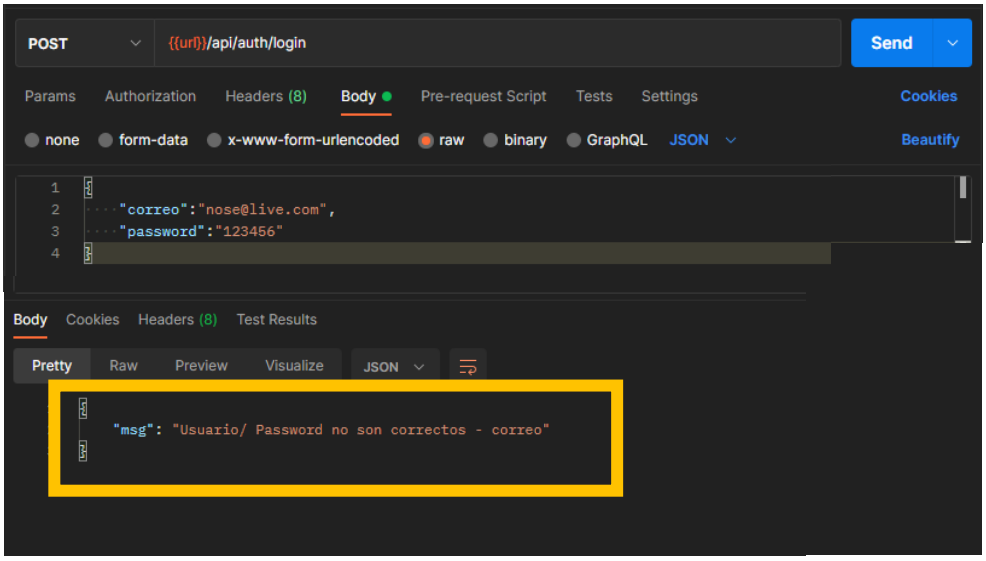


Figura 65 Prueba Login con datos incorrectos

Caso contrario en la Figura 66 se ingresa los datos de un usuario previamente registrado y como consecuente muestra los datos de este y la generación del token de sesión.

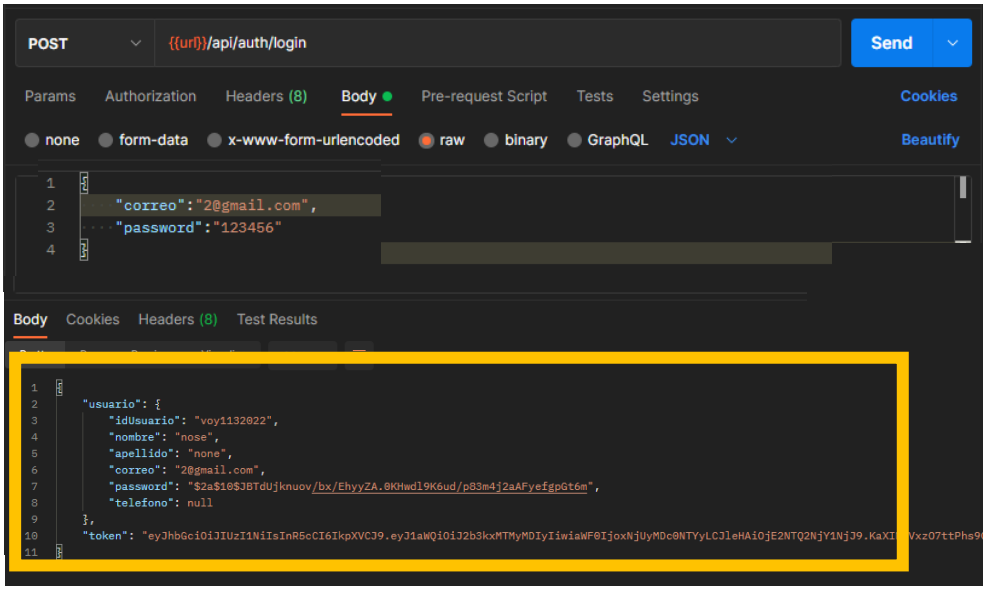


Figura 66 Prueba Login con un usuario registrado

Realizando otra prueba como se muestra en la Figura 67 para el registro de un contacto se realiza la solicitud POST con datos incorrectos, en el cual este caso contempla para el número de teléfono 7 dígitos, como consecuente mandara un mensaje indicando donde hay un error.

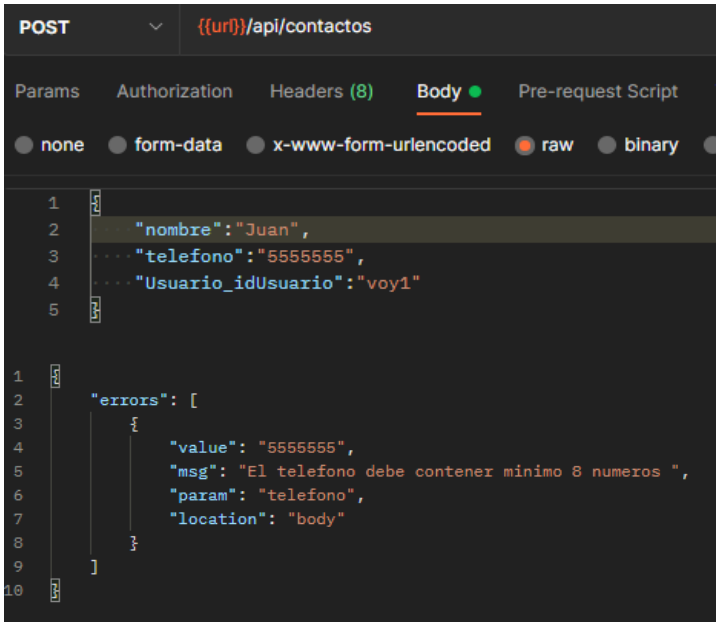


Figura 67 Método POST para registrar a un usuario con datos incorrectos

De esta manera probando los datos de cada método para cada ruta generada del Back-End se realizó el mismo procedimiento, es así como durante el proceso de *testeo* se averiguaron diversos errores en los cuales se implementaron filtros para el manejo de estos y muestreo de mensajes que permitirán mostrar al usuario e identificar donde se equivocaron.

Cabe mencionar que algunos manejos de errores no pueden ser tan trasparentes para el usuario así que se asignaron mensajes que indiquen que tienen que ser revisados por “Soporte técnico” ya que dichos errores tendrán que ver con la comunicación y desarrollo del Back-End.

Gracias a la implementación de los middlewares de las rutas generadas permite tener mejor control sobre la validación de la información permitiendo identificar fallas en el sistema.

Funcionamiento del Front-End

En el caso del Front-End para realizar diversas pruebas se hizo uso de un emulador de dispositivos virtuales Android, como también de dispositivos físicos

Como primer caso que se tiene el Inicio de la aplicación. Al iniciar la aplicación mostrada la pantalla *Loading* que verificará el estado de los permisos de acceso al GPS, el cual posteriormente mostrará la pantalla de permisos, el cual le pedirá los permisos al usuario para poder mostrar la pantalla de inicio como se muestra en la Figura 68.

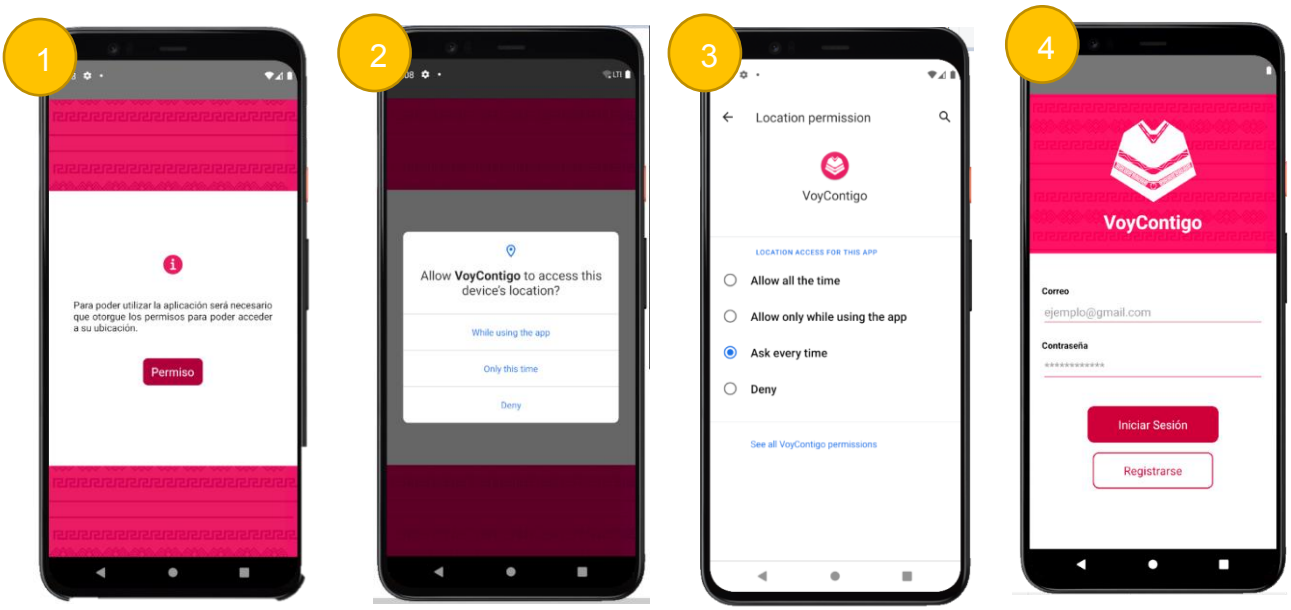


Figura 68 Casos de acceso al GPS

Al accionar el botón de “Permiso” se mostrará la pantalla 2 de la Figura 68, pero en caso de que el usuario denegué directamente en la configuración de la aplicación como se muestra en la pantalla 3, la aplicación detectara el estado de la pantalla siempre preguntando que necesita el acceso a la ubicación para poder utilizar la aplicación ya que es indispensable su uso.

Posteriormente en caso de que el usuario intente identificarse y no exista algún registro u la información sea incorrecta se mostrara un mensaje error como se muestra en la Figura 69.

Figura 69 Validación de la información de la pantalla Login

En la sección de Registro de un usuario en caso de que el usuario ingrese la información incorrecta, se contemplara una serie de errores en los que el usuario puede encontrarse como se muestra en la Figura 70, de tal manera que al mostrar el tipo de error le puede servir de ayuda para ingresar los datos correctos a el usuario.

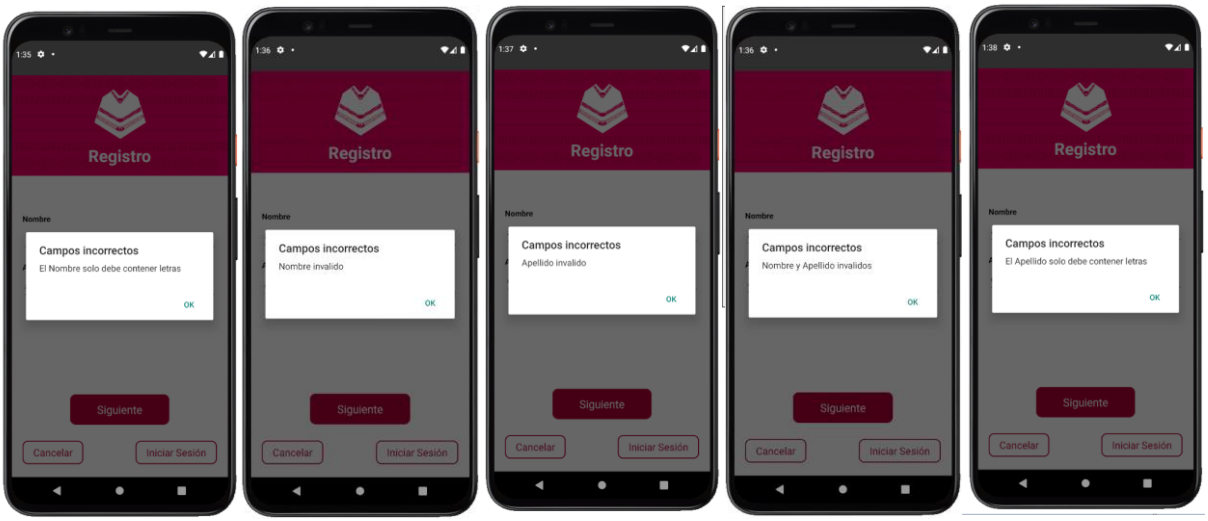


Figura 70 Casos de errores para la pantalla Registro 1

En el caso de la pantalla de Registro 2 de la misma manera contempla los errores de los formularios como se muestra en la Figura 71.

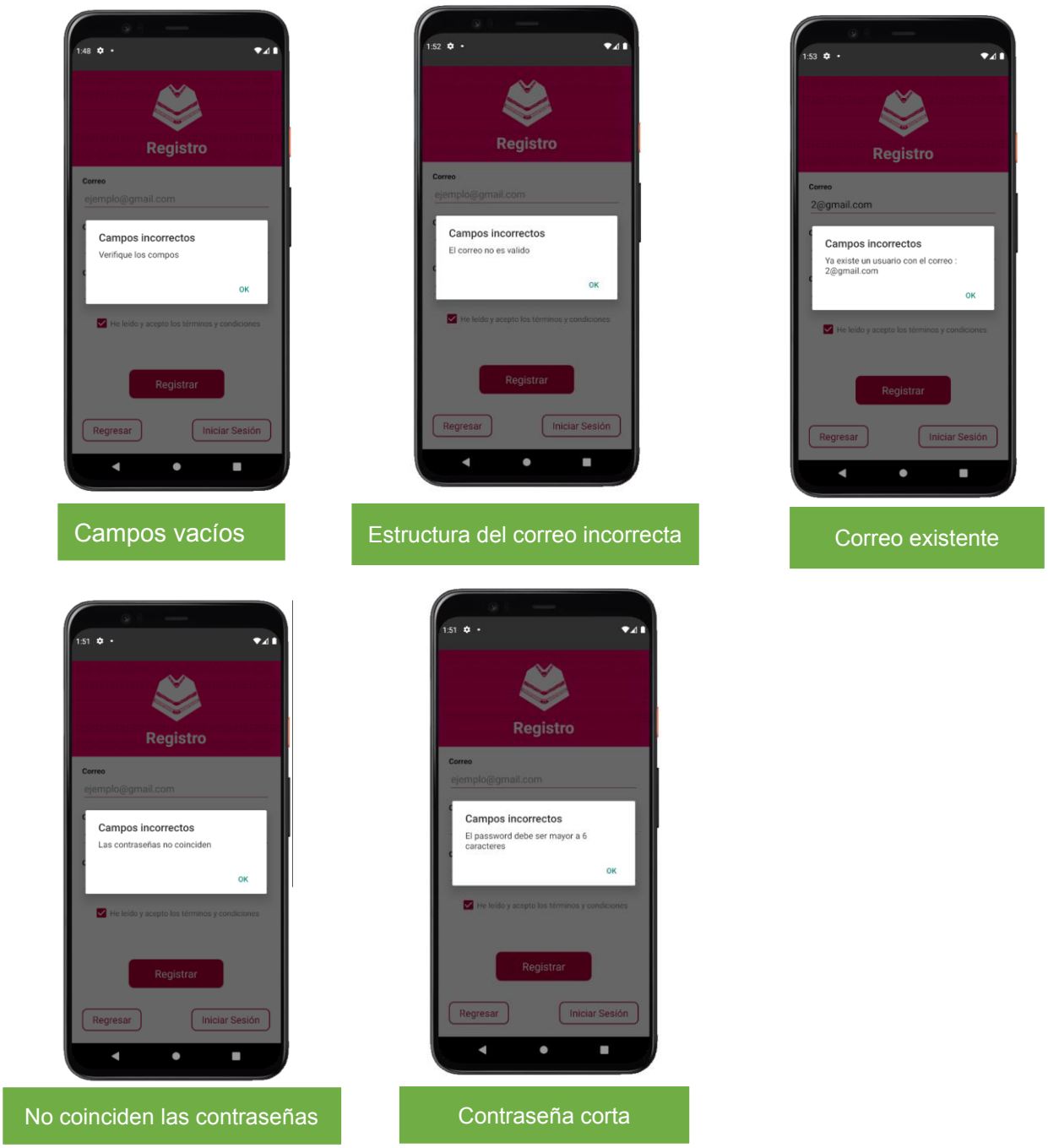


Figura 71 Casos de Errores para la pantalla de Registro 2

La pantalla Registro 2 también contempla un *checkbox* como se muestra en la Figura 72, que al presionar la leyenda o el cuadro de confirmación mostrara una pantalla emergente que indique los *Términos y Condiciones de uso de la aplicación,* esto con el objetivo de deslindar la responsabilidad de cualquier incidente o mal uso de la misma, donde es indispensable que el usuario acepte los *términos y condiciones* en caso de que quiera utilizar la aplicación.

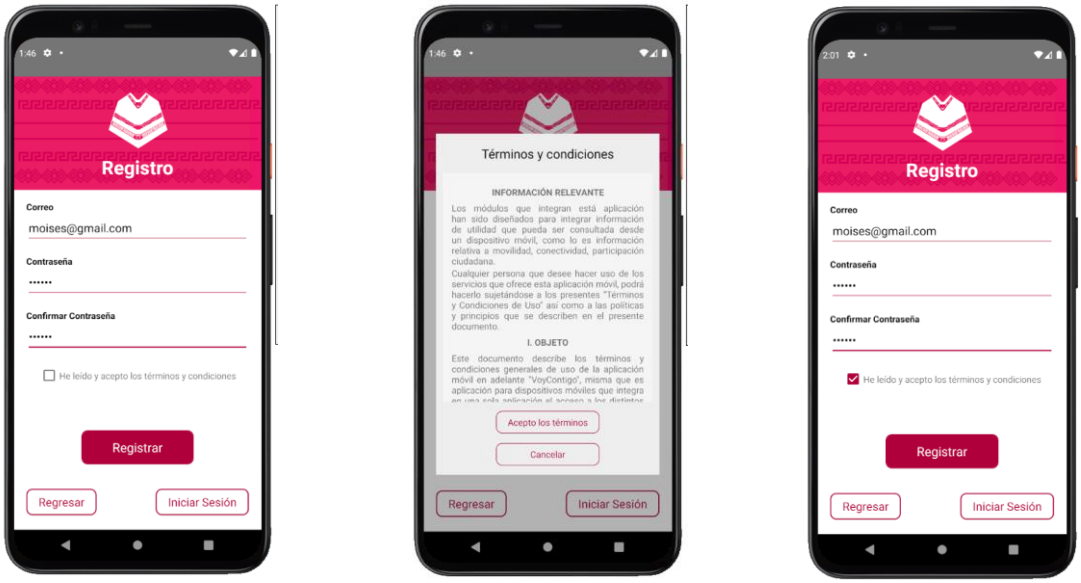


Figura 72 Pantalla de Términos y condiciones

En caso de que no se encuentre la casilla marcada y se intente registrar se mandara un mensaje de error como se muestra en la Figura 73.

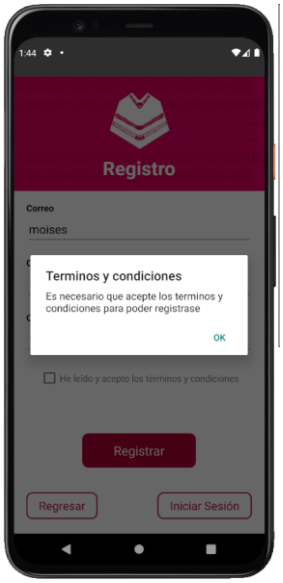


Figura 73 Mensaje de error términos y condiciones

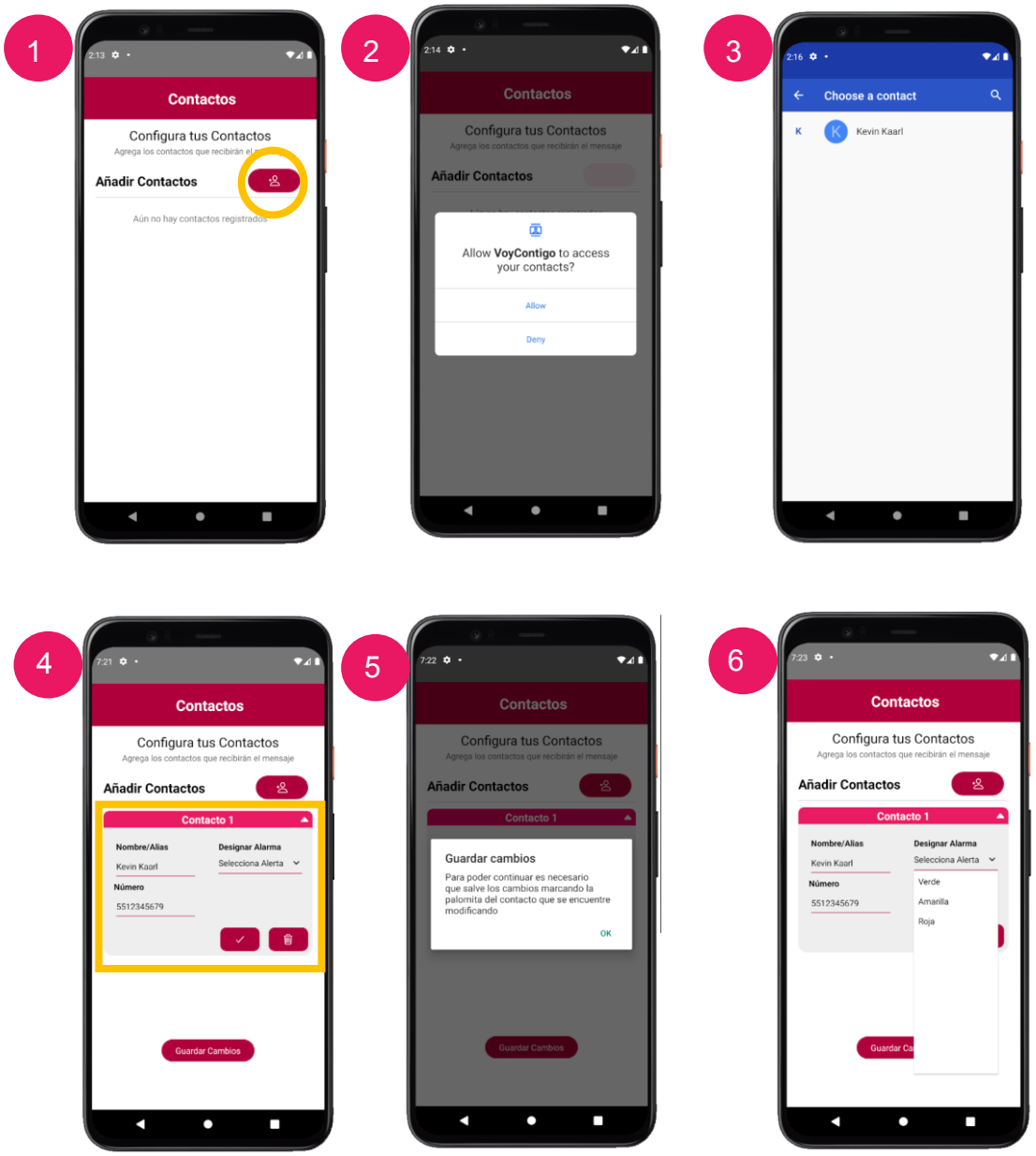


Figura 74 Configuración de Contactos

Una vez realizando el registro, la aplicación navegara hacia la pantalla de *configuración de contactos* como se muestra en la Figura 74 punto 1.

En dicha pantalla al momento de accionar el botón de añadir contactos mostrara un mensaje que indique si da acceso a la lectura de los contactos, como se muestra en el punto 2.

Al momento de dar los permisos se procederá abrir la agenda de contactos como se indica en el punto 3, cuando se seleccione el contacto los datos del mismo se mostrarán en el componente del punto 4, en caso de que el usuario intente salvar los cambios sin antes haber seleccionado una alerta se mostrara un mensaje de error como se muestra en el punto 5, en caso contrario el usuario podrá desplegar un menú con la alerta que quiera asignarle al contacto.

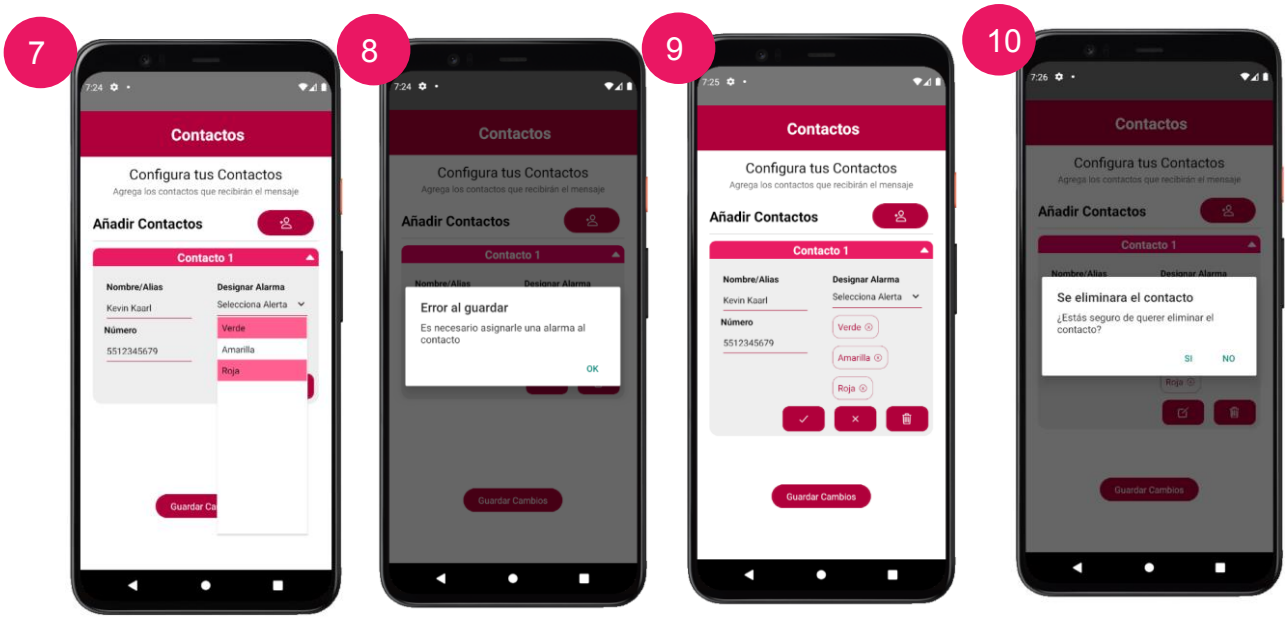


Figura 75 Configuración de Contactos parte 2

El usuario al seleccionar una alerta se marcará con otro color como se muestra en la Figura 75 punto 7, pero en cambio si el usuario presiona el botón del componente de la Figura 74 punto 4 que muestra un icono de una palomita le mandará un error como se muestra en la Figura 75 punto 8 en caso de que no seleccione ninguna alerta.

Cuando el usuario seleccione una o más alertas y salve los cambios se mostrarán las alertas que selecciono como lo muestra el punto 9 de la Figura 75.

Por última acción si el usuario desea eliminar el contacto, al momento de presionar el botón aparecerá un mensaje de confirmación de la acción como se muestra en la Figura 75 punto 10, cuando indique que sí, se eliminará el componente y el registro de dicho contacto asignado.

En caso de que el usuario ya haya configurado sus contactos, las alertas y presione el botón de ***"Guardar cambios",*** el estado de la aplicación cambiara ya que estará autenticado y no será la primera vez que configure sus contactos, permitiendo redireccionar a la pantalla de inicio.

Al momento de cargar la pantalla de inicio los botones aparecerán en un color gris indicando que están deshabilitados, como se muestra en la Figura 76. En primera instancia se encontrarán así porque se están mandando las peticiones para la obtención de los contactos y las alertas del usuario

Figura 76 Cargando información de las alertas y contactos

Cabe mencionar que en caso de que persista el color indicara que aún no hay ningún contacto configurado a alguna de las alertas.

Una vez que el componente compruebe que existen contactos configurados a las a alertas se mostrara la pantalla como la Figura 77.

Figura 77 Pantalla de inicio

Estando en la pantalla de inicio se procederá a activar los botones de riesgo los cuales tendrán que mandar un mensaje de texto a los contactos previamente configurados

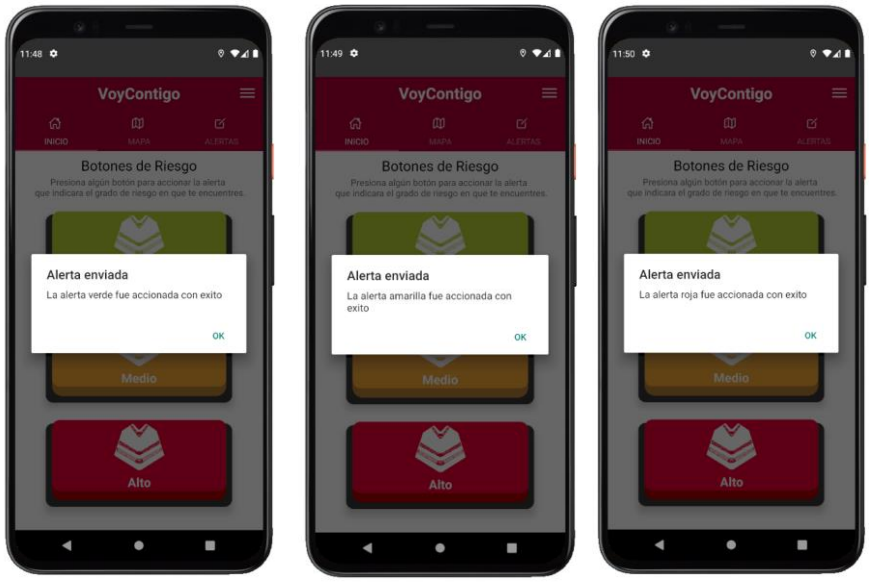


Figura 78 Activación de los botones de riesgo

Una vez presionado el botón de riesgo que se desea activar se mostrara una pantalla emergente que indicara que “la alerta se ha mandado con éxito” como se muestra en la Figura 78, cuando esto suceda, en la aplicación de mensajes de texto se mostrara el historial de mensajes del contacto configurado con el mensaje predeterminado de dicha alerta como se muestra en la Figura 79.

Figura 79 Mensaje de texto de la alerta

En el contenido del mensaje de texto mostrara una leyenda que indique la situación en la que se encuentra el usuario seguido de la ubicación en el instante donde se encuentre ubicado, de tal modo para que al recibir el mensaje el contacto ubique a través de la aplicación de Google Maps donde activo la alerta.

Al momento de activar la alerta también guardara el registro de la ubicación y el tipo de alerta que se emitió para que de esta manera todos los usuarios puedan ver en el mapa virtual donde se han accionado las alertas.

Figura 80 Marcadores de Alertas

Como se muestra en la Figura 80 el usuario podrá observar los marcadores configurados con el icono del logo de la aplicación que corresponden al color de las alertas accionadas. De este modo el usuario puede navegar por el mapa para analizar zonas que puedan o no tener registro de alguna emisión de alerta, identificando los iconos personalizados en los cuales al dar clic podrá obtener la información de cuando sucedió la activación.

En caso de que el usuario no logre ubicarse en el mapa, podrá presionar un botón que lo ubicara en el punto donde se encuentra. Del mismo en caso de que quiera refrescar u actualizar el estado del mapa podrá presionar el botón para visualizar las nuevas alertas en caso de que existan.

Figura 81 Marcado de la ruta del usuario

Así mismo el usuario podrá seguir su traslado como se muestra en la Figura 61, ya que este se marcará en el mapa virtual. Sin embargo, un detalle a marcar es que cuando la aplicación se cierre el uso del GPS se desactivara esto con el objetivo de que ahorre batería y dada la posibilidad de expansión de la funcionalidad el trasado de la ruta del usuario serviría para trasmitir el seguimiento a los contactos que hayan sido configurados.

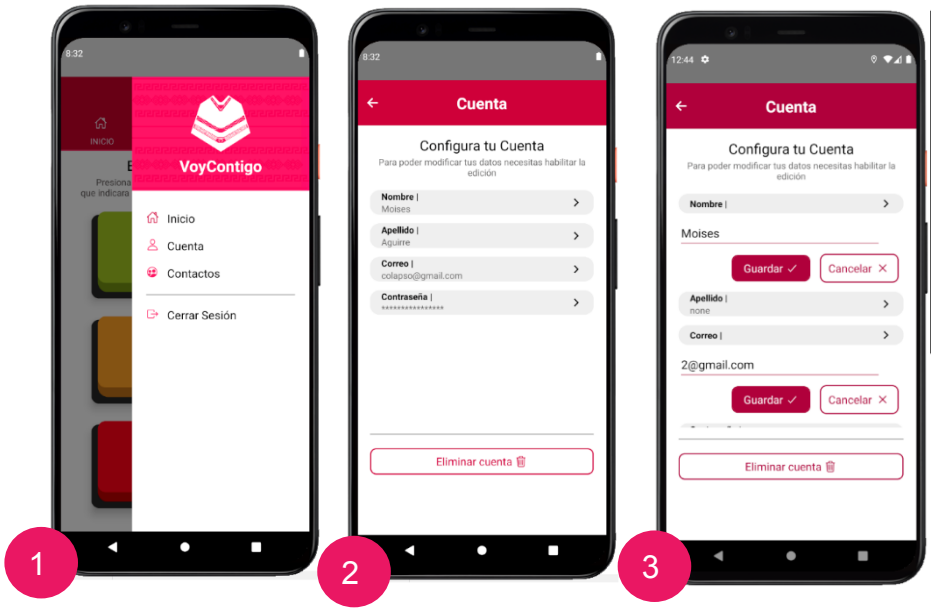


Figura 82 Editar información de contacto

Así mismo a través de la navegación del menú lateral como se muestra en la Figura 82 punto 1, se accede al apartado de la cuenta del usuario, donde se mostrará los datos del registro, como se muestra en el punto 2. Una vez ubicado el usuario en el aparatado de la cuenta en los componentes se cargará la información y al mismo tiempo al dar clic en apartado de interés se podrá modificar como se muestra en el punto 3, y a través de los botones de guardar o cancelar la información se procederá a mandar la solitud para realizar el cambio correspondiente.

En este mismo apartado se muestra un botón que indica la eliminación de la cuenta, al presionarlo eliminara todos los registros vinculados al usuario como son sus contactos y alertas. Una vez eliminado el usuario cambiara el estado de la aplicación y mostrara la pantalla de *Login*.

Como último apartado se tiene el funcionamiento de la visualizaron de la información de las alertas, como se muestra la Figura 83. Se mostrará el contenido respectivo a dicha alerta, en la cual se podrá observar la leyenda del mensaje predefinido por la aplicación al momento de accionar alguna alerta.

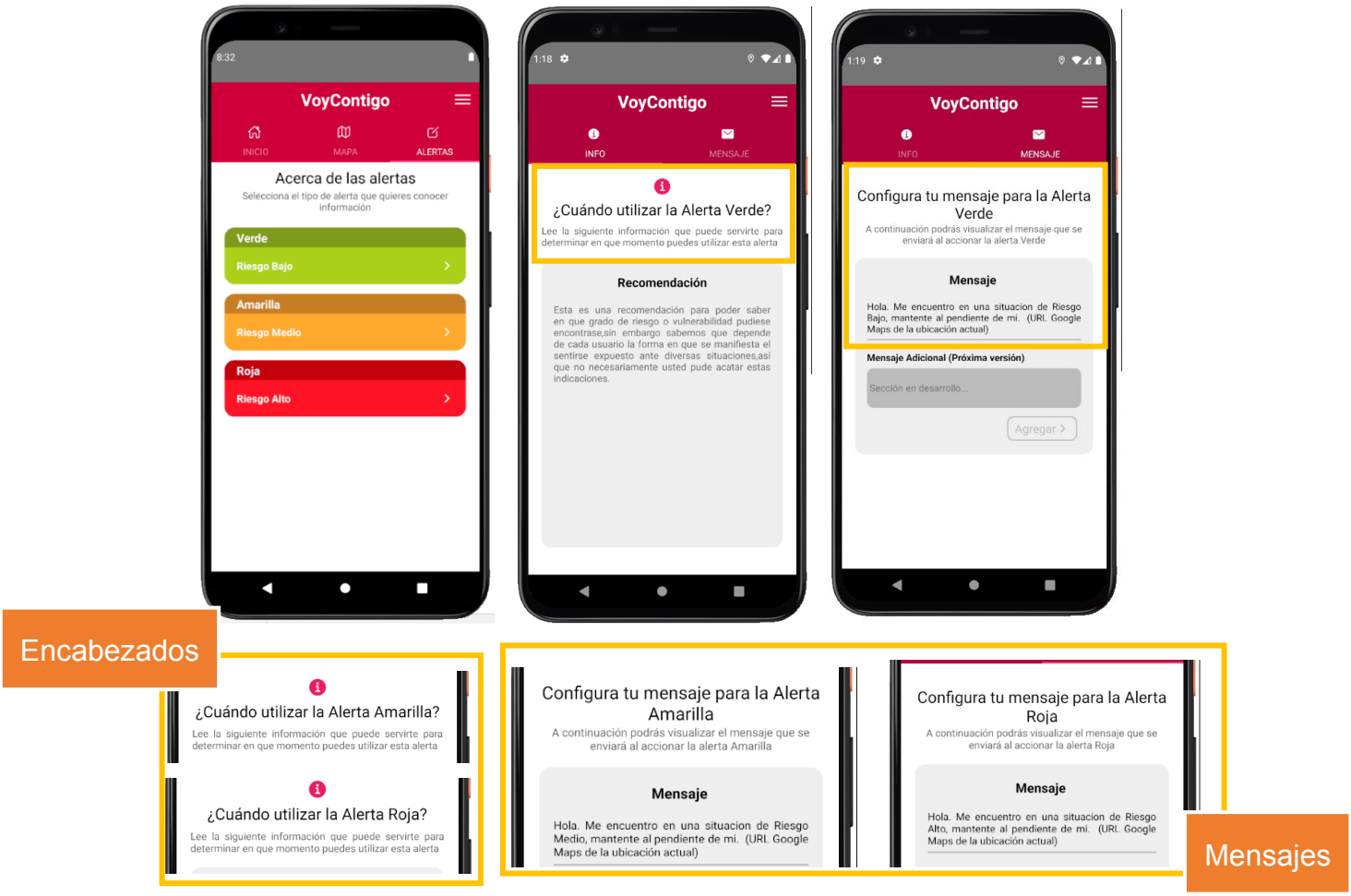


Figura 83 Información de las Alertas

También se mostrara un aparatado inhabilitado que indicará que dado caso de una futura versión se podrá añadir un mensaje personalizado por el usuario haciendo más atractiva la aplicación.

Conclusiones

Para la finalización exitosa del desarrollo de la aplicación móvil se logró emplear diversas tecnologías y conceptos que son piezas fundamentales para la creación del proyecto, las cuales se estructuran a lo largo del mismo, permitiendo identificar el empleo de la integración de diversos componentes y la aplicación de la arquitectura REST, ya que gracias a la arquitectura se puede emplear el modelo cliente-servidor basados en peticiones y respuestas.

El emplear la arquitectura REST permite separar la lógica del negocio con la interfaz grafía permitiendo darle una gran escalabilidad a la aplicación y haciendo esta interacción mucho más ágil y permitiendo realizar buenas prácticas en el desarrollo de sistemas.

El poder crear una API permite el consumo y manipulación de la información por lo que el emplear peticiones HTTP y crear métodos que permitan realizar operaciones CRUD hacen que sean una pieza fundamental para el logro de los objetivos de la aplicación. Ya que en el desarrollo del Back-End se realizaron las respectivas configuraciones de las rutas y los métodos asociados para por hacer la gestión de la creación, modificación, obtención y eliminación de la información de usuarios, alarmas, contactos y ubicaciones.

Así mismo el poder emplear un Framework de desarrollo de aplicaciones nativas para Android y IOS permiten un desarrollo más rápido, sin embargo, existen configuraciones adiciones para cada sistema, el cual se tiene como inconveniente en caso de que no se cuente con algún dispositivo o simulador de alguno de los sistemas Android o IOS, ya que en la realización de pruebas no se sabrá si funciona correctamente la aplicación. Es por eso por lo que al final del proyecto solo se enfocó en el desarrollo para dispositivos Android, ya que no se contaban con las herramientas de pruebas para IOS así que se descartaron algunas configuraciones adicionales para este.

El uso de React-Native con TypeScript permite realizar mejores prácticas ya que en el momento de desarrollo ofrece la asistencia de ayudas que permiten entender el código de mejor manera. Así mismo el uso de React-Native permite integrar códigos reutilizables ya que permite implementar componentes en los cuales se pueden configurar para describir un comportamiento, siendo estos flexibles en la adaptación del contenido.

Gracias a las tecnologías, herramientas y conceptos se logró desarrollar una aplicación móvil que permite gestionar y emitir alertas en tiempo real, que consigue visualizar los lugares de donde se han emitido las alertas un mapa virtual.

Cabe mencionar que al ser una aplicación académica se contemplaron casos pequeños de datos, las cuales durante el desarrollo del proyecto se podrían añadir funcionalidades que contemplen más escenarios, sin embargo, se tuvieron algunas dificultades en realizar un sistema complejo y extenso.

Una de las desventajas a considerar es que al ser un sistema basado en peticiones se necesita un tiempo de respuesta y dicho sistema forzosamente necesita estar conectada internet , ya que a través de este podrá autenticar la información del usuario para dar acceso a la aplicación, como también en la obtención de la información como por ejemplo de las alertas y contactos que se configuraron, pudiese ser un inconveniente para el usuario ya que ante una situación de riesgo el poder notificar a algún contacto pudiese demorar un poco en el tiempo de emisión, ya que el registro de los contactos y alertas están almacenados en el servidor y no de manera local en el teléfono. Así mismo otro escenario a contemplar es el cargar la información de todas las alertas emitas el mapa virtual, ya que podría generar un problema en caso de tener un número muy grande de registros ya que harían un poco más lento la visualización del mapa virtual.

Es importante decir que para una persona no familiarizada en el desarrollo de las tecnologías de React-Native, JavaScript o TypeScript el familiarizarse trae consigo una curva de aprendizaje un poco lenta ya que definen varias características y en un principio se realizarían practicas repetitivas, que en un periodo de tiempo que se adquiera más experiencia y conocimiento se podría mejorar cualquier sistema haciendo de este más compacto. Ya que en la realización de este proyecto existen componentes que pudiesen optimizar el código de mejor manera, sin embargo, por razones de tiempo no se implementaron, tal es así que en base a la investigación y ejecución del proyecto no se descartan las buenas prácticas.

Dicho de esta manera en caso de ampliar o mejorar el proyecto, se considerarán esos escenarios que traen consigo otra complejidad en la hora del desarrollo y permiten ejecutar otro tipo de técnicas para realizar buenas prácticas.

Durante el desarrollo del proyecto se puso a prueba el conocimiento adquirido durante la carrera de ingeniería en computación, donde las bases son parte fundamental para el desarrollo del proyecto, ya que ayudan a adentrase a aprender nuevas tecnólogas y a resolver distintas problemáticas en proyecto extensos con periodos cortos de desarrollo.

Referencias

[1] L. F. Eizmendi Hernández, “Sistema móvil para alertas de consumo de medicamentos y seguimiento de pacientes con diabetes”, Proyecto Terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2018

[2] J. C. Hernández García, “Síntesis procedural aproximada de modelos urbanos 3D a partir de imágenes de mapa virtual de Google maps”, Proyecto Terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2013

[3](2021) SISTER- La APP que te protege (versión 2.4) [En línea] Disponible: https://joinsister.com/es/

[4](2021) En qroo no estoy sola (versión 1.0.5). [En línea]. Disponible: https://qroo.gob.mx/iqm/noestoysola

[5](2021) Mujer segura alerta rosa (versión 2.6.9). [En línea]. Disponible: https://mujerseguraar.com/

[6](2021) Sosmex (botón de pánico) #niunamenos (versión 2.8.6). [En línea]. Disponible: https://sosmex-niunamenos.com/

[7](2021) Vive segura CDMX (versión 1.0.4). [En línea]. Disponible: https://play.google.com/store/apps/details?id=mx.gob.df.inmujeres.app.vivesegura&hl=es_MX&gl=US

[8] React Native [En línea] Disponible: https://reactnative.dev/

[9] MapView [En línea]. Disponible https://github.com/react-native-maps/react-native-maps

[10] Geolocation [En línea]. Disponible https://reactnative.dev/docs/0.63/geolocation

[11] Google Location Services API [En línea]. Disponible https://developer.android.com/training/location

[12] MariaDB [En línea]. Disponible https://mariadb.org/

[13] TypeScript [En línea] Disponible: https://www.typescriptlang.org/

[14] Visual Studio Code [En línea] Disponible: https://code.visualstudio.com/

[15] Android Studio [En línea] Disponible: https://developer.android.com/studio

[16] Transferencia de Estado Representacional [En línea] Disponible: https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional

[17] API [En línea] Disponible: https://www.xataka.com/basics/api-que-sirve

[18] React Native [En línea] Disponible: https://es.wikipedia.org/wiki/React_Native

[19] ¿Qué es JavaScript? [En línea] Disponible: https://velneo.es/que-es-javascript/#:~:text=JavaScript%20es%20un%20lenguaje%20de,en%20la%20interfaz%20de%20usuario.

[20] Acerca de Node.js [En línea] Disponible: https://nodejs.org/es/about/

[21] Express.js [En línea] Disponible: https://expressjs.com/es/

[22] introducción a Android Studio [En línea] Disponible: https://developer.android.com/studio/intro?hl=es-419#:~:text=Android%20Studio%20es%20el%20entorno,est%C3%A1%20basado%20en%20IntelliJ%20IDEA.

[23] Cómo crear y administrar dispositivos virtuales [En línea] Disponible: https://developer.android.com/studio/run/managing-avds?hl=es-419#:~:text=Un%20dispositivo%20virtual%20de%20Android,deseas%20simular%20en%20Android%20Emulator.

[24] NPM [En línea] Disponible: https://www.npmjs.com/

[25] NPX [En línea] Disponible: https://www.npmjs.com/package/npx

[26] yarn [En línea] Disponible: https://yarnpkg.com/

[27] Postman [En línea] Disponible: https://www.postman.com/

[28] Sequelize [En línea] Disponible: https://sequelize.org/

[29] Bcryptjs [En línea] Disponible: https://www.npmjs.com/package/bcryptjs

[30] JWT [En línea] Disponible: https://jwt.io/

[31] @react-navigation/native [En línea] Disponible: https://www.npmjs.com/package/@react-navigation/native

[32] react-native-safe-area-context [En línea] Disponible:

https://www.npmjs.com/package/react-native-safe-area-context

[33] native-screens react [En línea] Disponible: https://github.com/software-mansion/react-native-screens

[34] @react-navigation/stack [En línea] Disponible: https://www.npmjs.com/package/@react-navigation/stack

[35] react-native-gesture-handler

[En línea] Disponible: https://github.com/software-mansion/react-native-gesture-handler

[36] @react-navigation/drawer [En línea] Disponible: https://www.npmjs.com/package/@react-navigation/drawer?activeTab=versions

[37] react-native-reanimated [En línea] Disponible: https://github.com/software-mansion/react-native-reanimated

[38] @react-navigation/material-top-tabs [En línea] Disponible: https://www.npmjs.com/package/@react-navigation/material-top-tabs

[39] react-native-tab-view [En línea] Disponible: https://github.com/satya164/react-native-tab-view

[40] react-native-pager-view [En línea] Disponible: https://github.com/callstack/react-native-pager-view

[41] react-native-vector-icons [En línea] Disponible: https://github.com/oblador/react-native-vector-icons

[42] react-native-permissions [En línea] Disponible: https://github.com/zoontek/react-native-permissions

[43] react-native-responsive-screen [En línea] Disponible: https://github.com/marudy/react-native-responsive-screen

[44] react-native-responsive-fontsize [En línea] Disponible: https://github.com/heyman333/react-native-responsive-fontsize

[45] @react-native-community/checkbox [En línea] Disponible: https://github.com/react-native-checkbox/react-native-checkbox

[46] react-native-maps [En línea] Disponible: https://github.com/react-native-maps/react-native-maps

[47] @react-native-community/geolocation [En línea] Disponible: https://github.com/michalchudziak/react-native-geolocation

[48] react-native-android-location-enabler [En línea] Disponible: https://github.com/Richou/react-native-android-location-enabler

[49] axios [En línea] Disponible: https://axios-http.com/docs/intro

[50] @react-native-async-storage/async-storage [En línea] Disponible: https://github.com/react-native-async-storage/async-storage

[51] react-native-element-dropdown [En línea] Disponible: https://github.com/hoaphantn7604/react-native-element-dropdown

[52] Google maps [En línea] Disponible: https://developers.google.com/maps/documentation/ios-sdk/get-api-key

[53] F. Herrera. React Native: Aplicaciones nativas para IOS y Android [En línea] Disponible: https://www.udemy.com/course/react-native-fh/

[54] F. Herrera. Node: De cero a experto [En línea] Disponible: https://www.udemy.com/course/node-de-cero-a-experto/

Anexos

En esta sección se adicionara el código implementado en el proyecto, el cual sestara dividió entre el Front-End y el Back-End.

Back-End

Carpeta Models

Server.ts

```
import express,{Application} from 'express';
import userRoutes from '../routes/usuario';
import alertaRoutes from '../routes/alerta';
import rolRoutes from '../routes/rol';
import ubicacionRoutes from '../routes/ubicacion';
import conalertaRoutes from '../routes/contactoalerta'
import authRoutes from '../routes/auth';
import contactoRoutes from '../routes/contacto';

import cors from 'cors';
import db from '../db/connection';
class Server {
  private app: Application;
  private port: string ;
  private apiPaths = {
        usuarios: '/api/usuarios',
        alertas:  '/api/alertas',
        ubi:      '/api/ubicacion',
        rol:      '/api/rol',
        authPath: '/api/auth',
        conalerta:'/api/conalerta',
        contactos:'/api/contactos',
  }
    constructor() {
         this.app = express();
         this.port = process.env.PORT || '8000';
         this.dbConnection();
         this.middlewares();
         this.routes();
     }
     async dbConnection (){
          try {
               await db.authenticate();
               console.log('Base de datos online');
          }catch(error:any){
```

```
                throw new Error(error);
            }
        }

        middlewares(){
            this.app.use(cors());
            this.app.use(express.json());
            this.app.use(express.static('public'));
        }

    routes () {
            this.app.use(this.apiPaths.authPath,  authRoutes),
            this.app.use(this.apiPaths.usuarios,  userRoutes),
            this.app.use(this.apiPaths.alertas,   alertaRoutes),
            this.app.use(this.apiPaths.ubi,       ubicacionRoutes),
            this.app.use(this.apiPaths.rol,       rolRoutes),
            this.app.use(this.apiPaths.conalerta, conalertaRoutes),
            this.app.use(this.apiPaths.contactos, contactoRoutes)
    }

    listen(){
        this.app.listen(this.port, ()=>{
            console.log('servidor corriendo en puerto ' + this.port);
        })
    }
}
export default Server;
```

Usuario.ts

```
import { DataTypes, Model } from "sequelize";
import db from "../db/connection";
export interface usuarioModel extends Model{
    idUsuario:string,
    nombre:string,
    apellido:string,
    correo:string,
    password:string,
    telefono:string,

}

const Usuario= db.define<usuarioModel>('Usuario',{
    idUsuario : {
        type:DataTypes.STRING,
        primaryKey:true,
    },
    nombre : {
        type:DataTypes.STRING,
        validate: {
            is: /^[a-z]+$/i,
            max: 12,
            min: 4,
          }
    },
    apellido : {
        type:DataTypes.STRING,
        validate: {
            is: /^[a-z]+$/i,
            max: 12,
            min: 4,
          }

    },
    correo : {
        type:DataTypes.STRING,
        validate: {
            isEmail: true,
          }
    },
    password: {
        type:DataTypes.STRING
    },
```

```
    }
},
    {
    tableName:'usuario',
    createdAt: false,
    updatedAt: false,},
);

export default Usuario;
```

Ubicación.ts

```
import { DataTypes, Model } from "sequelize";
import db from "../db/connection";
import Alerta from './alerta';
import Usuario from "./usuario";

export interface ubicacionModel extends Model{
    idUbicacion:string;
    longitud:number;
    latitud:number;
    fecha:string;
    hora:string;
    Alerta_idAlerta:number;
}

const Ubicacion = db.define<ubicacionModel>('Ubicacion',{
    idUbicacion : {
        type:DataTypes.NUMBER,
        primaryKey:true,
        autoIncrement: true,
    },
    longitud : {
        type:DataTypes.DOUBLE
    },
    latitud : {
        type:DataTypes.DOUBLE
    },
    fecha : {
        type:DataTypes.DATE
    },
    hora : {
        type:DataTypes.TIME
    },
    tiempo : {
        type:DataTypes.NUMBER
    },
    Alerta_idAlerta : {
        type: DataTypes.INTEGER,
        references: {
          model: Alerta,
          key: 'idUsuario'
        }
    },
},
    {
```

```
        tableName:'ubicacion',
        createdAt: false,
        updatedAt: false,


    },
);

export default Ubicacion;
```

Contacto.ts

```
import { DataTypes, Model } from "sequelize";
import db from "../db/connection";
import Usuario from "./usuario";
 export interface ContactoModel extends Model{
    idContacto:string,
    nombre:string,
    telefono:string,
    Usuario_idUsuario:string,

}
const Contacto= db.define<ContactoModel>('Contacto',{
    idContacto : {
        type:DataTypes.NUMBER,
        primaryKey:true,
        autoIncrement: true,
        validate: {
            isNumeric: true,
          }
    },
    nombre : {
        type:DataTypes.STRING,
        validate: {
            max: 15,
            min: 3,
          }
    },
    telefono: {
        type:DataTypes.STRING,
        validate: {
            is: /^[0-9]+$/i,
          }
    },
    Usuario_idUsuario : {
```

```
        type: DataTypes.INTEGER,
        references: {
          model: Usuario,
          key: 'idUsuario'
        }
    },
},
    {

    tableName:'contacto',
    createdAt: false,
    updatedAt: false,},

);

export default Contacto
```

Alerta.ts

```
import { DataTypes, Model } from "sequelize";
import db from "../db/connection";
import Contacto from "./contacto";
import Usuario from "./usuario";

 export interface alertaModel extends Model{
    idAlerta:number;
    tipo:string;
    mensaje:string;
    estado:number;
    Usuario_idUsuario:string;
    Contacto_idContacto:number;

}

const Alerta = db.define<alertaModel>('Alerta',{
    idAlerta: {
        type:DataTypes.NUMBER,
        primaryKey:true,
        autoIncrement: true,
    },
    tipo : {
        type:DataTypes.STRING
    },
    mensaje : {
        type:DataTypes.STRING
```

```
    },
    estado:{
        type:DataTypes.NUMBER
    },
    Usuario_idUsuario : {
        type:DataTypes.STRING,
        references: {
            model: Usuario,
            key: 'idUsuario'
          }
    },
    Contacto_idContacto: {
        type:DataTypes.NUMBER,
        references: {
            model: Contacto,
            key: 'idContacto'
          }
    },
   },
    {
        tableName:'alerta',
        createdAt: false,
        updatedAt: false,
    },
);

export default Alerta;
```

Carpeta Controller

Usuario ts

```
import {Request,Response} from 'express';
import Usuario from '../models/usuario';
import bcrypjs from 'bcryptjs';
import { generarJWT } from '../helper/generarjwt';

export const getUsuarios=  async (req: Request,res:Response) => {

    const usuarios = await Usuario.findAll();

    res.json(usuarios)
}
export const getUsuario= async (req: Request,res:Response) => {
    const {id}= req.params;
    const usuario = await Usuario.findByPk(id);
    if(usuario){
        res.json(usuario);
    }else{
        res.status(404).json({
             msg:`No existe el usuario buscando`
        });
    }
}

export const postUsuario = async( req: Request , res: Response ) => {

    const { body } = req;
    const {password} = body;
    try {
        const totalRegistro = await Usuario.count();
        const salt = bcrypjs.genSaltSync(10);
        body.password=bcrypjs.hashSync(password,salt);
        const fecha= new Date();
        const usuario =Usuario.build(
            {
            idUsuario:
'voy'+totalRegistro+fecha.getDay()+fecha.getHours()+fecha.getMinutes()+fecha
.getSeconds(),
            nombre: body.nombre,
            apellido: body.apellido,
            correo: body.correo,
            password: body.password,
            telefono: body.telefono
```

```
            }
             );

        await usuario.save();
        const token = await generarJWT( usuario.idUsuario );
        res.json({
            usuario,
            token
         })
    } catch (error) {
        console.log(error);
        res.status(500).json({
            msg: 'Erro al Registrar el Usuario. Intentelo nuevamente o
comuniquese con soporte técnico'
        })
    }



}

export const putUsuario=async (req: Request,res:Response) => {
    const {idUsuario} = req.params;
    const { body } = req;
    try {
         const usuario = await Usuario.findByPk(idUsuario);
         if(!usuario) {
              return res.status(404).json({
                  msg:'No existe el usuario'
              })
         }
         if(body.password!== undefined){
            const salt = bcrypjs.genSaltSync(10);
            body.password=bcrypjs.hashSync(body.password,salt);
         }
         await usuario.update(body);
         await usuario.save();
         const token = await generarJWT( usuario.idUsuario );
         res.json({
             usuario,
             token
         });
    } catch (error) {
        console.log(error);
        res.status(500).json({
```

```
            msg: 'Error al intentar Actualizar el Usuario. Intentelo
nuevamente o comuniquese con soporte técnico'
        })
    }

}
export const deleteUsuario=async (req: Request,res:Response) => {
    const {idUsuario}= req.params;

    const usuario = await Usuario.findByPk(idUsuario);
    if(!usuario) {
         return res.status(404).json({
             msg:'No existe el usurario'
         })
    }
    const usuarioAutenticado = (req as any ).usuario ;
    await usuario.destroy();
    console.log(usuarioAutenticado);
    res.json({
        msg:'El Usuario ha sido eliminado con éxito',
        idUsuario,
        usuarioAutenticado
    });
}
```

Ubicación.ts

```
import {Request,Response} from 'express';
import Ubicacion from '../models/ubicacion';
import Alerta from '../models/alerta';

export const getUbicaciones=  async (req: Request,res:Response) => {
    const ubicaciones = await (await Ubicacion.findAll());
    const  alertas =[];
    let j=0;

    for(let i=0 ; i<ubicaciones.length; i++){
        const ale = await Alerta.findByPk(ubicaciones[i].Alerta_idAlerta);
        alertas[j]= {tipo:ale?.tipo,idAlerta:ale?.idAlerta};
        j++;
    }
    res.json({ubicaciones,alertas});
}
```

```
export const postUbicacion = async( req: Request , res: Response ) => {
    const { body } = req;
    try {
        const date= new Date();
        var mm = date.getMonth() + 1;
        var dd = date.getDate();
        var yy = date.getFullYear();
        const dat=yy+'-'+mm+'-'+dd;
        const ubiacion =Ubicacion.build(
                {
                latitud: body.latitud,
                longitud: body.longitud,
                fecha:dat,
                hora:date.toLocaleTimeString(),
                tiempo: body.tiempo,
                Alerta_idAlerta:body.Alerta_idAlerta,

                }
        );
        await ubiacion.save();
        res.json( ubiacion );
    } catch (error) {
        console.log(error);
        res.status(500).json({
            msg: '¡Ups! comuniquese con soporte técnico o inténtelo de
nuevo'
        })
    }


}

export const putUbicacion=async (req: Request,res:Response) => {
    const {id} = req.params;
    const { body } = req;
    try {
         const ubiacion = await Ubicacion.findByPk(id);
         if(!ubiacion) {
              return res.status(404).json({
                  msg:'No existe una ubicación con el id ' +id
              })
         }
       await ubiacion.update(body);
       res.json(ubiacion);
    } catch (error) {
```

```
        console.log(error);
        res.status(500).json({
            msg: 'Hable con Soporte técnico '
        })
    }

}
export const deleteUbicacion=async (req: Request,res:Response) => {
    const {id}= req.params;
    const ubiacion = await Ubicacion.findByPk(id);
    if(!ubiacion) {
         return res.status(404).json({
             msg:'No existe un usurario con el id ' +id
         })
    }
      await ubiacion.destroy();
    res.json({
        msg:'deleteUsuario usuario borrado',
        id
    });
}
```

contactoAlerta.ts

```
import {Request,Response} from 'express';
import Alerta from "../models/alerta";
import Contacto from "../models/contacto";

export const postContactoAlerta = async( req: Request , res: Response ) => {
    const { body } = req;
    try {
        const existeTelefono= await Contacto.findOne({
            where: {
                telefono:body.telefono,
                Usuario_idUsuario:body.Usuario_idUsuario
            }
        });
        if (existeTelefono) {
              return res.status(400).json({ errors: [{
                        msg: `Ya existe un contacto con el telefono :
`+body.telefono
                        }]
                    });
        }
        const contacto =Contacto.build(
```

```
                {
                 nombre: body.nombre,
                 telefono: body.telefono,
                 Usuario_idUsuario:body.Usuario_idUsuario
                }
            );
        await contacto.save();
        const {idContacto, Usuario_idUsuario}=contacto;
        const  tipo= ['roja','amarilla','verde'];
        for(let i=0; i<3 ; i++){
                const existeTipo= await Alerta.findOne({
                    where: {
                        tipo:tipo[i],
                        Contacto_idContacto:idContacto,
                        Usuario_idUsuario
                    }
                });
                if (!existeTipo) {
                    const alerta =Alerta.build(
                          {
                           tipo:tipo[i],
                           Usuario_idUsuario,
                           Contacto_idContacto: idContacto,
                          }
                        );
                    await alerta.save();
                }
        }
          res.json( contacto );

    } catch (error) {

        console.log(error);
        res.status(500).json({errors: [{
            msg: 'Hable con soporte técnico '
            }]
       });
    }

}


export const deleteContactoAlerta=async (req: Request,res:Response) => {
    const { idContacto } = req.params;
    try {
```

```
        const contacto = await Contacto.findByPk(idContacto);
        if(!contacto) {
            return res.status(404).json({errors:[{
                msg:'No existe el contacto'
                }]
            }
            )
        }
        const contactoAutenticado = (req as any ).contacto ;
        await contacto.destroy();
        const alerta= await Alerta.findAll({
            where: {
                Contacto_idContacto:idContacto,
                Usuario_idUsuario:contacto.Usuario_idUsuario,
            }
            });
            if(!alerta) {
                return res.status(404).json({
                    msg:'No existen alertas - hable con soporte técnico '
                })
            }
        res.json({
            msg:'Contacto y Alertas Eliminados',
            contactoAutenticado
        });
    }catch(error){
        console.log(error);
        res.status(500).json({ errors: [{
            msg: 'Hable con soporte técnico '
            }]
           }
        )
    }
}
```

Contacto.ts

```
import {Request,Response} from 'express';
import Contacto from '../models/contacto';

export const getContactos=  async (req: Request,res:Response) => {
    const contactos = await Contacto.findAll();
    res.json(contactos)
}
export const getContacto= async (req: Request,res:Response) => {
```

```
    const {id}= req.params;
    const contacto = await Contacto.findByPk(id);
    if(contacto){
        res.json(contacto);
    }else{
        res.status(404).json({
             msg:`No existe un contacto con el id ${id}`
        });
    }
}
 export const getContactosUsuario= async (req: Request,res:Response) => {
     const {id}= req.params;
     const contacto = await Contacto.findAll({
        where: {
            Usuario_idUsuario:id
        }
    });
     if(contacto){
         res.json(contacto);
     }else{
         res.status(404).json({
              msg:`No hay contactos agregados`
         });
     }
 }
 export const postContacto = async( req: Request , res: Response ) => {
     const { body } = req;
     try {
         const contacto =Contacto.build(
             {
             nombre: body.nombre,
             telefono: body.telefono,
             Usuario_idUsuario:body.Usuario_idUsuario
            }
             );
         await contacto.save();
         res.json( contacto );
     } catch (error) {
         console.log(error);
         res.status(500).json({
             msg: 'Hable con soporte técnico '
         })
     }

 }
```

```
export const putContacto=async (req: Request,res:Response) => {
    const {idContacto} = req.params;
    const { body } = req;
    try {
         const contacto = await Contacto.findByPk(idContacto);
         if(!contacto) {
              return res.status(404).json({errors:[{
                  msg:'No existe un usurario con el id ' +idContacto
              }]
           })
         }
       await contacto.update(body);
       res.json(contacto);
    } catch (error) {
        console.log(error);
        res.status(500).json({
            msg: 'Hable con el administrador'
        })
    }

}
export const deleteContacto=async (req: Request,res:Response) => {
    const {id}= req.params;
    const contacto = await Contacto.findByPk(id);
    if(!contacto) {
         return res.status(404).json({
             msg:'No existe un usurario con el id ' +id
         })
    }
    const contactoAutenticado = (req as any ).contacto ;
      await contacto.destroy();
    console.log(contactoAutenticado);
    res.json({
        msg:'Contacto borrado',
        id,
        contactoAutenticado
    });
}
```

Auth ts

```
import {Request,Response} from 'express';
import Usuario from '../models/usuario';
import bcrypjs from 'bcryptjs';
import { generarJWT } from '../helper/generarjwt';

export const login=async (req: Request,res:Response) => {
    const {correo, password} = req.body;
    try {
        const usuario= await Usuario.findOne({where:{correo}});
        if(!usuario){
            return res.status(400).json({
                msg:'Usuario/ Password no son correctos - correo'
            })
        }
          const validarPassword =
bcrypjs.compareSync(password,usuario.password);
          if ( !validarPassword ) {
            return res.status(400).json({
                msg: 'Usuario / Password no son correctos - password'
            });
        }
        const token = await generarJWT( usuario.idUsuario );
        res.json({
           usuario,
           token
        })
    } catch (error) {
        console.log(error)
        res.json({
            msg:'Hable con soporte técnico.'
        })
    }
}
export const validarTokenUsuario = async (req: Request,res:Response ) => {
    const token = await generarJWT( (req as any).usuario.idUsuario );
    res.json({
        usuario:  (req as any).usuario,
        token,
    })

}
```

Alerta ts

```
import {Request,Response} from 'express';
import Alerta from '../models/alerta';

export const getAlertas=  async (req: Request,res:Response) => {
    const alertas = await Alerta.findAll();
    res.json(alertas)
}
export const getAlerta= async (req: Request,res:Response) => {
    const {id}= req.params;
    const alerta = await Alerta.findByPk(id);

    if(alerta){
        res.json(alerta);
    }else{
        res.status(404).json({errors:[{
             msg:`No existe una alerta con el id ${id}`
             }]
        });
    }
}
export const getAlertaUsuario= async (req: Request,res:Response) => {
    const {idUsuario}= req.params;
    const alerta = await Alerta.findAll({
        where: {
            Usuario_idUsuario:idUsuario
        }
    });
    if(alerta.length>0){
        res.json(alerta);
    }else{
        res.status(404).json({errors:[{
             msg:`Es necesario tener una alerta, borre su contacto y
ingreselo nuevamente. si el error persiste cominiquese con soporte tecnico`
             }]
        });
    }
}
export const getAlertaContacto= async (req: Request,res:Response) => {
    const {idContacto}= req.params;
    const alerta = await Alerta.findAll({
        where: {
            Contacto_idContacto:idContacto
        }
    });
```

```
    if(alerta.length>0){
        res.json(alerta);
    }else{
        res.status(404).json({errors:[{
             msg:`Es necesario tener una alerta, borre su contacto y
ingreselo nuevamente. si el error persiste cominiquese con soporte tecnico`
             }]
        });
    }
}
 export const postAlerta = async( req: Request , res: Response ) => {
     const { body } = req;
     try {
        const existeTipo= await Alerta.findOne({
            where: {
                tipo:body.tipo,
                Contacto_idContacto:body.Contacto_idContacto,
                Usuario_idUsuario:body.Usuario_idUsuario
            }
        });
        if (existeTipo) {
        }else{
         const alerta =Alerta.build(
             {
             tipo: body.tipo,
             mensaje: body.mensaje,
             Usuario_idUsuario: body.Usuario_idUsuario,
             Contacto_idContacto: body.Contacto_idContacto,
            }
             );
         await alerta.save();
         res.json( alerta );
        }
     } catch (error) {
         console.log(error);
         res.status(500).json({ errors: [{
             msg: 'Hable con soporte técnico'
             }]
            }
         )
     }

 }
 export const putAlerta=async (req: Request,res:Response) => {
     const {id} = req.params;
```

```
    const { body } = req;
    try {
         const alerta = await Alerta.findByPk(id);
         if(!alerta) {
              return res.status(404).json({errors:[{
                  msg:'No existe una alerta con el id ' +id
               }
              ]}
              )
         }
        await alerta.update(body);
        res.json(alerta);
    } catch (error) {
         console.log(error);
         res.status(500).json({errors:[{
             msg: 'Hable con soporte técnico'
            }
         ]})
    }
}
export const deleteAlerta=async (req: Request,res:Response) => {
    const {id}= req.params;
    const alerta = await Alerta.findByPk(id);
    if(!alerta) {
         return res.status(404).json({
             msg:'No existe la alerta'
         })
    }
      await alerta.destroy();
    res.json({
        msg:'Alerta Eliminada',
        id
    });
}
```

Carpeta Routes

Usuario.ts

```
import {Router} from 'express';
import { check } from 'express-validator';
import { deleteUsuario, getUsuario, getUsuarios, postUsuario, putUsuario }
from '../controller/usuario';
import {validarCampos,correoExiste, existeIdUsuario} from
'../middlewares/validar_Compos';
```

```
import { validarJWT } from '../middlewares/validarJWT';

const router = Router();

router.get('/'   ,getUsuarios);
router.get('/:idUsuario',[
        check('idUsuario').custom(existeIdUsuario),
        validarCampos,
      ],getUsuario);
router.post('/',
       [
         check('correo','El correo no es valido').isEmail(),
         check('nombre','El nombre es obligatorio').not().isEmpty(),
         check('password','El password debe ser mayor a 6 caracteres
').isLength({min:6}),
         check('correo').custom(correoExiste),
         validarCampos,
       ],postUsuario);
router.put('/:idUsuario',[
         check('idUsuario').custom(existeIdUsuario),
         validarCampos,
       ],putUsuario);
router.delete('/:idUsuario',[
         validarJWT,
         check('idUsuario').custom(existeIdUsuario),
         validarCampos,
       ],deleteUsuario);
export default router;
```

Ubicación.ts

```
import {Router} from 'express';
import { check } from 'express-validator';
import { deleteUbicacion, getUbicaciones, getUbicacion, postUbicacion,
putUbicacion } from '../controller/ubicacion';
import {validarCampos, existeIdUbicacion} from
'../middlewares/validar_Compos';

const router = Router();
router.get('/',getUbicaciones);
router.get('/:id',[
        validarCampos,
      ],getUbicacion);
router.post('/',
         [
```

```
          check('latitud','La latitud es obligatoria').not().isEmpty(),
          check('longitud','La longitud es obligatoria').not().isEmpty(),
          validarCampos,
         ]
        ,postUbicacion);
router.put('/:id',[
            check('idUbicacion').custom(existeIdUbicacion),
            validarCampos,
          ],putUbicacion);
router.delete('/:id',[
          check('idUbicacion').custom(existeIdUbicacion),
          validarCampos,
        ],deleteUbicacion);

export default router;
```

Contactoalerta.ts

```
import {Router} from 'express';
import { check } from 'express-validator';
import { getAlertaContacto } from '../controller/alerta';
import { deleteContactoAlerta, postContactoAlerta } from
'../controller/contactoAlerta';
import { validarJWT } from '../middlewares/validarJWT';
import { validarCampos, existeIdContacto} from
'../middlewares/validar_Compos';

const router = Router();
router.get('/:idContacto',[
        validarCampos,
      ],getAlertaContacto);
router.post('/',
         [
            check('nombre','El nombre es obligatorio').not().isEmpty(),
            check('telefono','El telefono es obligatorio').not().isEmpty(),
            check('telefono','El telefono debe contener minimo 8 numeros
').isLength({min:8}),
            validarCampos,
         ]
        ,postContactoAlerta);

router.delete('/:idContacto',[
        validarJWT,
        check('idContacto').custom(existeIdContacto),
        validarCampos,
```

```
        ],deleteContactoAlerta);

export default router;
```

Contacto.ts

```
import {Router} from 'express';
import { check } from 'express-validator';
import { deleteContacto, getContacto, getContactos, getContactosUsuario,
postContacto, putContacto } from '../controller/contacto';
import {validarCampos, existeIdContacto, telefonoExiste, existeIdUsuarioC}
from '../middlewares/validar_Compos';
import { validarJWT } from '../middlewares/validarJWT';

const router = Router();
router.get('/'    ,getContactos);
router.get('/:id',[
        check('id').custom(existeIdUsuarioC),
        validarCampos,
      ],getContactosUsuario);

router.get('/c/:id',[
        check('id').custom(existeIdContacto),
        validarCampos,
    ],getContacto);
router.post('/',
         [
           check('nombre','El nombre es obligatorio').not().isEmpty(),
           check('telefono','El telefono es obligatorio').not().isEmpty(),
           check('telefono','El telefono debe contener minimo 8 numeros
').isLength({min:8}),
           check('telefono').custom(telefonoExiste),
           validarCampos,
         ]
        ,postContacto);

router.put('/:idContacto',[
            check('idContacto').custom(existeIdContacto),
            validarCampos,
          ],putContacto);
router.delete('/:id',[
          validarJWT,
          check('idContacto').custom(existeIdContacto),
          validarCampos,
        ],deleteContacto);
```

```
export default router;
```

Auth.ts

```
import {Router} from 'express';
import { check } from 'express-validator';
import {validarCampos} from '../middlewares/validar_Compos';
import { login, validarTokenUsuario } from '../controller/auth';
import { validarJWT } from '../middlewares/validarJWT';

const router = Router();
router.post('/login',[
     check('correo','el correo es obligatorio').isEmail(),
     check('password','La contraseña es obligatoria').not().isEmpty(),
     validarCampos
],login);
router.get('/',[
     validarJWT
    ],validarTokenUsuario );

export default router;
```

Alerta.ts

```
import {Router} from 'express';
import { check } from 'express-validator';
import { validarCampos, existeIdUsuario, existeIdAlerta} from
'../middlewares/validar_Compos';
import { getAlertas, postAlerta, putAlerta, deleteAlerta, getAlertaUsuario }
from '../controller/alerta';

const router = Router();
router.get('/'   ,getAlertas);
router.get('/:idUsuario',[
        validarCampos,
      ],getAlertaUsuario);
router.post('/',
         [
           check('idAlerta').custom(existeIdAlerta),
           check('Usuario_idUsuario').custom(existeIdUsuario),
           check('Contacto_idContacto').not().isEmpty(),
           check('tipo','Es necesario definir el tipo de
alerta').not().isEmpty(),
           validarCampos,
```

```
         ]
        ,postAlerta);

router.put('/:id',[
            check('idAlerta').custom(existeIdAlerta),
            validarCampos,
          ],putAlerta);
router.delete('/:id',[
          check('idAlerta').custom(existeIdAlerta),
          validarCampos,
        ],deleteAlerta);
export default router;
```

Carpeta middlewares

validarJWT.ts

```
import {Request,Response} from 'express';
import Usuario from '../models/usuario';
import jwt from 'jsonwebtoken';

interface payloadUser{
    uid:string;
}


export const validarJWT = async( req: Request , res: Response, next:any ) =>
{
    const token = req.header('voy-token');
    if ( !token ) {
        return res.status(401).json({
            msg: 'No hay token en la petición'
        });
    }
    try {
        const payload  = (jwt.verify(token,
process.env.SECRETORPRIVATEKEY+"") as payloadUser);
        const usuario = await Usuario.findByPk(payload.uid);
        if( !usuario ) {
            return res.status(401).json({
                msg: 'Token no válido - usuario no existe '
            })
        }
        (req as any).usuario=usuario;
        next();
```

```
    } catch (error) {
        console.log(error);
        res.status(401).json({
            msg: 'Token no válido'
        })
    }

}
```

validar_Campos.ts

```
import {Request,Response} from 'express';
import { validationResult } from 'express-validator';
import Alerta from '../models/alerta';
import Ubicacion from '../models/ubicacion';
import Usuario from '../models/usuario';
import Contacto from '../models/contacto';


export const validarCampos = (req: Request,res:Response,next:any) => {
    const errors=validationResult(req);
    if(!errors.isEmpty()){
        return res.status(400).json(errors);
    }
    next();
}
export const correoExiste = async(correo:'')=>{
    const existeEmail = await Usuario.findOne({
        where: {
            correo
        }
    });
    if (existeEmail) {
        throw new Error(`Ya existe un usuario con el correo : `+correo);
    }
}

export const existeIdUsuario = async(idUsuario:'')=>{
      const existeId = await Usuario.findByPk(idUsuario);
      if (!existeId) {
        throw new Error(`El id de usuario no existe : `+idUsuario);
    }
```

```
}

export const existeIdAlerta = async(idAlerta:'')=>{
    const existeId = await Alerta.findByPk(idAlerta);
    if (existeId) {
        throw new Error(`El id de usuario no existe : `+idAlerta);
    }


}

export const existeIdUbicacion = async(idUbiacion:'')=>{
    const existeId = await Ubicacion.findByPk(idUbiacion);
    if (existeId) {
        throw new Error(`El id de usuario no existe : `+idUbiacion);
    }
}

export const existeIdRol = async(idRol:'')=>{
    const existeId = await Ubicacion.findByPk(idRol);
    if (existeId) {
        throw new Error(`El id de usuario no existe : `+idRol);
    }
}
export const existeIdContacto = async(idContacto:'')=>{
    const existeId = await Contacto.findByPk(idContacto);
    if (!existeId) {
        throw new Error(`El id del contacto no existe : `+idContacto);
    }
}
export const existeIdUsuarioC = async(Usuario_idUsuario:'')=>{
    const existeId = await Contacto.findOne({
        where: {
            Usuario_idUsuario
        }
    });

    if (!existeId) {
        throw new Error(`No existen contactos del usuario con el id :
`+Usuario_idUsuario);
    }
}

export const telefonoExiste = async(telefono:'')=>{
    const existeTelefono= await Contacto.findOne({
```

```
        where: {
            telefono
        }
    });

    if (existeTelefono) {
        throw new Error(`Ya existe un contacto con el telefono :
`+telefono);

    }
}
```

Carpeta Helper

generarJWT.ts

```
import jwt from 'jsonwebtoken';

export const generarJWT = ( uid = '' ) => {
    return new Promise( (resolve, reject) => {
        const payload = { uid };
        jwt.sign( payload, ""+process.env.SECRETORPRIVATEKEY, {
            expiresIn: '30 days'
        }, ( err, token ) => {
            if ( err ) {
                console.log(err);
                reject( 'No se pudo generar el token' )
            } else {
                resolve( token );
            }
        })

    })
}
```

validarCompos.ts

```
import {Request,Response} from 'express';
import { validationResult } from 'express-validator';
import Alerta from '../models/alerta';
import Ubicacion from '../models/ubicacion';
import Usuario from '../models/usuario';
import Contacto from '../models/contacto';


export const validarCampos = (req: Request,res:Response,next:any) => {
    const errors=validationResult(req);
    if(!errors.isEmpty()){
        return res.status(400).json(errors);
    }
    next();
}
export const correoExiste = async(correo:'')=>{
    const existeEmail = await Usuario.findOne({
        where: {
            correo
        }
    });
    if (existeEmail) {
        throw new Error(`Ya existe un usuario con el correo : `+correo);
    }
}

export const existeIdUsuario = async(idUsuario:'')=>{
    const existeId = await Usuario.findByPk(idUsuario);
    if (!existeId) {
        throw new Error(`El id de usuario no existe : `+idUsuario);
    }
}

export const existeIdAlerta = async(idAlerta:'')=>{
    const existeId = await Alerta.findByPk(idAlerta);
    if (existeId) {
        throw new Error(`El id de usuario no existe : `+idAlerta);
    }


}
export const existeIdUbicacion = async(idUbiacion:'')=>{
    const existeId = await Ubicacion.findByPk(idUbiacion);
    if (existeId) {
```

```
        throw new Error(`El id de usuario no existe : `+idUbiacion);
    }
}
export const existeIdRol = async(idRol:'')=>{
    const existeId = await Ubicacion.findByPk(idRol);
    if (existeId) {
        throw new Error(`El id de usuario no existe : `+idRol);
    }
}
export const existeIdContacto = async(idContacto:'')=>{
    const existeId = await Contacto.findByPk(idContacto);
    if (!existeId) {
        throw new Error(`El id del contacto no existe : `+idContacto);
    }
}
export const existeIdUsuarioC = async(Usuario_idUsuario:'')=>{
    const existeId = await Contacto.findOne({
        where: {
            Usuario_idUsuario
        }
    });
    if (!existeId) {
        throw new Error(`No existen contactos del usuario con el id :
`+Usuario_idUsuario);
    }
}
export const telefonoExiste = async(telefono:'')=>{
    const existeTelefono= await Contacto.findOne({
        where: {
            telefono
        }
    });

    if (existeTelefono) {
        throw new Error(`Ya existe un contacto con el telefono :
`+telefono);
    }
}
```

Carpeta DB

Connection.ts

```
import { Sequelize } from 'sequelize';

const db= new Sequelize('voycontigo','root','toor', {
       host: 'localhost',
       dialect:'mariadb',
});
export default db;
```

.env

```
PORT=8000
SECRETORPRIVATEKEY=V0yC0nt1g0S3cr3tK3y
```

app.ts

```
import dotenv from 'dotenv';
import Server from './models/server';


dotenv.config();
const server = new Server();
server.listen();
```

Front-End

App.js

```
import React from 'react'
import { NavigationContainer } from '@react-navigation/native';

import { StackNavigator } from './src/navigator/StackNavigator';
import { PermissionsProvider } from './src/context/PermissionsContext';
import { AuthProvider } from './src/context/AuthContext';


const AppState = ({children}:any) =>{
  return (
     <PermissionsProvider>
       <AuthProvider>
       {children}
       </AuthProvider>
     </PermissionsProvider>
  )
}

const App = () => {
  return (
    <NavigationContainer>
       <AppState>
       <StackNavigator />
       </AppState>
    </NavigationContainer>

  )
}

export default App;
```

Carpeta Theme

loginTheme.tsx

```
import { StyleSheet } from "react-native";
import { RFValue } from "react-native-responsive-fontsize";
import { widthPercentageToDP as wp, heightPercentageToDP as hp } from
'react-native-responsive-screen';
import { colors } from "./colorsTheme";

export const loginStyles = StyleSheet.create({
```

```
container:{
    flex:1,
    backgroundColor: 'white',
    justifyContent:'center',

},
containerLabel:{

    height: '50%',
    justifyContent:'center',
    paddingHorizontal:'5%'
},
containerButtons:{

    height: '50%',
    alignItems:'center'
},
containerButtonsR:{

     height: '30%',
     alignItems:'center',
     justifyContent:'space-between'
 },
label:{
    marginTop:hp(2.6),
    color:colors.ThemeText,
    fontWeight:'bold',

},
inputFiel:{
    color:colors.ThemeText,
    fontSize:hp(2.6),

},
inputFielIOS:{
     borderBottomColor:colors.primaryDark,
     borderBottomWidth:hp(0.26),
     paddingBottom:hp(0.52),
},

bottonContainer:{
    alignItems:'center',
    marginVertical:hp(5.6),
     paddingBottom:hp(5),
```

```
  },

  btnLight:{

  },

  btnDark:{
      backgroundColor:colors.primaryDark,
  },

  btn : {
      borderColor:colors.primaryDark,
      alignItems:'center',
      justifyContent:'center',
      borderRadius: 10,
      borderWidth:hp(0.26),
  },

  buttom:{
      borderColor:colors.primaryDark,
      alignItems:'center',
      justifyContent:'center',
      borderWidth:hp(0.26),
      paddingHorizontal:hp(6.6),
      paddingVertical:hp(2),
      borderRadius: 10,
      margin:hp(1.04),
  },
  buttonText:{
      color:colors.primaryDark,
      fontSize:hp(2.6),
  },

  btnRegister:{
      backgroundColor:colors.primaryDark,
      alignItems:'center',
      justifyContent:'center',

      paddingVertical:'4%',
      paddingHorizontal:'12%',
      borderRadius: 10,


    },
  textBtnbtnRegister:{
```

```
        fontSize:RFValue(20),
        color:'white',

    },

  btnBack:{
      borderWidth:1,
      borderColor:colors.primaryDark,
      top:'20%',
      alignItems:'center',
      justifyContent:'center',
      paddingVertical:'2%',
      paddingHorizontal:'5%',
      borderRadius: 10,
      marginBottom:15,
  },
  btnSingIn:{
      borderWidth:1,
      borderColor:colors.primaryDark,
      top:'20%',

      alignItems:'center',
      justifyContent:'center',
      paddingVertical:'2%',
      paddingHorizontal:'5%',
      borderRadius: 10,
      marginBottom:15,
  },
  btnTextBack:{
      color:colors.primaryDark,
      fontSize:RFValue(18),
  },
  containerBtnFotterReg:{
      width:'100%',
      flexDirection: "row",
      justifyContent:'space-between',
      paddingHorizontal:'5%',
      marginBottom:'5%',

  },
  containerBtnFootSing:{
      width:'100%',
      flexDirection: 'row',
      alignContent:'flex-end',
      justifyContent:'flex-end',
```

```
        paddingHorizontal:'5%',
        marginBottom:'5%'
    }

});
```

colorsTheme.tsx

```
import React from "react";


export const colors = {

    textDescription:'#686767',
    primary: '#e91e63',
    primaryLight:'#ff6090',
    primaryDark:'#b0003a',
    secondary: '#eeeeee',
    secondaryLight:'#ffffff',
    secondaryDark: '#bcbcbc',
    succes:'#5cb85c',
    succesBorder:'#4A934A',
    danger:'#dc3545',
    warning:'#ffc107',
    gray:'#6c757d',

    light:'#fff',
    dark:'#343a40',
    black:'#000',
    ThemeText:'black',
    ThemeBorder:'#343a40',
    ThemeBackground:'#000',

     aLverde:'#b5cd38',
     aLamarillo:'#f3a63a',
     aLrojo:'#e91c2b',
     aLverdeDark:'#87982c',
     aLamarilloDark:'#b47c2d',
     aLrojoDark:'#a70713',



}
```

appTheme.tsx

```
import { StyleSheet } from "react-native";
import { RFValue } from "react-native-responsive-fontsize";
import {widthPercentageToDP as wp, heightPercentageToDP as hp} from 'react-
native-responsive-screen';
import { colors } from "./colorsTheme";

export const colors2 = {

     primary: '#e91e63',
     primaryLight:'#ff6090',
     primaryDark:'#b0003a',
     primaryMedium:'#e91e63',
     secondary: '#eeeeee',
     secondaryLight:'#ffffff',
     secondaryDark: '#bcbcbc',
     succes:'#5cb85c',
     color3:'',

}

export const styles = StyleSheet.create({
    container:{
        flex:1,
        backgroundColor:'white',
    },
    conteinerTitle:{
        height:'10%',
        alignItems:'center',
        justifyContent:'center',
    },

    header:{
        height:'10%',
        alignItems:'center',
        backgroundColor:colors.primaryDark,
        flexDirection:'row',
        justifyContent:'center',
        paddingVertical:'2%',


    },
    titleHeader:{
        fontWeight: 'bold',
        fontSize:RFValue(25),
```

```
        fontFamily:'Hello Viktoria',
        color:'white'
    },



    tituloSection:{
        fontSize:RFValue(22),
        color:'black',
        textAlign:'center',
        margin:2,
     },

    textDescription:{

         textAlign:'center',
         fontSize:RFValue(14),
         paddingHorizontal:'2%',
         color:colors.textDescription
     },

    titutloTxt:{
        color:'#000',
        fontSize: 30,
        fontWeight:'bold',

    },

    btnLight:{

    },

    btnDark:{
        backgroundColor:colors.primaryDark,
    },

    btn : {
        borderColor:colors.primaryDark,
        alignItems:'center',
        justifyContent:'center',
        borderRadius: 10,
        borderWidth:hp(0.26),
        flexDirection:'row',
    },
    btnDimensionRegister:{
```

```
        paddingHorizontal:'12%',
        marginVertical:hp(.5),
        paddingVertical:'3%',
    },
    btnDimensionRegisterLight:{
        top:'20%',//
        paddingVertical:'2%',
        paddingHorizontal:'5%',
        marginBottom:15,

    },
    backBtn:{
        position:'absolute',
        marginLeft:'3%',
        left:0,
    },

    btnText:{
        color:colors.primaryDark,
        fontSize:hp(2.4),//
    },



});
```

Carpeta Textos

AlertMassage.tsx

```
export var TextAlertVerde    = 'Hola. Me encuentro en una situaci'+'o'+'n de
Riesgo Bajo, mantente al pendiente de mi. ';
export var TextAlertAmarilla = 'Hola. Me encuentro en una situaci'+'o'+'n de
Riesgo Medio, mantente al pendiente de mi. ';
export var TextAlertRoja     = 'Hola. Me encuentro en una situaci'+'o'+'n de
Riesgo Alto, mantente al pendiente de mi. ';
```

Carpeta screens

AboutAlertasScreen.tsx

```
import React, { useState } from 'react'
import { StyleSheet, Text, View, TouchableOpacity, ViewStyle } from 'react-
native';
import { RFValue } from 'react-native-responsive-fontsize';
import { colors } from '../theme/colorsTheme';
import { styles } from '../theme/appTheme';
import Icon from 'react-native-vector-icons/Ionicons';
import { StackScreenProps } from '@react-navigation/stack';

interface Props extends  StackScreenProps <any,any> {};

interface BtnAboutInterface {
    tipo:string;
    navigate: ()=> void;
}

export const AboutAlertasScreen = ({navigation}:Props) => {
    return (
            <View style={[styles.container,{paddingVertical:'3%'}]}>
                <View style={styles.conteinerTitle}>
                        <Text style={styles.tituloSection}>Acerca de las
alertas</Text>
                        <Text style={styles.textDescription}>{'Selecciona el
tipo de alerta que quieres conocer información'}</Text>
                </View>
                 <View style={stylesC.containerSections}>
                      <BtnAboutAlert tipo={'Verde'}    navigate={() =>
navigation.navigate('AlertasTopTab',{tipoAlerta:'Verde'})}/>
                     <BtnAboutAlert tipo={'Amarilla'} navigate={() =>
navigation.navigate('AlertasTopTab',{tipoAlerta:'Amarilla'})}/>
                     <BtnAboutAlert tipo={'Roja'}     navigate={() =>
navigation.navigate('AlertasTopTab',{tipoAlerta:'Roja'})}/>
                 </View>
            </View>


    )
}

const BtnAboutAlert = ({tipo,navigate}:BtnAboutInterface) => {
  let riesgo;
  switch (tipo){
      case 'Verde' :
```

```
          riesgo='Bajo';
       break;
       case 'Amarilla':
        riesgo='Medio';
       break;
       case 'Roja':
        riesgo='Alto';
       break;
  }
    return (
        <TouchableOpacity style={stylesC.btn} onPress={navigate}>
            <View style={stylesC.containerTarger}>
                <View style={[stylesC.containerHeadTarget,
                            {backgroundColor: (tipo==='Verde')?
colors.aLverdeDark : (tipo==='Amarilla')? colors.aLamarilloDark  :
(tipo==='Roja')? colors.aLrojoDark : colors.secondary,}]}>
                 <Text style={stylesC.textHeaderTarget}>{tipo}</Text>
                </View>
                <View style={[stylesC.containerFooterTarget,
                                {backgroundColor: (tipo==='Verde')?
colors.aLverde : (tipo==='Amarilla')? colors.aLamarillo  : (tipo==='Roja')?
colors.aLrojo : colors.secondary}
                                ]}>
                    <Text style={stylesC.textFooterTarget}>Riesgo {riesgo}
</Text>
                    <Icon name='chevron-forward' size={20} color='white'/>
                </View>

            </View>

        </TouchableOpacity>
    )
}

const stylesC = StyleSheet.create({

    containerSections:{
        height:'80%',
        marginVertical:'5%',
        paddingHorizontal:'5%',
    },
    btn:{
        backgroundColor:colors.secondary,
        flexDirection:'row',
        height:'18%',
```

```
        width:'100%',
        marginVertical:'2%',
        borderRadius:15,

    },
    textFooterTarget:{
        fontSize:RFValue(15),
        fontWeight:'bold',
        color:'#FFF'
    },
    textHeaderTarget:{
        fontSize:RFValue(17),
        fontWeight:'bold',
        color:'#FFF'
    },

    containerTarger:{
        borderRadius:15,
        width:'100%',
        flexDirection:'column',
        justifyContent:'space-between',
        alignItems:'center'
    },
    containerHeadTarget:{
        borderTopLeftRadius:15,
        borderTopEndRadius:15,
        flexDirection:'column',
        width:'100%',
        height:'35%',
        paddingHorizontal:'5%',
        paddingVertical:'1%'
    },
    containerFooterTarget:{
        borderBottomLeftRadius:15,
        borderBottomEndRadius:15,
        height:'65%',
        paddingHorizontal:'5%',
        flexDirection:'row',
        width:'100%',
        justifyContent:'space-between',
        alignItems:'center'
    }


});
```

ContactosScreen.tsx

```
import React, { useContext, useEffect, useState } from 'react';
import { Platform, StyleSheet, Text, TextInput, View, TouchableOpacity,
KeyboardAvoidingView, ScrollView, PermissionsAndroid, Alert,FlatList,
SafeAreaView, RefreshControl } from 'react-native';
import { RFValue } from 'react-native-responsive-fontsize';
import  Icon  from 'react-native-vector-icons/Ionicons';
import { colors } from '../theme/colorsTheme';
import { styles } from '../theme/appTheme';
import { widthPercentageToDP as wp, heightPercentageToDP as hp } from
'react-native-responsive-screen';
import { StackScreenProps } from '@react-navigation/stack';
import { AuthContext } from '../context/AuthContext';
import { ContactoAdd } from '../components/Contacto/ContactoAdd';
import { selectContactPhone } from 'react-native-select-contact';
import { Contacto } from '../interfaces/appInterfaces';

interface Props extends  StackScreenProps <any,any> {};
interface routeParam {
    primeraVez:boolean
}


export const ContactoScreen = ({route,navigation}:Props) => {
    const {primeraVez}=route.params  as routeParam;
   const renderContactsItem = (contactItem:Contacto,verificaCampos:boolean,
                               setVerificaCampos:(bol:boolean) => void,
                               habilitaCampoID:string,
                               getContact:() => void
                               ) =>{
       return(
           <>
           <SafeAreaView style={{flex: 1}}>
            <ContactoAdd
getContact={getContact}  habilitaCampo={(habilitaCampoID===contactItem.idCon
tacto)?true:false}
                        verificaCampos={verificaCampos}
                        setVerificaCampos={setVerificaCampos}
idUsuario={idUsuario}
                        idContacto={contactItem.idContacto}
                        noLabel={contactItem.indice}
                        alias={contactItem.nombre}
                        numero={contactItem.telefono}/>
           </SafeAreaView>
           </>
```

```
      )
   }


    const {user,obtenerContactos,
registaContacto,errorMessage,removeError,removeOneSingUp,oneSingup} =
useContext(AuthContext);
    const usuar= JSON.stringify(user,null);
    const usuario= JSON.parse(usuar);
    const idUsuario= usuario.idUsuario;
    const [contactos, setContactos] = useState<Contacto[]>([]);
    const [vacio, setVacio] = useState(false)
    const [verificaCampos, setVerificaCampos] = useState(false);
    const [habilitaCampoID, sethabilitaCampoID] = useState('')


   const getContact =  async() => {
        let m = await   obtenerContactos(idUsuario) as Contacto[];
        for(var i=0 ; i< m.length ; i++){
            m[i].indice=i+1;
        }
        m.reverse();
        (m.length===0)?setVacio(true):setContactos(m);

    }



    const getPhoneNumber = async () => {
        try {
          const granted = await PermissionsAndroid.request(
            PermissionsAndroid.PERMISSIONS.READ_CONTACTS,
            {
                title: 'Contactos',
                message:
                'La apliación desea acceder a tus contactos' +
                'para poder enviar mensajes.',
                buttonNeutral: 'Preguntar despues',
                buttonNegative: 'Cancelar',
                buttonPositive: 'OK',
            },
            );

          if (granted === PermissionsAndroid.RESULTS.GRANTED) {
              if(verificaCampos){
```

```
                  Alert.alert('Agregar contacto','Para poder agregar otro
contacto salve los cambios del contacto que se encuentre modificando');
                }else{
                    return selectContactPhone()
                    .then(async selection => {
                        if (!selection) {
                            return null;
                        }
                        let { contact, selectedPhone } = selection;
                        let snumber= selectedPhone.number.replace(/[^0-9]/g,
'');
                        let idContact=await
registaContacto({idContacto:'',indice:0,nombre:contact.name,
telefono:snumber,Usuario_idUsuario:usuario.idUsuario}) as string;
                        if(idContact ==='error') return;
                        sethabilitaCampoID(idContact);
                        await getContact();
                        setVacio(false);
                        setVerificaCampos(true);
                        return selectedPhone.number;
                    });
                }
          } else {
          Alert.alert('Los permisios para el envio de mensajes ha sido
denegado, activelos para poder mandar las alertas');
          }
        } catch (error:any) {
            console.warn(error);
            Alert.alert(' '+error);
        }
        };
        useEffect( () =>{
                getContact();
                if(errorMessage.length ===0) return;

                Alert.alert('No se puede agregar', errorMessage,
                [{
                    text:'ok',
                    onPress:removeError
                }
                ]
                );

            }
            ,[errorMessage])
```

```
    const [refresh, setRefresh] = useState(false);
    const onRefresh = () =>{
        setRefresh(true);

        setTimeout(async()=>{
            console.log('Refrescando');
            await getContact();
            setRefresh(false);
        },1500);

    };

    return (
        <View style={styles.container}>
            <View style={styles.header}>
                {(!primeraVez)?
                    <TouchableOpacity style={styles.backBtn} onPress={()=>
(verificaCampos)?
                                Alert.alert('Atención','Salve los cambios
del contacto que se encuentre modificando para poder regresar')
                               :
                                (vacio)?
                                   Alert.alert('Atención','Necesita tener
almenos un contacto registrado para poder regresar')
                                   :
                                   navigation.replace('MenuLateral')
                                }>

                    <Icon name='arrow-back-outline' size={RFValue(25)}
color='white'  />
                    </TouchableOpacity>
                :
                    null
                }
                 <Text style={styles.titleHeader}> Contactos</Text>
            </View>


            <View style={styles.conteinerTitle}>
                <Text style={styles.tituloSection}>Configura tus
Contactos</Text>
                <Text style={styles.textDescription}>Agrega los contactos
que recibirán el mensaje</Text>
             </View>
```

```
            <View style={stylesC.containerForm}>
                <KeyboardAvoidingView
                enabled
                style={{ flex: 1 }}
                behavior={ (Platform.OS === 'ios') ? 'padding': 'height' }
                >


                <View style={stylesC.containAdd}>
                    <Text style={stylesC.txtTitle}>Añadir Contactos</Text>
                    <TouchableOpacity
style={[styles.btn,styles.btnDark,stylesC.btnDimension,stylesC.btnAddDimensi
on]}
                                                                activeOpacity={0
.08}
                                                                onPress={()=>[ge
tPhoneNumber()]}
                    >
                    <Icon name='person-add-outline' size={20} color='white'
/>

                    </TouchableOpacity>
                </View>


               { (!vacio)?
               <FlatList  data={contactos} renderItem={({item}) =>
                          renderContactsItem(item,verificaCampos,
setVerificaCampos,
                                             habilitaCampoID,
                                             getContact)}
                          keyExtractor={(item)=> item.idContacto}
                          refreshControl={
                              <RefreshControl
                                refreshing={refresh}
                                onRefresh={onRefresh}
                              />
                          }
                />:
                <View style={{margin:'5%',
alignItems:'center',justifyContent:'center'}}><Text
style={{fontSize:RFValue(15)}}>Aún no hay contactos
registrados</Text></View>
```

```
                }

                </KeyboardAvoidingView>
            </View>
            <View style={{ alignItems:'center', justifyContent:'center',
height:'10%'}}>
           {(primeraVez && !vacio  ) ?
              <TouchableOpacity
style={[styles.btn,styles.btnDark,stylesC.btnDimension,stylesC.btnSaveDimens
ion]}
                onPress={()=> (verificaCampos)?
                                Alert.alert('Guardar cambios','Para poder
continuar es necesario que salve los cambios marcando la palomita del
contacto que se encuentre modificando')
                                :
                                [removeOneSingUp() , console.log('DIME EL
ESTADO EN QUE ESTAS AHORA' ,oneSingup)]
                            }>
                <Text style={{color:'#fff'}}>Guardar Cambios</Text>
              </TouchableOpacity>
               :
              null
           }

            </View>
          </View>
    )
}


const stylesC = StyleSheet.create({

    containerForm:{
        marginVertical:'1%',
        height:'60%',
        paddingHorizontal:'5%',
    },
    btnDimension:{
        paddingHorizontal:hp(3),//20
        paddingVertical:hp(1),//5
        borderRadius: 100,
        margin:hp(1.04),//8
    },
    btnAddDimension:{
        width: '25%',
```

```
    },
    btnSaveDimension:{
        width: '40%'
    },
    containAdd:{
        height:'13%',
        flexDirection:'row',
        justifyContent:'space-between',
        alignItems:'center',
        borderColor:'lightgray',
        borderBottomWidth:1,
        marginBottom:'2%'
    },

    txtTitle:{
        fontWeight: 'bold',
        fontSize:RFValue(22),
        fontFamily:'Hello Viktoria',
        color:'black'
    },



});
```

ContactPeligroScreen.tsx

```
import React, { useContext, useEffect, useState } from 'react';
import { Platform, StyleSheet, Text, TextInput, View, TouchableOpacity,
KeyboardAvoidingView, ScrollView, PermissionsAndroid, Alert,FlatList,
SafeAreaView, RefreshControl } from 'react-native';
import { RFValue } from 'react-native-responsive-fontsize';
import  Icon  from 'react-native-vector-icons/Ionicons';
import { colors } from '../theme/colorsTheme';
import { styles } from '../theme/appTheme';
import { widthPercentageToDP as wp, heightPercentageToDP as hp } from
'react-native-responsive-screen';
import { StackScreenProps } from '@react-navigation/stack';
import { AuthContext } from '../context/AuthContext';
import { ContactoAdd } from '../components/Contacto/ContactoAdd';
import { selectContactPhone } from 'react-native-select-contact';
import { Contacto } from '../interfaces/appInterfaces';

interface Props extends  StackScreenProps <any,any> {};
interface routeParam {
```

```
    primeraVez:boolean
}


export const ContactoScreen = ({route,navigation}:Props) => {
    const {primeraVez}=route.params  as routeParam;
   const renderContactsItem = (contactItem:Contacto,verificaCampos:boolean,
                               setVerificaCampos:(bol:boolean) => void,
                               habilitaCampoID:string,
                               getContact:() => void
                               ) =>{
       return(
           <>
           <SafeAreaView style={{flex: 1}}>
            <ContactoAdd
getContact={getContact}  habilitaCampo={(habilitaCampoID===contactItem.idCon
tacto)?true:false}
                        verificaCampos={verificaCampos}
                        setVerificaCampos={setVerificaCampos}
idUsuario={idUsuario}
                        idContacto={contactItem.idContacto}
                        noLabel={contactItem.indice}
                        alias={contactItem.nombre}
                        numero={contactItem.telefono}/>
           </SafeAreaView>
           </>
       )
   }


    const {user,obtenerContactos,
registaContacto,errorMessage,removeError,removeOneSingUp,oneSingup} =
useContext(AuthContext);
    const usuar= JSON.stringify(user,null);
    const usuario= JSON.parse(usuar);
    const idUsuario= usuario.idUsuario;
    const [contactos, setContactos] = useState<Contacto[]>([]);
    const [vacio, setVacio] = useState(false)
    const [verificaCampos, setVerificaCampos] = useState(false);
    const [habilitaCampoID, sethabilitaCampoID] = useState('')


   const getContact =  async() => {
        let m = await   obtenerContactos(idUsuario) as Contacto[];
        for(var i=0 ; i< m.length ; i++){
```

```
            m[i].indice=i+1;
        }
        m.reverse();
        (m.length===0)?setVacio(true):setContactos(m);

    }



    const getPhoneNumber = async () => {
        try {
          const granted = await PermissionsAndroid.request(
            PermissionsAndroid.PERMISSIONS.READ_CONTACTS,
            {
                title: 'Contactos',
                message:
                'La apliación desea acceder a tus contactos' +
                'para poder enviar mensajes.',
                buttonNeutral: 'Preguntar despues',
                buttonNegative: 'Cancelar',
                buttonPositive: 'OK',
            },
            );

          if (granted === PermissionsAndroid.RESULTS.GRANTED) {
              if(verificaCampos){
                 Alert.alert('Agregar contacto','Para poder agregar otro
contacto salve los cambios del contacto que se encuentre modificando');
                }else{
                    return selectContactPhone()
                    .then(async selection => {
                        if (!selection) {
                            return null;
                        }
                        let { contact, selectedPhone } = selection;
                        let snumber= selectedPhone.number.replace(/[^0-9]/g,
'');
                        let idContact=await
registaContacto({idContacto:'',indice:0,nombre:contact.name,
telefono:snumber,Usuario_idUsuario:usuario.idUsuario}) as string;
                        if(idContact ==='error') return;
                        sethabilitaCampoID(idContact);
                        await getContact();
                        setVacio(false);
                        setVerificaCampos(true);
```

```
                        return selectedPhone.number;
                    });
                }
          } else {
          Alert.alert('Los permisios para el envio de mensajes ha sido
denegado, activelos para poder mandar las alertas');
          }
        } catch (error:any) {
            console.warn(error);
            Alert.alert(' '+error);
        }
        };
        useEffect( () =>{
                getContact();
                if(errorMessage.length ===0) return;

                Alert.alert('No se puede agregar', errorMessage,
                [{
                    text:'ok',
                    onPress:removeError
                }
                ]
                );

            }
            ,[errorMessage])

    const [refresh, setRefresh] = useState(false);
    const onRefresh = () =>{
        setRefresh(true);

        setTimeout(async()=>{
            console.log('Refrescando');
            await getContact();
            setRefresh(false);
        },1500);

    };

    return (
        <View style={styles.container}>
            <View style={styles.header}>
                {(!primeraVez)?
                    <TouchableOpacity style={styles.backBtn} onPress={()=>
(verificaCampos)?
```

```
                                Alert.alert('Atención','Salve los cambios
del contacto que se encuentre modificando para poder regresar')
                               :
                                (vacio)?
                                   Alert.alert('Atención','Necesita tener
almenos un contacto registrado para poder regresar')
                                   :
                                   navigation.replace('MenuLateral')
                                }>

                    <Icon name='arrow-back-outline' size={RFValue(25)}
color='white'  />
                    </TouchableOpacity>
                :
                    null
                }
                 <Text style={styles.titleHeader}> Contactos</Text>
            </View>


            <View style={styles.conteinerTitle}>
                <Text style={styles.tituloSection}>Configura tus
Contactos</Text>
                <Text style={styles.textDescription}>Agrega los contactos
que recibirán el mensaje</Text>
             </View>


            <View style={stylesC.containerForm}>
                <KeyboardAvoidingView
                enabled
                style={{ flex: 1 }}
                behavior={ (Platform.OS === 'ios') ? 'padding': 'height' }
                >


                <View style={stylesC.containAdd}>
                    <Text style={stylesC.txtTitle}>Añadir Contactos</Text>
                    <TouchableOpacity
style={[styles.btn,styles.btnDark,stylesC.btnDimension,stylesC.btnAddDimensi
on]}
                                                                activeOpacity={0
.08}
                                                                onPress={()=>[ge
tPhoneNumber()]}
```

```
                    >
                    <Icon name='person-add-outline' size={20} color='white'
/>

                    </TouchableOpacity>
                </View>


               { (!vacio)?
               <FlatList  data={contactos} renderItem={({item}) =>
                          renderContactsItem(item,verificaCampos,
setVerificaCampos,
                                             habilitaCampoID,
                                             getContact)}
                          keyExtractor={(item)=> item.idContacto}
                          refreshControl={
                              <RefreshControl
                                refreshing={refresh}
                                onRefresh={onRefresh}
                              />
                          }
                />:
                <View style={{margin:'5%',
alignItems:'center',justifyContent:'center'}}><Text
style={{fontSize:RFValue(15)}}>Aún no hay contactos
registrados</Text></View>
                }

                </KeyboardAvoidingView>
            </View>
            <View style={{ alignItems:'center', justifyContent:'center',
height:'10%'}}>
           {(primeraVez && !vacio  ) ?
             <TouchableOpacity
style={[styles.btn,styles.btnDark,stylesC.btnDimension,stylesC.btnSaveDimens
ion]}
                onPress={()=> (verificaCampos)?
                               Alert.alert('Guardar cambios','Para poder
continuar es necesario que salve los cambios marcando la palomita del
contacto que se encuentre modificando')
                               :
                               [removeOneSingUp()]
                         }>
                <Text style={{color:'#fff'}}>Guardar Cambios</Text>
             </TouchableOpacity>
```

```
            :
            null
          }

          </View>
        </View>
    )
}


const stylesC = StyleSheet.create({

    containerForm:{
        marginVertical:'1%',
        height:'60%',
        paddingHorizontal:'5%',
    },
    btnDimension:{
        paddingHorizontal:hp(3),//20
        paddingVertical:hp(1),//5
        borderRadius: 100,
        margin:hp(1.04),//8
    },
    btnAddDimension:{
        width: '25%',
    },
    btnSaveDimension:{
        width: '40%'
    },
    containAdd:{
        height:'13%',
        flexDirection:'row',
        justifyContent:'space-between',
        alignItems:'center',
        borderColor:'lightgray',
        borderBottomWidth:1,
        marginBottom:'2%'
    },

    txtTitle:{
        fontWeight: 'bold',
        fontSize:RFValue(22),
        fontFamily:'Hello Viktoria',
        color:'black'
    },
```

```
});
```

CuentaScreen.tsx

```
import React, { useContext, useEffect, useState } from 'react';
import { Platform, StyleSheet, Text, TextInput, View, TouchableOpacity,
KeyboardAvoidingView, FlatList, RefreshControl, Alert } from 'react-native';
import { RFValue } from 'react-native-responsive-fontsize';
import  Icon  from 'react-native-vector-icons/Ionicons';
import { useForm } from '../hooks/useForm';
import { colors } from '../theme/colorsTheme';
import {styles} from '../theme/appTheme';
import { loginStyles } from '../theme/loginTheme';
import { widthPercentageToDP as wp, heightPercentageToDP as hp } from
'react-native-responsive-screen';
import { StackScreenProps } from '@react-navigation/stack';
import { AuthContext } from '../context/AuthContext';


interface Props extends  StackScreenProps <any,any> {};

interface FormularioComponent {
    tipo:string;

  }

  const forms = [
    {
      id: '1',
      tipo: 'nombre',
    },
    {
      id: '2',
      tipo: 'apellido',
    },
    {
      id: '3',
      tipo: 'correo',
    },
    {
      id: '4',
      tipo: 'password',
```

```
    },
  ];


export const CuentaScreen = ({navigation}:Props) => {
    const [habilitaBtn,setHabilitaBtn]= useState(false);
    const {user,eliminaUsuario} = useContext(AuthContext);
    const usuar= JSON.stringify(user,null);
    const usuario= JSON.parse(usuar);
    const idUsuario= usuario.idUsuario;
    const componentFormulario = (tipo:string) => {
        return (
            <>
            <Formulario tipo={tipo}  />
            </>
        )

    }

    const eliminaUser = async() => {
         await  eliminaUsuario(idUsuario);
    }

    const [refresh, setRefresh] = useState(false);
    const onRefresh = () =>{
        setRefresh(true);

        setTimeout(async()=>{
            console.log('Refrescando');

            setRefresh(false);
        },1500);
    }

    return (
        <View style={styles.container}>
            <View style={styles.header}>
                <TouchableOpacity style={styles.backBtn} onPress={()=>
navigation.replace('MenuLateral')}>
                <Icon name='arrow-back-outline' size={RFValue(25)}
color='white'  />
                </TouchableOpacity>
                <Text style={styles.titleHeader}>Cuenta</Text>
            </View>
```

```
            <View style={[stylesC.containerForm,{backgroundColor:'white'}]}>
                <View style={styles.conteinerTitle}>
                        <Text style={styles.tituloSection}>Configura tu
Cuenta</Text>
                        <Text style={styles.textDescription}>Para poder
modificar tus datos necesitas habilitar la edición</Text>
                </View>


                <View style={{height:'65%',marginVertical:'5%'}}>
                    <KeyboardAvoidingView
                        enabled
                        style={{ flex: 1 }}
                        behavior={ (Platform.OS === 'ios') ? 'padding':
'height' }
                    >
                    <FlatList  data={forms} renderItem={({item}) =>
                            componentFormulario(item.tipo)}
                            keyExtractor={(item)=> item.id}
                            refreshControl={
                                <RefreshControl
                                    refreshing={refresh}
                                    onRefresh={onRefresh}
                                />
                            }
                    />
                    </KeyboardAvoidingView>
                </View>

                <View style={[{backgroundColor:'white',
height:'15%',  alignItems:'center', borderTopWidth:2,
justifyContent:'center' , borderColor:colors.secondaryDark}]}>
                            <TouchableOpacity
style={[styles.btn,stylesC.btnDimension,stylesC.btnWidthDelete,]}
                                            activeOpacity={0.08}
                                            onPress={()=>{eliminaUser()}}

                            >
                                <Text
style={[stylesC.buttonText,{color:colors.primaryDark}]}>Eliminar cuenta
</Text>
                                <Icon name='trash-outline'  size={20}
style={{color:colors.primaryDark}}/>
                            </TouchableOpacity>
```

```
                </View>
            </View>

        </View>
    )
};


const Formulario = ({tipo}:FormularioComponent) =>{
     const [habilitaBtn,setHabilitaBtn]= useState(false);
     const {user,actualizaUsuario} = useContext(AuthContext);
     const usuar= JSON.stringify(user,null);
     const usuario= JSON.parse(usuar);
     const idUsuario= usuario.idUsuario;
     const {nombre,apellido,correo,password,onChange} =useForm ({
         nombre:usuario.nombre,
         apellido:usuario.apellido,
         correo:usuario.correo,
         password:'**********',
     });
    const [temp, setTemp] = useState('')
    let label:any;
    let etiqueta:any;
    let teclado;
    let titulo;
    let pass= false;
    switch (tipo){
         case 'nombre':
             titulo= 'Nombre';
             label= 'nombre';
             etiqueta=nombre;
             teclado='words';
             break;
        case 'apellido':
             titulo= 'Apellido';
             label= 'apellido';
             etiqueta=apellido;
             teclado='words';
             break;
        case 'correo':
            titulo= 'Correo';
            label= 'correo';
            etiqueta=correo;
            teclado='none';
            break;
```

```
        case 'password':
            titulo= 'Contraseña';
            label= 'password';
            etiqueta='****************';
            teclado='words';
            pass=true;
            break;
    };

      const primerEstado = ()=>{
           setTemp(etiqueta);
      };
      const [errorDesc, setErrorDesc] = useState("");
      const [errorGen, setErrorGen] = useState(false);
      const actualiza = async(tipo:string, dato:string) => {
            if(validaForm()===true){
                await actualizaUsuario(idUsuario,tipo,dato);
                setHabilitaBtn(false);
                setTemp(etiqueta);
            }else{
                setErrorGen(true);
            }
       };
       const cancelar = async() => {
             etiqueta= await temp;
             await setHabilitaBtn(false);
       };
        const validaForm = () => {
            const soloLetras= /^[a-z]+$/i;
            setErrorDesc("");
            let valido = true;
            if(nombre.length <4){
                setErrorDesc("Nombre invalido");
                valido=false;
            }else
            if(apellido.length <4) {
                setErrorDesc("Apellido invalido");
                valido=false;
            }else
            if(!soloLetras.test(nombre)){
                setErrorDesc("El Nombre solo debe contener letras");
                valido=false;
            }else
            if(!soloLetras.test(apellido)){
                setErrorDesc("El Apellido solo debe contener letras");
```

```
                valido=false;
            }

            return valido;
        }


        useEffect( () =>{

            if(errorGen===true) {

                    Alert.alert('Error al Guardar', errorDesc,
                    [
                     {
                      text:'ok',
                      onPress:()=>setErrorGen(false)
                     }
                    ]
                );
             }
             primerEstado();

        }
        ,[]);


    return (
        <View style={{flex:1, }}>
            <TouchableOpacity style={stylesC.btn1} disabled={habilitaBtn?
true: false } onPress={() =>(setHabilitaBtn(true))}>
                <View style={{flexDirection:'column', width:'90%'}}>
                        <Text
style={[stylesC.textBtn,{marginVertical:habilitaBtn?'2%':0}]}>{titulo} |
</Text>
                        {habilitaBtn?
                        null
                        :
                        <Text style={{alignItems:'center',
color:colors.textDescription}}>{temp}</Text>
                        }

                </View>

                <Icon name='chevron-forward' size={20} color='black'/>
            </TouchableOpacity>
```

```
          {habilitaBtn?
          <>
                <TextInput
                editable={(!habilitaBtn) ? false : true }
                secureTextEntry={pass===true?true: false}
                autoCorrect={pass===true?false: true}
                placeholderTextColor='rgba(0,0,0,0.4)'
                keyboardType='default'
                underlineColorAndroid={colors.primaryDark}
                style={[stylesC.wInput,
                         stylesC.txtInput,
                        loginStyles.inputFiel,
                        (Platform.OS === 'ios') && stylesC.inputFielIOS,
                        (!habilitaBtn) ? stylesC.colorD : stylesC.colorT]}
                selectionColor={colors.pink[100]}
                autoCapitalize='words'
                onChangeText={(value)=> onChange(value,label)}
                value={etiqueta}

                />
            <View
style={[{backgroundColor:'white',alignItems:'center',justifyContent:'flex-
end', height:'40%', flexDirection:'row'}]}>
                 <TouchableOpacity style={[styles.btn,
styles.btnDark,stylesC.btnDimension, stylesC.btnWidthEdit ]}
                                                  activeOpacity={0.08}
                                                  onPress={()=>actualiza(tipo,
etiqueta)}
                                                  disabled={(!habilitaBtn)?
true: false}

                        >
                        <Text
style={[stylesC.buttonText,{color:'white'}]}>Guardar </Text>
                        <Icon name='checkmark-outline' size={20}
color='white' />
                 </TouchableOpacity>
                 <TouchableOpacity style={[styles.btn,
stylesC.btnDimension,stylesC.btnWidthEdit ]}
                                                 activeOpacity={0.08}
                                                 onPress={()=> cancelar()}
                                                 disabled={(!habilitaBtn)?
true: false}

                       >
```

```
                        <Text
style={[stylesC.buttonText,{color:colors.primaryDark}]}>Cancelar </Text>
                        <Icon name='close-outline' size={25}
color={colors.primaryDark} />
                    </TouchableOpacity>
            </View>
            </>
            :
            null}
        </View>
    );
};


const stylesC = StyleSheet.create({

    btnDimension:{
        paddingHorizontal:hp(3),//20
        paddingVertical:hp(1),//5
        margin:hp(1.04),//8,
    },
    btnWidthEdit:{
        width:'35%',
    },
    btnWidthDelete:{
        width:'100%',
    },
    containIB:{
         flexDirection:'row',
    },

    containerForm:{
        height: '80%',
        paddingHorizontal:'5%',
        marginTop:'5%'

    },

    buttonText:{
        color:colors.primaryDark,
        fontSize:hp(2.4),//20
    },
    txtInput:{
       width:'90%'
    },
```

```
    wInput:{
        width:'60%'
    },
    inputFiel:{

        fontSize:hp(2.6),//20

    },
    inputFielIOS:{
         borderBottomColor:colors.primaryDark,
         borderBottomWidth:hp(0.26),//2
         paddingBottom:hp(0.52),//4
    },
    colorT:{
        color:'black'
    },
    colorD:{
        color:'gray'
    },
    containBtn:{
        justifyContent:'center',
        height:'20%',
        flexDirection:'row',
    },
    btn:{
        height:'15%'
    },
     txtTitle:{
         fontWeight: 'bold',
         fontSize:RFValue(22),
         fontFamily:'Hello Viktoria',
         color:'black'
     },
     textBtn:{
        fontSize:RFValue(14),
        fontWeight:'bold',
        color:'#000'
    },
    btn1:{
        backgroundColor:colors.secondary,
        flexDirection:'row',
        width:'100%',
        paddingHorizontal:'5%',
        marginVertical:'2%',
        justifyContent:'space-between',
```

```
        alignItems:'center',
        borderRadius:15,
    },
});
```

InfoScreen.tsx

```
import React from 'react'
import { StyleSheet, Text, View, ScrollView, TouchableOpacity, TextInput,
Platform, KeyboardAvoidingView } from 'react-native';
import { RFValue } from 'react-native-responsive-fontsize';
import { colors } from '../theme/colorsTheme';
import Icon from 'react-native-vector-icons/Ionicons';
import { StackScreenProps } from '@react-navigation/stack';

interface Props extends  StackScreenProps <any,any> {};
interface TipoAlertaInterface {
    tipoAlerta: string;
}


export const InfoScreen = ({route}: Props) => {
    const {tipoAlerta} = route.params as TipoAlertaInterface;

    let descripcion = '';
    let principal = 'Esta es una recomendación para poder saber en que grado
de riesgo o vulnerabilidad pudiese encontrase,' +
                    'sin embargo sabemos que depende de cada usuario la
forma en que se manifiesta el sentirse expuesto ante diversas situaciones,'+
                    'así  que no necesariamente usted pude acatar estas
indicaciones.';
    switch (tipoAlerta){
         case 'Verde':
             descripcion= '';
             break;
         case 'Amarilla':
             descripcion= '';
             break;
         case 'Roja':
             descripcion= '';
             break;
    }


    return (
```

```
        <View style={styles.container}>

                    <View style={styles.conteinerTitle}>
                    <Icon name='information-circle' color={colors.primary}
size={RFValue(30)} style={{alignSelf:'center'}}/>
                        <Text style={styles.tituloSection}>¿Cuándo utilizar
la Alerta {tipoAlerta}?</Text>
                    </View>
                    <View style={styles.containerSections}>

                        <Text style={styles.textDescription}>Lee la
siguiente información que puede servirte para determinar en que momento
puedes utilizar esta alerta</Text>

                        <View  style={styles.target}>
                                <Text
style={styles.title}>Recomendación</Text>
                                <ScrollView style={styles.scroll}>
                                <Text
style={styles.description}>{descripcion}
                                {principal}
                                </Text>
                                </ScrollView>

                        </View>



                    </View>

        </View>
    )
}


const styles = StyleSheet.create({

    scroll:{

    },

    container:{
        flex:1,

    },
```

```
    conteinerTitle:{
        height:'10%',
        alignItems:'center',
        justifyContent:'center',
        marginTop:'5%'
    },
    tituloSection:{
        fontSize:RFValue(22),
        color:'black',
        textAlign:'center',
        margin:2,

    },
    title:{
        fontSize:RFValue(18),
        fontWeight:'bold',
        color:'black',
        textAlign:'center',
        margin:2,
    },
    containerSections:{
        paddingVertical:'2%',
        height:'90%',
        paddingHorizontal:'5%',
    },
    btn:{
        backgroundColor:colors.succes,
        flexDirection:'row',
        width:'30%',
        paddingHorizontal:'5%',
        marginVertical:'2%',
        justifyContent:'space-between',
        alignItems:'center',
        borderRadius:15,
    },
    textBtn:{
        fontSize:RFValue(14),
        fontWeight:'bold',
        color:'#000',
        marginHorizontal:'5%',
        marginVertical:'10%',
    },
    textDescription:{
        marginBottom:'5%',
        textAlign:'justify',
```

```
        fontSize:RFValue(14),
        color:colors.textDescription
    },
    description:{
        marginVertical:'5%',
        textAlign:'justify',
        fontSize:RFValue(14),
        color:colors.textDescription
    },
    label:{
        color:'#000',
    },
    inputText:{
        color:'#000',
        borderWidth:1,
        borderColor:colors.primaryDark,
        borderRadius:10,
        backgroundColor:'#FFF',
        marginVertical:'2%'
    },
    target:{
        backgroundColor:colors.secondary,
        height:'80%',
        padding:'5%',
        borderRadius:15,
    }

});
```

InicioScreen.tsx

```
import React, { useContext, useEffect, useState } from 'react'
import { Text, View, TouchableOpacity, StyleSheet, Image, Alert, Animated,
TouchableWithoutFeedback } from 'react-native';
import { RFValue } from 'react-native-responsive-fontsize';
import { sendAlerta } from '../components/Alerta/sendAlerta';
import { colors } from '../theme/colorsTheme';
import { styles } from '../theme/appTheme';
import { widthPercentageToDP as wp, heightPercentageToDP as hp } from
'react-native-responsive-screen';
import { useLocation } from '../hooks/useLocation';
import { AuthContext } from '../context/AuthContext';
import { Alerta, Contacto } from '../interfaces/appInterfaces';


interface BtnAlerta {
```

```
    tipo : string,
    onPress: () => void;
    tam:number;
}



export const InicioScreen = () => {

    const {user,
contactos,status,obtenerAlertas,obtenerContacto,registraUbicacionAlerta,erro
rMessage,removeError} = useContext(AuthContext);
    const usuar= JSON.stringify(user,null);
    const usuario= JSON.parse(usuar);
    const idUsuario= usuario.idUsuario;
    const [alertas, setAlertas] = useState<Alerta[]>([]);
    const [alertaVerde, setAlertasVerde] = useState<string[]>([]);
    const [alertaAmarilla, setAlertasAmarrilla] = useState<string[]>([]);
    const [alertaRoja, setAlertasRoja]   = useState<string[]>([]);
    const [idalertaVerde, setIdAlertaVerde] = useState<number>(0);
    const [idalertaAmarilla, setIdAlertaAmarrilla] = useState<number>(0);
    const [idalertaRoja, setIdAlertaRoja]   = useState<number>(0);
    const obtenerAlertasData = async ()=>{
         const a=await obtenerAlertas(idUsuario) as Alerta[];
         setAlertas(a);
         const v: string[]=[];
         const y: string[]=[];
         const r: string[]=[];
        for(let i=0; i<a.length;i++){
            if(a[i].estado===1){
                const s=await obtenerContacto(a[i].Contacto_idContacto) as
Contacto;
                switch(a[i].tipo){
                    case 'verde':
                        v.push(s.telefono);
                        setIdAlertaVerde(a[i].idAlerta);
                        break;
                    case 'amarilla':
                        y.push(s.telefono);
                        setIdAlertaAmarrilla(a[i].idAlerta);
                        break;
                    case 'roja':
                        r.push(s.telefono);
                        setIdAlertaRoja(a[i].idAlerta);
                        break;
```

```
            }
        }
    }
    setAlertasVerde(v);
    setAlertasAmarrilla(y);
    setAlertasRoja(r);

     return a;
}
const{ getCurrentLocation} = useLocation();

 const [lat, setLat] = useState(0);
 const [lon, setLon] = useState(0);

 const obtenerUbicacion = async () => {
    const {latitude,longitude} =await getCurrentLocation();
    setLat(latitude);
    setLon(longitude);
 }


const activaAlerta = async (tipo:string) =>{

    let alerta : string[] =[];
    let tam=0;
    let id: number =0;


    switch(tipo){
        case 'verde':
            {
                alerta=alertaVerde,
                tam=alertaVerde.length,
                id=idalertaVerde
            }
            break;
        case 'amarilla':
           {
               alerta=alertaAmarilla,
               tam=alertaAmarilla.length,
               id=idalertaAmarilla
           }
           break;
        case 'roja':
            {
```

```
                    alerta=alertaRoja,
                    tam=alertaRoja.length,
                    id=idalertaRoja
                }
                break;
        }

        if(tam===0) return Alert.alert('No se puede activar la alerta','Por
favor es necesario que configures un contacto con la alerta, dirígete a la
sección de configuración de contactos');
        obtenerUbicacion();
        for(let i=0;i<tam;i++){
            sendAlerta(alerta[i],tipo,lat,lon);

        }
        registraUbicacionAlerta(lon,lat,id);
    }


     useEffect( () =>{

             obtenerAlertasData();
             obtenerUbicacion();

            if(errorMessage.length ===0) return;

            Alert.alert('No se puede accionar', errorMessage,
                [
                {
                    text:'ok',
                    onPress:removeError
                }
                ]
            );

        }
    ,[errorMessage])


    return (

            <View style={[styles.container,{paddingVertical:'3%'}]}>

                  <View style={styles.conteinerTitle}>
```

```
                        <Text style={styles.tituloSection}>Botones de
Riesgo</Text>
                        <Text style={styles.textDescription}>{'Presiona
algún botón para accionar la alerta \n que indicara el grado de riesgo en
que te encuentres.'}</Text>
                </View>

                <View style={stylesI.containerAlertas }>
                            <BtnAlerta tipo={'verde'}    onPress={() =>
activaAlerta('verde')}    tam={alertaVerde.length} />
                            <BtnAlerta tipo={'amarilla'} onPress={() =>
activaAlerta('amarilla')} tam={alertaAmarilla.length}/>
                            <BtnAlerta tipo={'roja'}     onPress={() =>
activaAlerta('roja')}     tam={alertaRoja.length}/>
                </View>

            </View >
        )
  }

const BtnAlerta = ({tipo,onPress,tam}:BtnAlerta) => {
    const state  = {
        animation: new Animated.Value(0)
    }
    const alturaStyle = {
        marginTop: state.animation.interpolate({
          inputRange: [0, 1],
          outputRange: [-15, 0],
        }),
        paddingBottom: state.animation.interpolate({
          inputRange: [0, 1],
          outputRange: [15, 0],
        }),
      };
      const internoBtn = {
        borderRadius: state.animation.interpolate({
          inputRange: [0, 1],
          outputRange: [12, 16],
        }),
      };
      const bajaBtn = async () => {
          Animated.timing(state.animation, {
                toValue: 1,
                duration: 100,
                useNativeDriver: false
```

```
            }).start();

      };

      const levantaBtn = () => {
        Animated.timing(state.animation, {
            toValue: 0,
            duration: 50,
            useNativeDriver: false
        }).start();
      };



   return (

   <View style={[stylesI.containerBotones,]}>
    <TouchableWithoutFeedback disabled={tam===0 ? true : false}
onPress={onPress} onPressIn={bajaBtn} onPressOut={levantaBtn}>
            <View style={stylesI.btnArea}>
                <View style={[stylesI.btnBase]}>
                <Animated.View style={[stylesI.btnAltura,
                                       {backgroundColor: (tipo=== 'verde') ?
colors.aLverdeDark : (tipo=== 'amarilla')? colors.aLamarilloDark : (tipo===
'roja')? colors.aLrojoDark : 'gray'}  ,
                                        alturaStyle]}>
                    <Animated.View style={[stylesI.btnSuperior,internoBtn ,
                                        {backgroundColor:  tam===0 ?
colors.secondaryDark : (tipo=== 'verde') ? colors.aLverde : (tipo===
'amarilla')? colors.aLamarillo : (tipo=== 'roja')? colors.alerta8 : 'gray'},

                                        ]}>
                    <Image source={require ('../img/logo.png')}
                           style={ stylesI.imgLogo }
                    />
                     <Text style={stylesI.textBtn}> {(tipo=== 'verde')
?'Bajo' : (tipo=== 'amarilla')? 'Medio' : (tipo=== 'roja')? 'Alto' : '' }
</Text>
                    </Animated.View>
                </Animated.View>
                </View>
            </View>
    </TouchableWithoutFeedback>
    </View>
   )
```

```
}

const stylesI = StyleSheet.create({

    containerAlertas:{
        width:'100%',
        height:'90%',
        alignItems:'center',
        padding:'3%',
        marginTop:'5%'
    },
    containerBotones:{
        width:'100%',
        flex:1,
        justifyContent:'space-around',
        flexDirection: 'row',
    },

    textBtn:{
       fontSize:RFValue(20),
       fontWeight:'bold',
       textAlign:'center',
       color:'white',

    },

    containerImg:{
        height:  '100%',
        justifyContent:'center',
        alignItems:'center',
        padding:'5%',
    },
    imgLogo:{
        width: '30%',
        height: '65%',
        marginHorizontal:'5%',
        marginVertical:'2%',
    },

btnArea: {
    height: '80%',
    width: '80%',
},
btnBase: {
```

```
    flex: 1,
    padding: 10,
    backgroundColor: '#363636',
    borderRadius: 12,//12
    shadowColor: "#000",
    shadowOffset: {
        width: 0,
        height: 5,
    },
    shadowOpacity: 0.34,
    shadowRadius: 6.27,

    elevation: 10,
},

btnAltura: {
    borderRadius: 16,//16
  },
btnSuperior: {
    height: "100%",
    alignItems: "center",
    justifyContent: "center",
    flexDirection:'column'
  },
});
```

LoadingScreen.tsx

```
import React from 'react'
import { ActivityIndicator, Text, View } from 'react-native'
import { colors } from '../theme/colorsTheme'
import { LogoHeader } from '../components/Header/LogoHeader';

export const LoadingScreen = () => {

    return (
        <View style={{
             flex:1,
             alignItems:'center',
             justifyContent:'center',
             backgroundColor:colors.primary,
        }}>
             <View style={{height:'50%', width:'100%'}}>
```

```
                    <LogoHeader section='Login'/>
                </View>
                <ActivityIndicator size={80} color='white'/>
            </View>
        )
}
```

LoginScreen.tsx

```
import React, { useContext,useEffect } from 'react'
import { Text, TextInput, TouchableOpacity, View, Platform,
KeyboardAvoidingView, StyleSheet, Keyboard, Alert } from 'react-native';
import { StackScreenProps } from '@react-navigation/stack';
import {styles} from '../theme/appTheme';
import { loginStyles } from '../theme/loginTheme';
import { heightPercentageToDP as hp, widthPercentageToDP as wp } from
'react-native-responsive-screen';
import { LogoHeader } from '../components/Header/LogoHeader';
import { colors } from '../theme/colorsTheme';
import { useForm } from '../hooks/useForm';
import { AuthContext } from '../context/AuthContext';


interface Props extends  StackScreenProps <any,any> {};


export const LoginScreen = ({navigation}:Props) => {

    const {singIn,errorMessage,removeError} = useContext (AuthContext );
    const {email,password, onChange} =useForm ({
        email:'',
        password:'',
    });
   useEffect( () =>{
       if(errorMessage.length ===0) return;

       Alert.alert('Login incorrecto', errorMessage,
         [{
           text:'ok',
           onPress:removeError
          }
         ]
       );
   }
```

```
  ,[errorMessage])


   const onLogin  = () => {
           console.log({email,password});
           Keyboard.dismiss();
           singIn({correo:email, password});
   }


   return (
       <>
         <KeyboardAvoidingView
         enabled
         style={{ flex: 1 }}

         behavior={ (Platform.OS === 'ios') ? 'padding': 'height' }
         >
          <View  style={loginStyles.container}>
               <View style={stylesL.containerImgLogin}>
                        <LogoHeader section='Login'/>
               </View>

               <View style={stylesL.containerInputsButtons}>

                   <View style={loginStyles.containerLabel}>


                          <Text style={loginStyles.label}>Correo</Text>
                          <TextInput
                              placeholder='ejemplo@gmail.com'
                              placeholderTextColor='rgba(0,0,0,0.4)'
                              keyboardType='default'
                              underlineColorAndroid={colors.primaryDark}
                              style={[loginStyles.inputFiel,(Platform.OS
=== 'ios') && loginStyles.inputFielIOS]}
                              selectionColor={colors.pink[100]}
                              autoCapitalize='none'
                              onChangeText={(value)=>
onChange(value,'email')}
                              value={email}

                          />
```

```
                            <Text
style={loginStyles.label}>Contraseña</Text>
                                        <TextInput
                                                secureTextEntry={true}
                                                placeholder='***********'

                                                placeholderTextColor='rgba(0
,0,0,0.4)'

                                                underlineColorAndroid={color
s.primaryDark}
                                                style={[loginStyles.inputFie
l,(Platform.OS === 'ios') && loginStyles.inputFielIOS]}
                                                autoCorrect={false}
                                                onChangeText={(value)=>
onChange(value,'password')}
                                                value={password}
                                                onSubmitEditing={onLogin}
                                        />

                    </View>
                    <View style={loginStyles.containerButtons}>
                             <TouchableOpacity
style={[styles.btn,styles.btnDark,stylesL.btnDimension]}
                                                activeOpacity={0.08}
                                                onPress={() => {onLogin(),
[onLogin]} }
                            >
                                        <Text
style={[loginStyles.buttonText,{color:'white'}]}>Iniciar Sesión</Text>
                            </TouchableOpacity>
                            <TouchableOpacity
style={[styles.btn,stylesL.btnDimension]}
                                                activeOpacity={0.08}
                                                onPress={() =>
navigation.replace('RegistroScreen')}
                            >
                                        <Text
style={loginStyles.buttonText}>Registrarse</Text>
                            </TouchableOpacity>

                    </View>
                </View>
            </View>
        </KeyboardAvoidingView>
```

```
       </>
    )
}

const stylesL = StyleSheet.create({

   btnDimension:{
        paddingHorizontal:hp(6.6),
        paddingVertical:hp(2),
        margin:hp(1.04),
   },

    containerImgLogin:{
        backgroundColor:colors.primary,
        height: hp('35%')
    },
    containerInputsButtons:{
        height:hp('65%')
    },
    imgLogo:{
        width: wp('35%'),//150,
        height: hp('20%'), //150
        borderRadius: 30,
    },

});
```

MapaScreen.tsx

```
import React, { useState } from 'react';
import {  StyleSheet, Text, View } from 'react-native';
import { Map } from '../components/Maps/Map';
import {styles} from '../theme/appTheme';


export const MapaScreen = () => {


    return (
        <View style={[styles.container,{paddingVertical:'3%'}]}>
           <View style={styles.conteinerTitle}>
              <Text style={[styles.tituloSection]}>Contadores de
Alertas</Text>
              <Text style={styles.textDescription}>{'Desplázate sobre el
mapa para ver las marcas de alertas accionadas'}</Text>
```

```
            </View>
           <View style={[stylesM.containMap]}>

                <Map/>
           </View>

        </View>
    )
}

const stylesM = StyleSheet.create({
    containMap:{
        width:'100%',
        height:'88%',
        marginTop:'5%'
    }

});
```

MensajeConfigScreen.tsx

```
import React from 'react'
import { StyleSheet, Text, View, ScrollView, TouchableOpacity, TextInput,
Platform, KeyboardAvoidingView, ViewStyle } from 'react-native';
import { RFValue } from 'react-native-responsive-fontsize';
import { colors } from '../theme/colorsTheme';
import Icon from 'react-native-vector-icons/Ionicons';
import { StackScreenProps } from '@react-navigation/stack';
import { TextAlertVerde,TextAlertAmarilla ,TextAlertRoja} from
'../textos/AlertMessages';
import { styles } from '../theme/appTheme';
import { heightPercentageToDP as hp, widthPercentageToDP as wp } from
'react-native-responsive-screen';
interface Props extends  StackScreenProps <any,any> {};
interface TipoAlertaInterface {
    tipoAlerta: string;
}



export const MensajeConfigScreen = ({route}: Props) => {

    const {tipoAlerta} = route.params as TipoAlertaInterface;
    let descripcion = '';
```

```
    switch (tipoAlerta){
        case 'Verde':
            descripcion= TextAlertVerde;
            break;
        case 'Amarilla':
            descripcion= TextAlertAmarilla;
            break;
        case 'Roja':
            descripcion= TextAlertRoja;
            break;
     }

    return (
        <View style={[styles.container, {marginTop:'8%'}]}>

                    <View style={[styles.conteinerTitle,{margin:'4%'}]}>
                        <Text style={styles.tituloSection}>Configura tu
mensaje para la Alerta {tipoAlerta}</Text>
                        <Text style={styles.textDescription}>A continuación
podrás visualizar el mensaje que se enviará al accionar la alerta
{tipoAlerta}</Text>
                    </View>

                    <View style={stylesM.containerSections}>
                        <View  style={stylesM.target}>
                            <KeyboardAvoidingView
                            enabled
                            style={{ flex: 2 }}
                            behavior={ (Platform.OS === 'ios') ? 'padding':
'height' }
                            >
                                <View style={{height:'40%'}}>
                                    <Text style={stylesM.title}>Mensaje
</Text>
                                    <Text
style={stylesM.description}>{descripcion} {'(URL Google Maps de la ubicación
actual)'}</Text>
                                </View>

                                    <View
style={[stylesM.containerMessage]}>
                                        <Text style={stylesM.label}>Mensaje
Adicional {'(Próxima versión)'}</Text>
                                        <TextInput
                                            editable={false}
```

```
                                            multiline={true}
                                            numberOfLines={2}
                                            placeholder='Sección en
desarrollo...'
                                            placeholderTextColor='rgba(0,0,0
,0.4)'
                                            keyboardType='default'
                                            style={stylesM.inputText}
                                            selectionColor={colors.pink[100]
}
                                            autoCapitalize='none'
                                        />
                                        <View style={{alignItems:'flex-
end'}}>
                                            <TouchableOpacity
style={[styles.btn,stylesM.btnDimension,{borderColor:colors.secondaryDark}]}
disabled >
                                                <Text
style={[stylesM.btnText]}>Agregar</Text>
                                                <Icon name='chevron-forward'
size={20} color={colors.secondaryDark} />
                                            </TouchableOpacity>
                                        </View>
                                    </View>
                                </KeyboardAvoidingView>
                            </View>
                        </View>
        </View>
    )
}


const stylesM = StyleSheet.create({
    title:{
        fontSize:RFValue(18),
        fontWeight:'bold',
        color:'black',
        textAlign:'center',
        margin:2,
    },

    btnDimension:{
        paddingHorizontal:'3%',
        paddingVertical:'1.5%',
        margin:'2%',
```

```
  },

  containerMessage:{
   borderTopWidth:2,
   borderTopColor:colors.secondaryDark,
   paddingVertical:'5%',
   height:'60%'
   },

   containerSections:{
       height:'80%',
       padding:'5%',
   },
   btnText:{
       color:colors.secondaryDark,
       fontSize:hp(2.4),
   },

   textDescription:{
       marginBottom:'5%',
       textAlign:'center',
       fontSize:RFValue(14),
   },
   description:{
       marginVertical:'5%',
       textAlign:'justify',
       fontSize:RFValue(14),
       color:'#000'
   },

   label:{
       color:'#000',
       fontWeight:'bold',
   },
   inputText:{
       color:'#000',
       borderWidth:1,
       borderColor:colors.secondaryDark,
       borderRadius:10,
       backgroundColor:colors.secondaryDark,
       marginVertical:'2%'

   },
   target:{
       backgroundColor:colors.secondary,
```

```
        height:'70%',
        padding:'5%',
        borderRadius:15,
    }

});
```

PermissionsScreen.tsx

```
import React, { useContext } from 'react'
import { Button, Platform, StyleSheet, Text, View, TouchableOpacity } from
'react-native';
import { check, PERMISSIONS, PermissionStatus, request } from 'react-native-
permissions'
import { PermissionsContext } from '../context/PermissionsContext';
import { colors } from '../theme/colorsTheme';
import { BackGround } from '../components/Header/BackGround';
import Icon from 'react-native-vector-icons/Ionicons';
import { RFValue } from 'react-native-responsive-fontsize';

export const PermissionsScreen = () => {

   const {permissions,askLocationPermission} =
useContext(PermissionsContext);
  return (
        <View style={[styles.container,{justifyContent:'center',
alignItems:'center'}]}>
           <View  style={styles.hf}>
               <BackGround section='Register'/>
           </View>
           <View  style={styles.conten}>
                <Icon name='information-circle' color={colors.primary}
size={RFValue(40)} style={{alignSelf:'center'}}/>
                <Text style={styles.description}>
                      Para poder utilizar la aplicación será necesario que
otorgue los permisos para poder acceder a su ubicación.
                </Text>
                <View style={styles.containerBtn}>
                  <TouchableOpacity style={styles.btn}
onPress={askLocationPermission} >
                              <Text style={styles.textBtn}>Permiso</Text>
                  </TouchableOpacity>
```

```
                </View>
            </View>
            <View  style={styles.hf}>
                <BackGround section='Register'/>
            </View>

        </View>
    )
}

const styles = StyleSheet.create({
    container:{
      flex:1,
      backgroundColor:"#FFF",
    },
    hf:{
      height:'20%',
      width:'100%',
      backgroundColor:colors.primary,
    },
    conten:{
       width:'100%',
       height:'60%',
       alingItems:'center',
       justifyContent:'center',
       padding:'10%'
    },
    btn:{
      backgroundColor:colors.primaryDark,
      width:'30%',
      height:'40%',
      alignItems:'center',
      justifyContent:'center',
      borderRadius:8,
    },
    textBtn:{
       color:'#FFF',
       fontSize:RFValue(18)
    },
    containerBtn:{
      alignItems:'center',
      justifyContent:'center',
      height:'30%'
    },
    description:{
```

```
      color:'#000',
      fontSize:RFValue(15),
      marginTop:'10%',
      marginBottom:'2%',
      textAlign:'justify',
    }
});
```

RegistroScreen.tsx

```
import React, {useContext, useEffect, useState} from 'react'
import { StackScreenProps } from '@react-navigation/stack'
import { Text, View, TouchableOpacity, Platform, KeyboardAvoidingView,
StyleSheet, Keyboard, TextInput, Alert, NavigatorIOS, ViewStyle } from
'react-native';
import { LogoHeader } from '../components/Header/LogoHeader';
import { heightPercentageToDP as hp, widthPercentageToDP as wp } from
'react-native-responsive-screen';
import { loginStyles } from '../theme/loginTheme';
import {styles} from '../theme/appTheme';
import { colors } from '../theme/colorsTheme';
import { useForm } from '../hooks/useForm';
import { AuthContext } from '../context/AuthContext';



interface Props extends  StackScreenProps <any,any> {};



export const RegistroScreen = ({navigation}:Props) => {

  const {errorMessage,removeError} = useContext (AuthContext );
  const [errorDesc, setErrorDesc] = useState("");
  const [errorGen, setErrorGen] = useState(false);


    const {nombre,apellido, onChange} =useForm ({
         nombre:'',
         apellido:'',

    });

    const validaForm = () => {
```

```
        setErrorDesc("");
        let valido = true;
        if(nombre.length <4){
            setErrorDesc("Nombre invalido");
            valido=false;
        }
        if(apellido.length <4) {
            setErrorDesc("Apellido invalido");
            valido=false;
        }
        if(nombre.length <4 && apellido.length <4){
            setErrorDesc("Nombre y Apellido invalidos");
            valido=false;
        }
        const soloLetras= /^[a-z]+$/i;
        if(!soloLetras.test(nombre)){
            setErrorDesc("El Nombre solo debe contener letras");
            valido=false;
        }
        if(!soloLetras.test(apellido)){
            setErrorDesc("El Apellido solo debe contener letras");
            valido=false;
        }

        return valido;
    }

    useEffect (() => {
        if(errorGen===false) return;
        Alert.alert('Campos incorrectos',  errorDesc,
        [{
          text:'ok',
          onPress:()=>setErrorGen(false)
         }
        ]
         );
    },[errorGen])


    const onRegister2  = () => {
        Keyboard.dismiss();
        if(validaForm()){
        navigation.navigate('RegistroScreen2',{nombre,apellido});
        }else{
          setErrorGen(true);
```

```
          }

     }

    return (
        <>

          <KeyboardAvoidingView
               enabled
               style={{ flex: 1 }}
               behavior={ (Platform.OS === 'ios') ? 'padding': 'height' }
          >
               <View style={loginStyles.container}>
                    <View style={stylesL.containerImgLogin}>
                    <LogoHeader section='Registro'/>
                    </View>
                    <View style={stylesL.containerInputsButtons}>
                         <View style={loginStyles.containerLabel}>
                             <Text style={loginStyles.label}>Nombre</Text>
                              <TextInput
                                   placeholder='Ejemplo:Juan'
                                   placeholderTextColor='rgba(0,0,0,0.4)'

                                   keyboardType='default'
                                   underlineColorAndroid={colors.primaryDark
}
                                   style={[loginStyles.inputFiel,(Platform.O
S === 'ios') && loginStyles.inputFielIOS]}
                                   selectionColor={colors.pink[100]}
                                   autoCapitalize='words'
                                   onChangeText={(value)=>
onChange(value,'nombre')}
                                   value={nombre}

                              />
                              <Text
style={loginStyles.label}>Apellido</Text>
                              <TextInput
                                   placeholder='Ejemplo:Garcia'
                                   placeholderTextColor='rgba(0,0,0,0.4)'
                                   keyboardType='default'
                                   underlineColorAndroid={colors.primaryDark
}
                                   style={[loginStyles.inputFiel,(Platform.O
S === 'ios') && loginStyles.inputFielIOS]}
```

```
                                selectionColor={colors.pink[100]}
                                autoCapitalize='words'
                                onChangeText={(value)=>
onChange(value,'apellido')}
                                value={apellido}
                            />
                        </View>
                            <View
style={[loginStyles.containerButtonsR,]}>
                            <View
style={[{height:'50%',alignItems:'center', justifyContent:'center',}]}>
                                <TouchableOpacity
style={[styles.btn,styles.btnDark, styles.btnDimensionRegister]}
                                                    onPress={
onRegister2}
                                >
                                    <Text
style={loginStyles.textBtnbtnRegister}>Siguiente</Text>
                                </TouchableOpacity>
                            </View>

                            <View
style={loginStyles.containerBtnFotterReg}>
                                    <TouchableOpacity
style={[styles.btn, styles.btnDimensionRegisterLight]}
                                                onPress={() =>
navigation.replace('LoginScreen')}

                                    >
                                        <Text
style={loginStyles.btnTextBack}>Cancelar</Text>
                                    </TouchableOpacity>
                                    <TouchableOpacity
style={[styles.btn, styles.btnDimensionRegisterLight]}
                                                    onPress={() =>
navigation.replace('LoginScreen')}
                                    >
                                        <Text
style={loginStyles.btnTextBack}>Iniciar Sesión</Text>

                                    </TouchableOpacity>
                                </View>
                            </View>
                    </View>
                </View>
```

```
            </KeyboardAvoidingView>
        </>
    )
}


const stylesL = StyleSheet.create({
     containerImgLogin:{
         backgroundColor:colors.primary,
         height: hp('25%'), //35
     },
     containerInputsButtons:{
         height:hp('75%'),//65
         justifyContent:'space-between'
     },


 });
```

RegistroScreen2.tsx

```
import React, { useContext, useEffect, useState } from 'react'
import { Alert, Keyboard, KeyboardAvoidingView, Platform, StyleSheet, Text,
TextInput, TouchableOpacity, View } from 'react-native';
import { StackScreenProps } from '@react-navigation/stack';
import { loginStyles } from '../theme/loginTheme';
import {styles} from '../theme/appTheme';
import { LogoHeader } from '../components/Header/LogoHeader';
import { colors } from '../theme/colorsTheme';
import { heightPercentageToDP as hp, widthPercentageToDP as wp } from
'react-native-responsive-screen';
import { Terminos } from '../components/Terminos/Terminos';
import { useForm } from '../hooks/useForm';
import { AuthContext } from '../context/AuthContext';



interface Props extends  StackScreenProps <any,any> {};
interface routeParam {
     nombre:string,
     apellido:string,
}

export const RegistroScreen2 = ({navigation,route}:Props) => {
```

```
     const {singUp,primeraVez,errorMessage,removeError} =
useContext(AuthContext);
     const {nombre,apellido}=route.params  as routeParam;
     const [errorDesc, setErrorDesc] = useState("");
     const [errorGen, setErrorGen] = useState(false);
     const {correo,password,confirmpassword, onChange} =useForm ({
      correo:'',
      password:'',
      confirmpassword:'',
      });

     const validaForm = () => {
          setErrorDesc("");
          let valido = true;
          if(confirmpassword!== password){
               setErrorDesc("Las contraseñas no coinciden");
               valido = false;
          }

          return valido;
     }

     useEffect( () =>{

          if(errorGen) {
               if(!confirma){
                    Alert.alert('Términos y condiciones', 'Es necesario que
acepte los terminos y condiciones para poder registrase',
                    [
                     {
                      text:'ok',
                      onPress:()=>setErrorGen(false)
                     }
                    ]
               );
               }else{
               Alert.alert('Campos incorrectos', errorDesc,
                    [
                     {
                      text:'ok',
                      onPress:()=>setErrorGen(false)
                     }
                    ]
               );
```

```
            }
        }

        if(errorMessage.length ===0 ) return;

        Alert.alert('Campos incorrectos', errorMessage ,
            [
             {
              text:'ok',
              onPress:removeError
             }
            ]
        );
    }
    ,[errorMessage,errorGen])



    const onRegister  = () => {
        Keyboard.dismiss();
        if(validaForm() && confirma){
            singUp({
                nombre,
                apellido,
                correo,
                password
            });
            primeraVez();

         }else {
            setErrorGen(true);
         }


    }

     const [confirma,setConfirma]=useState(false);
    return (

    <>

    <KeyboardAvoidingView
        enabled
        style={{ flex: 1 }}
        behavior={ (Platform.OS === 'ios') ? 'padding': 'height' }
```

```
    >
        <View style={loginStyles.container}>
            <View style={stylesL.containerImgLogin}>
            <LogoHeader section='Registro'/>
            </View>
            <View style={stylesL.containerInputsButtons}>
                <View
style={[loginStyles.containerLabel,{paddingTop:'10%'}]}>
                        <Text style={loginStyles.label}>Correo</Text>
                        <TextInput
                            placeholder='ejemplo@gmail.com'
                            placeholderTextColor='rgba(0,0,0,0.4)'
                            keyboardType='default'
                            underlineColorAndroid={colors.primaryDark}
                            style={[loginStyles.inputFiel,(Platform.OS
=== 'ios') && loginStyles.inputFielIOS]}
                            selectionColor={colors.pink[100]}
                            autoCapitalize='none'
                            onChangeText={(value)=>
onChange(value,'correo')}
                            value={correo}
                        />
                        <Text
style={loginStyles.label}>Contraseña</Text>
                        <TextInput
                                secureTextEntry={true}
                                placeholder='************'

                                placeholderTextColor='rgba(0,0,0,0.4)'

                                underlineColorAndroid={colors.primaryDa
rk}
                                style={[loginStyles.inputFiel,(Platform
.OS === 'ios') && loginStyles.inputFielIOS]}
                                autoCorrect={false}
                                onChangeText={(value)=>
onChange(value,'password')}
                                value={password}
                        />
                        <Text style={loginStyles.label}>Confirmar
Contraseña</Text>
                        <TextInput
                                secureTextEntry={true}
                                placeholder='************'
```

```
                                    placeholderTextColor='rgba(0,0,0,0.4)'

                                    underlineColorAndroid={colors.primaryDa
rk}
                                    style={[loginStyles.inputFiel,(Platform
.OS === 'ios') && loginStyles.inputFielIOS]}
                                    autoCorrect={false}
                                    onChangeText={(value)=>
onChange(value,'confirmpassword')}
                                    value={confirmpassword}
                                    onSubmitEditing={onRegister}
                                />
                                <Terminos confirma={confirma}
setConfirma={setConfirma}/>
                            </View>
                        <View style={loginStyles.containerButtonsR}>
                                <TouchableOpacity
style={[styles.btn,styles.btnDark, styles.btnDimensionRegister]}
                                                onPress={onRegister}
                                >
                                    <Text
style={loginStyles.textBtnbtnRegister}>Registrar</Text>
                                </TouchableOpacity>
                                <View
style={loginStyles.containerBtnFotterReg}>
                                <TouchableOpacity style={[styles.btn,
styles.btnDimensionRegisterLight]}
                                                onPress={() =>
navigation.navigate('RegistroScreen')}

                                >
                                    <Text
style={loginStyles.btnTextBack}>Regresar</Text>
                                </TouchableOpacity>
                                <TouchableOpacity style={[styles.btn,
styles.btnDimensionRegisterLight]}
                                                onPress={() =>[
navigation.replace('LoginScreen')]}
                                >
                                    <Text
style={loginStyles.btnTextBack}>Iniciar Sesión</Text>
                                </TouchableOpacity>
                                </View>
                        </View>
                </View>
```

```
          </View>
     </KeyboardAvoidingView>
     </>
     )
}



const stylesL = StyleSheet.create({
     containerImgLogin:{
     backgroundColor:colors.primary,
     height: hp('25%'), //35
     },
     containerInputsButtons:{
     height:hp('75%'),//65
     justifyContent:'space-between'
     },


});
```

Carpeta Navigator

AlertasTopTab.tsx

```
import React, {useEffect, useState} from 'react'
import { createMaterialTopTabNavigator } from '@react-navigation/material-
top-tabs';
import { useSafeAreaInsets } from 'react-native-safe-area-context';
import Icon from 'react-native-vector-icons/Ionicons';
import { colors } from '../../theme/colorsTheme';
import { StyleSheet, Text, TouchableOpacity, View } from 'react-native';
import { DrawerScreenProps } from '@react-navigation/drawer';
import { RFValue } from 'react-native-responsive-fontsize';
import { ContactPeligroScreen } from '../../screens/ContactPeligroScreen';
import { MensajeConfigScreen } from '../../screens/MensajeConfigScreen';
import { InfoScreen } from '../../screens/InfoScreen';

const Tab = createMaterialTopTabNavigator();
interface Props extends DrawerScreenProps  <any,any> {};
interface routeParam {
     tipoAlerta:string,
}


export const AlertasTopTab = ({navigation,route}:Props) => {
    const {top:paddingTop} = useSafeAreaInsets();
    const {tipoAlerta}= route.params as routeParam;

    const [tipo, setTipo] = useState(tipoAlerta)


    useEffect(() => {
        setTipo(tipoAlerta);
        navigation.setOptions({
            headerRight: () =>(
             <View style={{backgroundColor:'white'}}>
              <TouchableOpacity style={{
                      marginLeft:10
                      }}
                  onPress={() => navigation.toggleDrawer()} >

                  <Icon name='menu-outline' size={35}
color={colors.primary}/>

              </TouchableOpacity>
              </View>
```

```
            )} )

    }, [])

    return (
        <>
        <View style={styles.headerContainer}>
            <Text style={styles.titleHeader}>VoyContigo</Text>
             <TouchableOpacity style={styles.hamburgerBtn}
                  onPress={() => navigation.toggleDrawer()} >
                  <Icon name='menu-outline' size={33} color='white'/>
              </TouchableOpacity>
        </View>

        <Tab.Navigator  style={{paddingTop,
backgroundColor:colors.primaryDark}}
                        sceneContainerStyle={{backgroundColor:'white'}}

                        screenOptions={({route}) => ({
                                tabBarPressColor: colors.primaryLight,
                                tabBarShowIcon: true,
                                tabBarIndicatorStyle: {
                                    backgroundColor: colors.primaryLight,

                                },

                                tabBarActiveTintColor:'white',
                                tabBarStyle: {
                                borderTopColor: colors.primary,
                                borderTopWidth: 0,
                                elevation: 0,
                                backgroundColor:colors.primaryDark,
                                height:'10%',
                                },

                                style: {
                                    backgroundColor:'#000',
                                },
                                tabBarIcon:({color,focused}) => {

                                        let iconName:string='';
                                        switch (route.name) {
                                            case 'Mensaje':
                                                iconName='mail';
```

```
                                                break;
                                            case 'Info':
                                                iconName='information-
circle';
                                                break;
                                            default:
                                                break;
                                        }

                                        return (
                                                <Text
style={{color:'white'}}>
                                                <Icon name={iconName}
size={20} color='white' />
                                                </Text>
                                                );
                                    }
                            })
                           }
        >

        <Tab.Screen
name="Info"    component={InfoScreen}          initialParams={{tipoAlerta:t
ipo}} />
        <Tab.Screen name="Mensaje"
component={MensajeConfigScreen}  initialParams={{tipoAlerta:tipo}}  />
        </Tab.Navigator>

</>

    )
}
const styles = StyleSheet.create({
    headerContainer:{
        backgroundColor:colors.primaryDark,
        height:'7%',
        flexDirection:'row',
        alignItems:'center',
        justifyContent:'center'
    },
    titleHeader:{
        fontWeight: 'bold',
        fontSize:RFValue(25),
        fontFamily:'Hello Viktoria',
         color:'white'
```

```
    },
    hamburgerBtn:{
        position:'absolute',
        marginLeft:'2%',
        right:0,
    }


});
```

MenuLateral.tsx

```
import React, { useContext } from 'react';
import { createDrawerNavigator, DrawerContentComponentProps,
DrawerContentScrollView } from '@react-navigation/drawer';
import { Image, StyleSheet, Text, TouchableOpacity, useWindowDimensions,
View } from 'react-native';
import { colors } from '../theme/colorsTheme';
import { styles } from '../theme/appTheme';
import { TopTabNavigator } from './TopTabNavigator';
import { RFValue } from 'react-native-responsive-fontsize';
import { CuentaScreen } from '../screens/CuentaScreen';
import { BackGround } from '../components/Header/BackGround';
import { AlertasTopTab } from './aboutAlertas/AlertasTopTab';
import { TopTabConfig } from '../components/Navigation/TopTabConfig';
import { AuthContext } from '../context/AuthContext';
import { ContactoScreen } from '../screens/ContactosScreen';
import { LoadingScreen } from '../screens/LoadingScreen';
import Icon from 'react-native-vector-icons/Ionicons';


const Drawer = createDrawerNavigator();


export const MenuLateral = () => {
    const {width} =useWindowDimensions();
    const {status} = useContext(AuthContext);
    if(status!== 'authenticado') return <LoadingScreen/>;

    return (
        <Drawer.Navigator

                            screenOptions={{
                                drawerType: width >= 768 ? 'permanent' :
'front',
```

```
                                drawerPosition:'right',

                                headerShown:
false,
                                headerStyle: {
                                  backgroundColor:
colors.primaryDark,
                                },
                                headerTintColor:
'#fff',
                                headerTitleStyle: {
                                  fontWeight: 'bold',
                                },
                                headerTitleAlign:'center',
                                headerLeftContainerStyle:{
backgroundColor:'white'},
                              }}
                            drawerContent={(props)=><MenuInterno
{...props}/>}>
                            <Drawer.Screen name="TopTabNavigator"
component={TopTabNavigator}
                                options={{
                                  title: 'VoyContigo',

                                }}
                            />

                             <Drawer.Screen name="CuentaScreen"
component={CuentaScreen}
                                options={{
                                  title: 'VoyContigo',

                                }}
                            />
                             <Drawer.Screen name="ContactosScreen"
component={ContactoScreen}
                                options={{
                                  title: 'VoyContigo',

                                }}
                                initialParams={{primeraVez:false}}
                            />
                             <Drawer.Screen name="AlertasTopTab"
component={AlertasTopTab}
                                options={{
```

```
                                title: 'VoyContigo',

                                }}
                            />
                             <Drawer.Screen name="TopTabConfig" component={
TopTabConfig}
                                options={{
                                  title: 'VoyContigo',

                                }}
                            />
        </Drawer.Navigator>
    )
}


const MenuInterno = ({ navigation }: DrawerContentComponentProps) => {
  const {logOut} = useContext(AuthContext);

  return (

        <View style={styles.container}>
              <View  style={stylesM.header}>
                <BackGround section='Login'/>
                    <Image source={require ('../img/logo.png')}
                        style={stylesM.imgLogo  }
                     />
                    <Text style={stylesM.tituloTxt }>
                       VoyContigo
                    </Text>
              </View>
             <View style={[stylesM.menuContainer]}>
                <TouchableOpacity style={stylesM.menuBoton} onPress={()=>
navigation.navigate('TopTabNavigator')}>
                      <Icon name="home-outline" style={stylesM.iconS}
size={20} color={ colors.primaryDark}  />
                      <Text style={stylesM.menuTexto}>
                              Inicio
                       </Text>
                </TouchableOpacity>
                <TouchableOpacity style={stylesM.menuBoton} onPress={()=>
navigation.navigate('CuentaScreen')}>
                      <Icon name="person-outline" style={stylesM.iconS}
size={20} color={ colors.primary}  />
                      <Text style={stylesM.menuTexto}>
```

```
                                Cuenta
                            </Text>
                    </TouchableOpacity>
                    <TouchableOpacity style={stylesM.menuBoton} onPress={()=>
navigation.navigate('ContactosScreen')}>
                            <Icon name="people-circle-outline"
style={stylesM.iconS} size={20} color={ colors.primary}  />
                            <Text style={stylesM.menuTexto}>
                                Contactos
                            </Text>
                    </TouchableOpacity>
                    <View style={[stylesM.menuContainerDown]}>
                        <TouchableOpacity  style={stylesM.menuBoton}
onPress={logOut}>
                                <Icon name="exit-outline" style={stylesM.iconS}
size={20} color={ colors.primary}  />
                                <Text style={stylesM.menuTexto}>
                                        Cerrar Sesión
                                </Text>
                        </TouchableOpacity>
                    </View>
                </View>

          </View>

        );
}


const stylesM = StyleSheet.create({

     header:{
        height:'25%',
        backgroundColor:colors.primary,
        alignItems:'center',
        justifyContent:'center'
     },
     menuContainer:{
       padding:'8%',
       height:'75%',

     },
     menuContainerDown:{
       marginVertical:'5%',
       borderTopWidth:2,
```

```
      borderTopColor:colors.secondaryDark,
      flexDirection:'column-reverse',
      paddingTop:'4%'
    },
    menuBoton:{
      flexDirection:'row',
      marginVertical:'4%',

    },
    menuTexto:{
      fontSize:RFValue(18),
      color:'black',
      marginHorizontal:'4%',

    },

    iconS:{
      width:'10%',
      fontWeight:'bold'
    },
    imgLogo:{
     width: '35%',
     height: '50%',
     marginHorizontal:'5%',
     marginVertical:'2%',
    },
    tituloTxt:{
      fontSize:RFValue(25),
      fontWeight:'bold',
      color:'white'
    }
});
```

StackNavigator.tsx

```
import React, { useContext } from 'react'
import { createStackNavigator } from '@react-navigation/stack';
import { RegistroScreen } from '../screens/RegistroScreen';
import { RegistroScreen2 } from '../screens/RegistroScreen2';
import { LoginScreen } from '../screens/LoginScreen';
import { MenuLateral } from './MenuLateral';
import { LoadingScreen } from '../screens/LoadingScreen';
import { PermissionsContext } from '../context/PermissionsContext';
import { PermissionsScreen } from '../screens/PermissionsScreen';
import { AuthContext } from '../context/AuthContext';
```

```
import { ContactoScreen } from '../screens/ContactosScreen';


export type RootStackParams = {
    MenuLateral:undefined;
    RegistroScreen:undefined;
    RegistroScreen2:undefined;
    MainScreen:{usuario:string, password:string};
  }

  const Stack = createStackNavigator();

 export const StackNavigator = () => {
        const {permissions}  = useContext(PermissionsContext);
        const {status,oneSingup} = useContext(AuthContext);
        if(permissions.locationStatus === 'unavailable') {
          return <LoadingScreen/>
        }
        if(status=== 'cheking') return <LoadingScreen/>;

    return (
        <Stack.Navigator  screenOptions={{
            headerShown:false,
          }} >

          {
            (permissions.locationStatus === 'granted')?
              (
                (status!== 'authenticado')?
                  (<>

                  <Stack.Screen name="LoginScreen" options={{title:'Login'}}
component={LoginScreen} />
                  <Stack.Screen name="RegistroScreen"
options={{title:'Registro'}} component={RegistroScreen} />
                  <Stack.Screen name="RegistroScreen2"
options={{title:'Registro'}} component={RegistroScreen2} />
                  </>
                  )
          :
                  ((oneSingup===true)?
                    <Stack.Screen name="ContactoScreen"
options={{title:'Registro'}}
component={ContactoScreen}  initialParams={{primeraVez:true}}/>
                     :
```

```
                    <>
                     <Stack.Screen name="MenuLateral"
options={{title:'Inicio'}} component={MenuLateral}  />
                    </>
                   )

              )
            :<Stack.Screen name="PermissionsScreen"
component={PermissionsScreen} />
            }
        </Stack.Navigator>
    )
}
```

TopTabNavigator.tsx

```
import React, { useEffect } from 'react';
import { createMaterialTopTabNavigator } from '@react-navigation/material-
top-tabs';
import { useSafeAreaInsets } from 'react-native-safe-area-context';
import Icon from 'react-native-vector-icons/Ionicons';
import { MapaScreen } from '../screens/MapaScreen';
import { AboutAlertasScreen } from '../screens/AboutAlertasScreen';
import { colors } from '../theme/colorsTheme';
import { StyleSheet, Text, TouchableOpacity, View } from 'react-native';
import { DrawerScreenProps } from '@react-navigation/drawer';
import { InicioScreen } from '../screens/InicioScreen';
import { RFValue } from 'react-native-responsive-fontsize';

const Tab = createMaterialTopTabNavigator();
interface Props extends DrawerScreenProps  <any,any> {};


export const TopTabNavigator = ({navigation}:Props) => {
    const {top:paddingTop} = useSafeAreaInsets();
    useEffect(() => {
        navigation.setOptions({
            headerRight: () =>(
             <View style={{backgroundColor:'white'}}>
              <TouchableOpacity style={{
                      marginLeft:10
                      }}
                  onPress={() => navigation.toggleDrawer()} >
```

```
                <Icon name='menu-outline' size={35}
color={colors.primary}/>
              </TouchableOpacity>
              </View>

            )} )

    }, [])
    return (
        <>
        <View style={styles.headerContainer}>
            <Text style={styles.titleHeader}>VoyContigo</Text>
             <TouchableOpacity style={styles.hamburgerBtn}
                  onPress={() => navigation.toggleDrawer()} >
                  <Icon name='menu-outline' size={33} color='white'/>
              </TouchableOpacity>
        </View>
        <Tab.Navigator  style={{paddingTop,
backgroundColor:colors.primaryDark}}
                        sceneContainerStyle={{backgroundColor:'white'}}

                        screenOptions={({route}) => ({
                                tabBarPressColor: colors.primaryLight,
                                tabBarShowIcon: true,
                                tabBarIndicatorStyle: {
                                    backgroundColor: colors.primaryLight,

                                },

                                tabBarActiveTintColor:'white',
                                tabBarStyle: {
                                borderTopColor: colors.primary,
                                borderTopWidth: 0,
                                elevation: 0,
                                backgroundColor:colors.primaryDark,
                                height:'10%',

                                },

                                style: {
                                    backgroundColor:'#000',

                                },
                                tabBarIcon:({color,focused}) => {
```

```
                                        let iconName:string='';
                                        switch (route.name) {
                                            case 'Inicio':
                                                iconName='home-outline';
                                                break;
                                            case 'Mapa':
                                                iconName='map-outline';
                                                break;
                                            case 'Incidencia':
                                                iconName='speedometer-
outline';
                                                break;
                                            case 'Alertas':
                                                iconName='create-outline';
                                                break;

                                            default:
                                                break;
                                        }

                                        return (
                                                <Text
style={{color:'white'}}>

                                                <Icon name={iconName}
size={20} color='white' />

                                                </Text>
                                                );
                                    },

                            })
                           }
            >

            <Tab.Screen name="Inicio"  component={InicioScreen} />
            <Tab.Screen name="Mapa"    component={MapaScreen}  />
            <Tab.Screen name="Alertas" component={AboutAlertasScreen} />
            </Tab.Navigator>

</>

    )
}

const styles = StyleSheet.create({
    headerContainer:{
```

```
        backgroundColor:colors.primaryDark,
        height:'7%',
        flexDirection:'row',
        alignItems:'center',
        justifyContent:'center'
    },
    titleHeader:{
        fontWeight: 'bold',
        fontSize:RFValue(25),
        fontFamily:'Hello Viktoria',
         color:'white'
    },
    hamburgerBtn:{
        position:'absolute',
        marginLeft:'2%',
        right:0,
    }


});
```

Carpeta Interfaces

appInterfaces.tsx

```
export interface Location {
    latitude:number;
    longitude:number;
}

export interface loginData {
    correo: string;
    password:string;
}
export interface LoginResponse {
        usuario:Usuario;
        token: string;

}
export interface RegistroDatos{
    nombre:string;
    apellido:string;
    correo:string;
    password:string;
}
export interface Usuario {
```

```
    idUsuario:string;
    nombre:string;
    apellido:string;
    correo:string;
    password:string;
    telefono:number;
}

export interface Contacto {
     idContacto:string;
     nombre:string;
     telefono:string;
     Usuario_idUsuario:string;
     indice:number;
}
export interface Contactos {
    contacto:Contacto;
}
export interface Alerta{
    idAlerta:number;
    tipo:string;
    mensaje:string;
    estado:number;
    Usuario_idUsuario:string;
    Contacto_idContacto:string;
}

export interface Ubicacion {
    idUbicacion: number;
    longitud:number;
    latitud: number;
    fecha:string;
    hora:number;
    tiempo:number;
    Alerta_idAlerta:number;


}

export interface UbicacionDescrip {
    ubicaciones:Ubicacion[],
    alertas:{
            idAlerta:number,
            tipo:string
```

```
            }[]
}

export interface UbicacionMarker {
    id: number;
    longitud:number;
    latitud: number;
    fecha:string;
    hora:number;
    tipo:string;
}
```

Carpeta Hooks

useForm.tsx

```
import { useState } from 'react';

export const useForm = <T extends Object>( initState: T ) => {

    const [state, setState] = useState( initState );

    const onChange = ( value: string, field: keyof T ) => {
        setState({
            ...state,
            [field]: value
        });
    }

    return {
        ...state,
        form: state,
        onChange,
    }

}
```

useLocation.tsx

```tsx
import React, { useEffect, useRef, useState } from 'react'
import { Location } from '../interfaces/appInterfaces';
import Geolocation from 'react-native-geolocation-service';



export const useLocation = () => {

    const [hasLocation,setHaslocation]=useState(false);
    const [routeLines,setRouteLines]=useState<Location[]>([]);
    const [initialPosition,setinitialPosition]=useState<Location>({
        longitude:0,
        latitude:0
    });
    const [locationDialog, setLocationDialog] = useState(true);
     const [userLocation,setUserLocation] = useState<Location>({
        longitude:0,
        latitude:0
     });

    const watchId = useRef<number>();
    const isMounted = useRef(true);
    useEffect(() => {
        isMounted.current = true;
        return () => {
            isMounted.current = false;
        }
    }, [])

    useEffect(()=> {
            getCurrentLocation().then( location => {
                if( !isMounted.current ) return;

                setinitialPosition(location);
                setUserLocation(location);
                setRouteLines( routes => [ ...routes, location ]);
                setHaslocation(true);
            });
    },[]) ;

    const getCurrentLocation = (): Promise<Location> =>{
            return new Promise ((resolve,reject) => {
                Geolocation.getCurrentPosition(
```

```
                    ({coords}) =>{

                         resolve({
                             latitude:coords.latitude,
                             longitude:coords.longitude
                         });

                    },
                    (err) => reject({err}), { enableHighAccuracy:true,
showLocationDialog: locationDialog, }

                     );

            });

    }
    const followUserLocation = () =>{
        watchId.current= Geolocation.watchPosition(
             ({coords}) => {
                    if( !isMounted.current ) return;
                   const location: Location = {
                        latitude:coords.latitude,
                        longitude:coords.longitude,
                   }

                   setUserLocation(location);
                   setRouteLines( routes => [ ...routes, location ]);
                   console.log(location);
             },
             (err) => console.log(err), {enableHighAccuracy:true,
distanceFilter:10, showLocationDialog: locationDialog}
         )
    }

    const stopFollowLocation = () => {
        if(watchId.current)
         Geolocation.clearWatch(watchId.current);
    }


    return {

            hasLocation,
            initialPosition,
            getCurrentLocation,
```

```
            followUserLocation,
            userLocation,
            stopFollowLocation,
            routeLines
    }
}
```

Carpeta Context

AuthContext.tsx

```
import React, { createContext, useReducer,useEffect} from 'react';
import voyApi from '../api/voyApi';
import { Usuario, LoginResponse, loginData, RegistroDatos, Contacto,
Alerta,Ubicacion, UbicacionDescrip, UbicacionMarker } from
'../interfaces/appInterfaces';
import { authReducer, AuthState } from './authReducer';
import AsyncStorage from '@react-native-async-storage/async-storage';
import { AxiosResponse } from 'axios';

type AuthContextProps = {
    errorMessage:string;
    token: string | null;
    user:Usuario| null;
    contacto:Contacto|null;
    contactos:Contacto[]| null;
    alertas:Alerta[]|null;
    status:'cheking'|'authenticado' | 'no-autenticado';
    oneSingup:boolean|'cheking'|null,
    singUp: (registroDatos: RegistroDatos)=> void;
    singIn: (logindata:loginData)=> void;
    logOut: ()=> void;
    removeError: ()=>void;
    removeOneSingUp:()=> void;
    registaContacto:(registraContacto:Contacto)=>unknown;
    obtenerContactos:(idUsuario:string)=>unknown;
    obtenerContacto:(idContacto:string)=>unknown;
    eliminaContacto:(idContacto:string)=>void;
    agregaAlerta:( registraAlerta:Alerta )=> void;
    eliminaContactoAlerta:(idContacto:string)=>void;
    obtenerAlertas:(idUsuario:string)=>unknown;
    obtenerAlertasContacto:(idUsuario:string)=>unknown;
    actualizaAlertas:(idAlerta:number,estado:number)=>void;
    actualizaContacto:(idContacto:string,nombre:string,numero:string)=>void;
```

```
    actualizaUsuario:(idContacto:string,tipo:string,dato:string)=>void;
    eliminaUsuario:(idUsuario:string)=>void;
    registraUbicacionAlerta:
(Longitud:number,latitud:number,Alerta_idAlerta:number) =>void;
    obtenerUbicaciones:()=>unknown;
    primeraVez:()=>void;

}
export const AuthContext = createContext({} as AuthContextProps);

const authInitialState:AuthState = {
     status: 'cheking',
     token:null,
     user:null,
     contacto:null,
     contactos:null,
     alertas:null,
     errorMessage:'',
     oneSingup:null,
}

export const AuthProvider = ({children} :any) =>{
       const [state, dispatch] = useReducer(authReducer, authInitialState);
       useEffect(() => {
        revisaToken();
       }, [])

       const revisaToken = async () =>{
           const token = await AsyncStorage.getItem('token');
           if(!token) return [dispatch({type:'noAutenticado'}),
console.log('cai en error del roken ', token)];
           const resp = await voyApi.get('/auth');
           if(resp.status !== 200){
                return [dispatch({type:'noAutenticado'}),
           }
            await AsyncStorage.setItem('token',resp.data.token);
           dispatch({
               type:'singUp',
               payload:{
                   token:resp.data.token,
                   user:resp.data.usuario
               },
           })

           const oneSingup = await AsyncStorage.getItem('oneSingup');
```

```
          if(oneSingup==='true'){
                    return dispatch({type:'OneSingUp'})
               }else{
                    return dispatch({type:'removeOneSingUp'})
               }

        }

       const singUp= async({nombre, apellido,
password,correo}:RegistroDatos)=> {
            try {
                const {data} = await
voyApi.post<LoginResponse>('/usuarios',{nombre,apellido,correo, password});
                dispatch({
                    type:'singUp',
                    payload:{
                        token:data.token,
                        user:data.usuario
                    }
                });
                await AsyncStorage.setItem('token',data.token);

            } catch (error:any) {
                console.log(error.response.data);
                dispatch({type:'addError',
payload:error.response.data.errors[0].msg || error.response.data.msg ||
'Verifique los compos' })
            }

       };
       const singIn= async({correo, password}:loginData)=> {
            try {
                const {data} = await
voyApi.post<LoginResponse>('/auth/login',{correo, password});
                dispatch({
                    type:'singUp',
                    payload:{
                        token:data.token,
                        user:data.usuario
                    }
                });
                await AsyncStorage.setItem('token',data.token);
            } catch (error:any) {
                console.log(error.response.data);
```

```
                dispatch({type:'addError', payload: error.response.data.msg
|| 'Información incorrecta' })

            }
        };

        const eliminaUsuario = async(idUsuario:string) => {
            try {
                const resp = await
voyApi.delete<Contacto>('/usuarios/'+idUsuario);
                await logOut();
                return resp.data;

            } catch (error:any) {
            console.log(error.response.data.msg);
                dispatch({type:'addErrorRegisterContact',
payload:  error.response.data.msg ||error.response.data.errors[0].msg||
'Erro al actualizar el campo'});
            }


        }

        const registaContacto= async({nombre,
telefono,Usuario_idUsuario}:Contacto) => {
            try {
                const resp = await
voyApi.post<Contacto>('/conalerta',{nombre,telefono, Usuario_idUsuario});
                dispatch({
                    type:'loadContact',
                    payload:{
                        contacto:resp.data,

                    }
                });
                return resp.data.idContacto;
            } catch (error:any) {
                dispatch({type:'addErrorRegisterContact', payload:
error.response.data.errors[0].msg || 'Es posible que el numero del contacto
ya exista' })
                return 'error';
            }
        };
        const eliminaContacto= async(idContacto:string) => {
                try {
```

```
                    const resp = await
voyApi.delete<Contacto>('/contactos/'+idContacto);
                } catch (error:any) {
                    console.log(error.response.data);
                    dispatch({type:'addErrorRegisterContact', payload:
error.response.data.errors[0].msg || 'Es posible que el numero del contacto
ya exista' })
                }
        };
        const obtenerContactos  = async(idUsuario:string) => {
                try {
                    const resp = await
voyApi.get<Contacto[]>('/contactos/'+idUsuario);
                    dispatch({
                        type:'loadContacts',
                        payload:{
                            contactos: resp.data,
                        }
                    });

                    await
AsyncStorage.setItem('contactos',JSON.stringify(resp.data));
                        return resp.data;

                } catch (error:any) {
                    let mes='No existen contactos del usuario con el id :
'+idUsuario;
                    if( error.response.data.errors[0].msg===mes ||
error.response.data.errors[0].msg==='No hay contactos agregados' ){
                        return [];
                }else{
                    dispatch({type:'addErrorRegisterContact', payload:
error.response.data.errors[0].msg|| 'Verifique los datos del contacto e
intentelo nuevamente'});
                }
                }

        };
        const actualizaContacto  = async(idContacto:string, nombre:string,
telefono:string) => {
            try {
                const resp = await
voyApi.put<Contacto>('/contactos/'+idContacto,{nombre,telefono});
                return resp.data;
            } catch (error:any) {
```

```
                dispatch({type:'addErrorRegisterContact', payload:
error.response.data.errors[0].msg|| 'Es necesario seleccionar una alerta'});
            }

        };

        const actualizaUsuario  =
async(idUsuario:string,tipo:string,dato:string) => {
            try {
                let tok:string='';
                let use: Usuario
={idUsuario:'',nombre:'',apellido:'',telefono:0,correo:'',password:''};
                let resp: AxiosResponse<LoginResponse, any>;
                    switch (tipo){
                        case 'nombre':
                                resp = await
voyApi.put<LoginResponse>('/usuarios/'+idUsuario,{nombre:dato});
                                use=resp.data.usuario;
                                tok=resp.data.token;
                               break;
                    case 'apellido':
                                resp = await
voyApi.put<LoginResponse>('/usuarios/'+idUsuario,{apellido:dato});
                                use=resp.data.usuario;
                                tok=resp.data.token;
                               break;
                    case 'correo':
                                resp = await
voyApi.put<LoginResponse>('/usuarios/'+idUsuario,{correo:dato});
                                use=resp.data.usuario;
                                tok=resp.data.token;
                               break;
                    case 'password':
                                resp = await
voyApi.put<LoginResponse>('/usuarios/'+idUsuario,{password:dato});
                                use=resp.data.usuario;
                                tok=resp.data.token;
                            break;
                };

                dispatch({
                    type:'singUp',
                    payload:{
                        token:tok,
                        user:use
```

```
                    }
                });
                await AsyncStorage.setItem('token',tok);

            } catch (error:any) {
                console.log(error.response.data.msg);
                dispatch({type:'addErrorRegisterContact',
payload:  error.response.data.msg ||error.response.data.errors[0].msg||
'Erro al actualizar el campo'});
            }

        };



        const obtenerContacto  = async(idContacto:string) => {
            try {
                const resp = await
voyApi.get<Contacto>('/contactos/c/'+idContacto);
                    return resp.data;
            } catch (error:any) {
                dispatch({type:'addErrorRegisterContact', payload:
error.response.data.errors[0].msg|| 'Verifique los datos del contacto e
intentelo nuevamente'});
            }
         };

        const obtenerAlertasContacto  = async(idContacto:string) => {
            try {
                const resp = await
voyApi.get<Alerta[]>('/conalerta/'+idContacto);
                console.log(resp.data);
                dispatch({
                    type:'loadAlertas',
                    payload:{
                        alertas:resp.data,
                    }
                });

                return resp.data;

            } catch (error:any) {
            }
        };
```

```
        const obtenerAlertas  = async(idUsuario:string) => {
            try {
                const resp = await
voyApi.get<Alerta[]>('/alertas/'+idUsuario);
                dispatch({
                    type:'loadAlertas',
                    payload:{
                        alertas:resp.data,
                    }
                });

                return resp.data;
            } catch (error:any) {
            }

        };
        const actualizaAlertas  = async(idAlerta:number, estado:number) => {
            try {
                const resp = await
voyApi.put<Alerta[]>('/alertas/'+idAlerta,{estado});
                console.log(resp.data,'mirame');
                return resp.data;
            } catch (error:any) {
                dispatch({type:'addErrorRegisterContact', payload:
error.response.data.errors[0].msg|| 'Es necesario seleccionar una alerta'});
            }

        };

        const agregaAlerta  =
async({tipo,Usuario_idUsuario,Contacto_idContacto}:Alerta) => {
            try {
                const resp = await
voyApi.put<Alerta>('/alertas',{tipo,Usuario_idUsuario,Contacto_idContacto});
                return resp.data;
            } catch (error:any) {
                console.log(error.response.data);
                dispatch({type:'addErrorRegisterContact',
payload:  error.response.data.errors[0].msg|| 'Verifique los datos del
contacto e intentelo nuevamente'});

            }

        };
```

```
        const eliminaContactoAlerta= async(idContacto:string) => {
            try {
                const resp = await
voyApi.delete<Contacto>('/conalerta/'+idContacto);
            } catch (error:any) {
                console.log(error.response.data);
                dispatch({type:'addErrorRegisterContact', payload:
error.response.data.errors[0].msg || 'Hubo un error al eliminar el contacto
intentelo más tarde' })
            }
        };

        const registraUbicacionAlerta = async
(Longitud:number,latitud:number,Alerta_idAlerta:number) => {
            try {
                const resp = await
voyApi.post<Ubicacion>('/ubicacion',{longitud,latitud,Alerta_idAlerta});

            } catch (error:any) {
                console.log(error.response.data);
                dispatch({type:'addErrorRegisterContact', payload:
error.response.data.msg ||error.response.data.errors[0].msg || 'Intentelo
nuevamente' })
            }
        }
        const obtenerUbicaciones = async() => {
            try {
                const {data} = await
voyApi.get<UbicacionDescrip>('/ubicacion');
                const {ubicaciones, alertas} =data;
                let marker : UbicacionMarker[] = [];
                let j=0;
                let tam= ubicaciones.length;
                for(let i=0; i<tam; i++){
                    if(ubicaciones[i].Alerta_idAlerta===alertas[i].idAlerta)
{
                    marker[j]={
                                id:j,
                                longitud:ubicaciones[i].longitud,
                                latitud: ubicaciones[i].latitud,
                                fecha:ubicaciones[i].fecha,
                                hora:ubicaciones[i].hora,
                                tipo:alertas[i].tipo
                            };
                            j++;
```

```
                    }
                }
                return marker;
            } catch (error:any) {
                console.log(error.response.data);
                dispatch({type:'addErrorRegisterContact', payload:
error.response.data.errors[0].msg || 'Intentelo nuevamente' });
                return [];
            }
        }



       const logOut= async()=> {
            await AsyncStorage.removeItem('token');
            dispatch({type:'logout'})

       };
       const removeError= ()=>{
            dispatch({type:'removeError'});

       };
       const primeraVez = async() =>{
            dispatch({
                type:'OneSingUp',
            });
            await AsyncStorage.setItem('oneSingup','true');
       }
       const removeOneSingUp = async ()=>{
            await AsyncStorage.removeItem('oneSingup');
            dispatch({type:'removeOneSingUp'})

       }
        return (
            <AuthContext.Provider value={{
                ...state,
                singUp,
                singIn,
                logOut,
                removeError,
                removeOneSingUp,
                registaContacto,
                obtenerContactos,
                obtenerContacto,
                actualizaContacto,
```

```
                eliminaContacto,
                eliminaUsuario,
                agregaAlerta,
                actualizaAlertas,
                obtenerAlertas,
                obtenerAlertasContacto,
                eliminaContactoAlerta,
                registraUbicacionAlerta,
                obtenerUbicaciones,
                primeraVez,
                actualizaUsuario,


            }}>
            {children}
            </AuthContext.Provider>
        )

}
```

authReducer.tsx

```
import { Alerta, Contacto, Usuario } from "../interfaces/appInterfaces";

export interface AuthState {
    status:'cheking'|'authenticado' | 'no-autenticado';
    token:string | null;
    errorMessage: string;
    user:Usuario| null;
    contacto:Contacto| null;
    contactos:Contacto[]| null;
    alertas:Alerta[]|null;
    oneSingup:boolean|null;
}


export type AuthAction =
           | {type: 'singUp', payload: {token:string , user:Usuario}}
           | {type: 'addError', payload:string}
           | {type: 'removeError'}
           | {type: 'noAutenticado'}
           | {type: 'logout'}
           | {type: 'loadContact',payload: { contacto:Contacto}}
           | {type: 'loadContacts',payload: { contactos:Contacto[]}}
           | {type: 'addErrorRegisterContact', payload:string}
```

```
          | {type: 'loadAlertas',payload: {alertas:Alerta[]}}
          | {type: 'removeOneSingUp'}
          | {type: 'OneSingUp'}



export const authReducer = (state:AuthState, action:AuthAction) : AuthState
=>{
       switch (action.type) {
           case 'addError':
                return{
                    ...state,
                    user:null,
                    status:'no-autenticado',
                    token:null,
                    errorMessage:action.payload
                }
            case 'removeError':
                 return{
                     ...state,
                     errorMessage:''
                 }
            case 'singUp':
                return {
                       ...state,
                        errorMessage:'',
                        status:'authenticado',
                        token:action.payload.token,
                        user:action.payload.user,
                }
            case 'OneSingUp':
                   return {
                       ...state,
                       oneSingup:true,
                   }
            case 'removeOneSingUp':
                 return {
                     ...state,
                     oneSingup:false,
                 }
            case 'logout':
            case 'noAutenticado':
                return {
                    ...state,
                     status:'no-autenticado',
```

```
                    oneSingup:null,
                    token:null,
                    user:null,
                }
            case 'addErrorRegisterContact':
                 return {
                    ...state,
                    errorMessage:action.payload,
                 }
            case 'loadContact':
                return {
                    ...state,
                    contacto:action.payload.contacto,
                    errorMessage:'',
                }
           case 'loadContacts':
                 return {
                    ...state,
                    contactos:action.payload.contactos,
                    errorMessage:'',
                 }
            case 'loadAlertas':
                return {
                    ...state,
                    alertas:action.payload.alertas,
                    errorMessage:'',
                }
           default:
               return state;
       }
}
```

PermissionsContext.tsx

```
import React,{ createContext, useEffect, useState } from "react";
import { AppState, Platform } from "react-native";
import { check, openSettings, PERMISSIONS, PermissionStatus, request } from
"react-native-permissions";



export interface PermissionsState {
    locationStatus: PermissionStatus;
}
export const permissionInitState: PermissionsState = {
```

```
          locationStatus: 'unavailable',
}

type PermisssionsContextProps = {
     permissions: PermissionsState;
     askLocationPermission: () => void;
     checkLocationPermission: () => void;
}

export const PermissionsContext = createContext({} as
PermisssionsContextProps) ;


export const PermissionsProvider = ({children}:any) =>{
    const [permissions, setPermissions] = useState( permissionInitState);

     useEffect(() => {
         checkLocationPermission();
         AppState.addEventListener('change',state =>{
              if(state !== 'active') return;
              checkLocationPermission();
         })

     }, [])


    const askLocationPermission = async () => {
        let permissionStatus: PermissionStatus;

        if(Platform.OS === 'ios'){
           permissionStatus = await
request(PERMISSIONS.IOS.LOCATION_WHEN_IN_USE);
        }else {
          permissionStatus = await
request(PERMISSIONS.ANDROID.ACCESS_FINE_LOCATION);
        }

         if(permissionStatus === 'blocked'){
             openSettings();
         }
        setPermissions({
            ...permissions,
            locationStatus: permissionStatus
        });
    }
```

```
    const checkLocationPermission = async () => {
        let permissionStatus: PermissionStatus;

        if(Platform.OS === 'ios'){
           permissionStatus = await
check(PERMISSIONS.IOS.LOCATION_WHEN_IN_USE);
        }else {
          permissionStatus = await
check(PERMISSIONS.ANDROID.ACCESS_FINE_LOCATION);
        }

        setPermissions({
            ...permissions,
            locationStatus: permissionStatus
        });
    }



    return(
        <PermissionsContext.Provider value={{
            permissions,
            askLocationPermission,
            checkLocationPermission
        }}>
            {children}
        </PermissionsContext.Provider>
    )
}
```

Carpeta Terminos

Términos.tsx

```
import React, { useState } from 'react'
import CheckBox from '@react-native-community/checkbox';
import { Modal, ScrollView, StyleSheet, Text, TouchableOpacity, View } from
'react-native'
import { RFValue } from 'react-native-responsive-fontsize';
import { colors } from '../../theme/colorsTheme';

interface Props {
    confirma: boolean;
    setConfirma:(bol:boolean) => void;
}
```

```
export const Terminos = ({confirma,setConfirma}:Props) => {

    const [muestra,setMuestra]=useState(false);
    const  op = () => {
        if(confirma === false){
             return setMuestra(true);
        }else{
         return setConfirma(false);
        }

    }


    return (
        <>
            <View style={styles.containerCB}>
                <CheckBox
                        value={confirma}
                        onValueChange={op}
                        style={styles.cb}
                        tintColors={{ true: colors.primaryDark, false:
'dark' }}
                />
                <Text>He leído y acepto los términos y condiciones</Text>
            </View>
            <Modal animationType='fade' visible={muestra} transparent={true}
>
                 <View style={styles.containerModal}>
                      <View style={styles.containerTerminos}>
                          <Text style={styles.titleHeader}>Términos y
condiciones</Text>
                          <ScrollView style={styles.scroll}>
                              <Text style={styles.titleTxt}>INFORMACIÓN
RELEVANTE</Text>
                              <Text style={styles.text}>
                              Los módulos que integran está aplicación
                              han sido diseñados para integrar información
de utilidad que pueda ser consultada
                              desde un dispositivo móvil, como lo es
información relativa a movilidad, conectividad,
                              participación ciudadana.
                              </Text>
                              <Text style={styles.text}>
```

```
                                Cualquier persona que desee hacer uso de los
servicios que ofrece esta aplicación
                                móvil, podrá hacerlo sujetándose a los
presentes “Términos y Condiciones de Uso” así
                                como a las políticas y principios que se
describen en el presente documento.
                                </Text>
                                <Text style={styles.titleTxt}>I.
OBJETO</Text>
                                <Text style={styles.text}>
                                Este documento describe los términos y
condiciones generales de uso de la aplicación
                                móvil en adelante "VoyContigo", misma que es
aplicación para dispositivos móviles
                                que integra en una sola aplicación el acceso
a los distintos módulos por los que está
                                integrada.
                                </Text>
                                <Text style={styles.titleTxt}>II. ACEPTACIÓN
DE LOS TÉRMINOS</Text>
                                <Text style={styles.text}>
                                El ingreso y utilización de la aplicación
"VoyContigo" implica que usted ha leído,
                                entendido y aceptado los presentes “Términos
y Condiciones de Uso”.
                                Y esta aplicación requerirá la autorización
expresa de los usuarios para acceder a las
                                funciones como Calendario, Cámara, Teléfono,
SMS y GPS para brindar atención y
                                seguimiento a las funcionalidades de alguno
de los módulos de la
                                aplicación.

                                </Text>
                                <Text style={styles.titleTxt}>III.
RESPONSABILIDAD SOBRE EL USO DE LA APLICACIÓN "VoyContigo"</Text>
                                <Text style={styles.text}>
                                Usted asume la responsabilidad de todas las
acciones que ejecute, en la utilización y
                                consulta de la "VoyContigo", mismas que se
considera que usted realiza
                                voluntariamente. Usted no utilizará, ni
permitirá que otros usuarios utilicen esta
                                aplicación móvil o cualquier servicio
prestado a través de ella, de forma contraria a lo
```

```
                                establecido en estos “Términos y Condiciones
de Uso”, así como a las disposiciones
                                legales aplicables a la funcionalidad que
ofrece la aplicación "VoyContigo". Es
                                responsabilidad de usted todo acto que se
realice en esta aplicación móvil, por medio
                                de su “Nombre de Usuario” y “Contraseña”;
asimismo, usted se responsabiliza del
                                cuidado y custodia de los mismos, para que
esta aplicación móvil únicamente sea
                                utilizada para los fines para los cuales ha
sido diseñada, por lo que cualquier mal uso
                                que se realice en su nombre, se presumirá
realizada por usted.
                                </Text>
                                <Text style={styles.text}>Usted reconoce y
acepta que las personas implicadas en la creación y desarrollo de
                                 la apliación quedan liberadas de
                                 toda responsabilidad por el mal uso que
llegara a realizarse en alguno de los módulos
                                 de la apliación "VoyContigo".</Text>
                                <Text style={styles.text}>
                                 Como todo desarrollo tecnológico, la
"VoyConrigo" es falible y estará en proceso
                                 permanente de mejora; por lo que existe la
posibilidad, aunque remota, de que ocurran
                                 fallas técnicas durante su uso; por lo que
la apliación "VoyConrigo" no
                                 garantiza que la aplicación móvil esté libre
de errores; por tal motivo, usted deslinda de
                                 toda la responsabilidad a las personas
implicadas en la creación y desarrollo,
                                 respondiendo de todos
                                 los daños y perjuicios derivados de las
circunstancias expuestas en este párrafo.
                                </Text>
                                <Text style={styles.text}>
                                Las personas implicadas en la creación y
desarrollo de "VoyContigo", como asi mismo "VoyContigo"
                                no asumirá ninguna responsabilidad por los
                                inconvenientes que usted pudiera experimentar
con el equipo de cómputo y accesorios
                                —hardware y software— utilizados para
conectarse a esta aplicación móvil; de igual
```

```
                                manera, tampoco asumirá responsabilidad
alguna por procedimientos que hayan
                                quedado inconclusos, por razones ajenas a las
funciones de la apliación.
                                </Text>
                                <Text style={styles.text}>
                                En ningún caso las personas implicadas en la
creación y desarrollo de "VoyContigo", como la misma apliación
                                será responsable por las
                                consecuencias del uso indebido o fraudulento
de la apliación "VoyContigo", cualquiera que sea
                                la causa del eventual daño.
                                </Text>
                                <Text style={styles.titleTxt}>IV. CAMBIOS Y
ACTUALIZACIONES DEL SERVICIO</Text>
                                <Text style={styles.text}>
                                Las personas implicadas en la creación y
desarrollo de "VoyContigo", tiene pleno derecho de
                                realizar modificaciones en los contenidos que
se ofrecen, ya sean temporales o
                                permanentes, en cualquier momento; por lo
tanto, no es posible garantizar la
                                disponibilidad ni la continuidad del
funcionamiento de la aplicación "VoyContigo" en todo
                                momento o durante el tiempo de actualización.
Las personas implicadas en la creación y desarrollo de "VoyContigo"
                                no
                                asume responsabilidad alguna por cualquier
falla que pudiera presentarse con el uso
                                de esta aplicación, ya sean por cambios y
actualizaciones, o por cualquiera otra causa
                                </Text>
                                <Text style={styles.text}>
                                Las personas implicadas en la creación y
desarrollo de "VoyContigo" se reserva el derecho de modificar,
                                restringir o
                                suprimir todos o cualquiera de los atributos
de los módulos de la apliación "VoyContigo", en
                                forma temporal o definitiva, sin que estas
medidas puedan ser objeto de requerimiento
                                alguno, ni de derecho a reclamar daños y
perjuicios por parte de usted.
                                </Text>
                                <Text style={styles.titleTxt}>V. POLÍTICAS DE
PRIVACIDAD Y PROTECCIÓN DE DATOS PERSONALES</Text>
```

```
                                <Text style={styles.text}>
                                    El tratamiento de sus datos personales se
limitará al cumplimiento de la finalidad
                                   establecida en el aviso de privacidad de
la aplicación.
                                </Text>
                                <Text style={styles.text}>
                                El tratamiento de sus datos personales se
limitará al cumplimiento de la finalidad
                                establecida en el aviso de privacidad de la
aplicación y se realizará de conformidad
                                con lo establecido en la Ley de Protección de
Datos Personales en Posesión de Sujetos
                                Obligados de la Ciudad de México y en los
Lineamientos Generales de Protección de
                                Datos Personales en Posesión de Sujetos
Obligados de la Ciudad de México.
                                </Text>
                                <Text style={styles.text}>
                                Los datos requeridos en la aplicación para el
registro de usuarios de la aplicación son:
                                Nombre y teléfono celular, correo
electrónico, usuario, contraseña y contactos,
                                 los cuales serán necesarios para para
consultar la información de utilidad
                                para el uso de usuarios en los distintos
módulos de la aplicación y para la
                                consulta de resultados de las alertas
emitidas.
                                </Text>
                                <Text style={styles.titleTxt}>VI. POLÍTICA DE
PROPIEDAD INTELECTUAL</Text>
                                <Text style={styles.text}>
                                 La apliación "VoyContigo" es una obra
protegida por las leyes mexicanas, internacionales y
                                 tratados en materia de propiedad
intelectual.</Text>
                                <Text style={styles.text}>
                                El uso de la aplicación "VoyContigo", no lo
convierte a usted en titular de ninguno de los derechos
                                 de propiedad intelectual del mismo, ni del
contenido o información a la que acceda.
                                 Queda prohibido utilizar el nombre, la marca
o el logotipo del mismo.
```

```
                                Asimismo, no se podrá eliminar, ocultar ni
alterar los
                                avisos legales que se muestran en la misma.
Sin perjuicio de lo anterior, usted podrá
                                imprimirlos, copiarlos o almacenarlos,
siempre y cuando sea para uso estrictamente
                                personal.

                              </Text>
                              <Text style={styles.titleTxt}>VII.
CONTROVERSIAS Y JURISDICCIÓN APLICABLE.</Text>
                              <Text style={styles.text}>
                                "LA AGENCIA" se reserva el derecho de
presentar acciones civiles o penales cuando se
                                advierta el uso indebido de la información
contenida en la aplicación "VoyContigo", o por el
                                incumplimiento de los presentes "TÉRMINOS Y
CONDICIONES DE USO"
                                </Text>
                              <Text style={styles.text}></Text>
                              <Text style={styles.titleTxt}>VIII.
ACEPTACIÓN DE LOS TÉRMINOS.</Text>
                              <Text style={styles.text}></Text>

                              <Text style={styles.text}>
                                Este documento describe los términos y
condiciones generales de uso de la aplicación
                                móvil en adelante "VoyContigo", misma que es
aplicación para dispositivos móviles
                                que integra en una sola aplicación el acceso
a los distintos módulos por los que está
                                integrada.

                                </Text>
                            </ScrollView>
                            <View style={[{marginTop:'2%'}]}>
                            <TouchableOpacity
style={[styles.btn,{marginTop:'2%'}]}
                                onPress={() =>
{setMuestra(false);setConfirma(true);} } >
                                  <Text
style={{color:colors.primaryDark}}>Acepto los términos</Text>
                            </TouchableOpacity>
                            <TouchableOpacity style={[styles.btn,]}
```

```
                                onPress={() =>
{setMuestra(false);setConfirma(false);} } >
                                    <Text
style={{color:colors.primaryDark}}>Cancelar</Text>
                            </TouchableOpacity>
                            </View>
                        </View>
                    </View>
                </Modal>
            </>
        )
}


const styles = StyleSheet.create({
    containerCB:{
        flexDirection:'row',
        alignItems:'center',
        justifyContent:'center',
        marginTop:'6%'
    },
    cb:{
        alignSelf: "center"
    },
    containerModal:{
        flex:1,
        backgroundColor:'rgba(0,0,0,0.3)',
        alignItems:'center',
        justifyContent:'center'
    },
    containerTerminos:{
        backgroundColor:'#eeeeee',
        width:'90%',
        height:'75%',
        alignItems:'center'
    },
    titleHeader:{
        color:'black',
        marginVertical:'5%',
        fontSize:RFValue(20)
    },
    scroll:{
        marginHorizontal:'3%',
        backgroundColor:'#f5f5f5',
        padding:'4%'
```

```
    },
    titleTxt:{
        fontWeight:'bold',
        textAlign:'center',
        marginVertical:'2%',
    },
    text:{
        textAlign:'justify'
    },

   btn:{
    borderWidth:1,
    borderColor:colors.primaryDark,
    alignItems:'center',
    justifyContent:'center',
    paddingVertical:'2%',
    paddingHorizontal:'5%',
    borderRadius: 10,
    marginBottom:15,
   }

});
```

Carpeta Maps

Map.tsx

```
import React, { useEffect, useRef,useContext ,useState} from 'react'
import MapView ,{Marker, Polyline} from 'react-native-maps'
import { useLocation } from '../../hooks/useLocation';
import { LoadingScreen } from '../../screens/LoadingScreen';
import { Fab } from './Fab';
import { colors } from '../../theme/colorsTheme';
import { AuthContext } from '../../context/AuthContext';
import { UbicacionDescrip, Ubicacion, UbicacionMarker } from
'../../interfaces/appInterfaces';
import { Image, View } from 'react-native';

interface Props {
    markers?: Marker[];

}


export const Map = ({markers}:Props) => {
```

```
        const {user,obtenerUbicaciones} = useContext(AuthContext);
        const usuar= JSON.stringify(user,null);
        const usuario = JSON.parse(usuar);
        const idUsuario= usuario.idUsuario;
        const [ubicaciones, setUbicaciones] =
useState<UbicacionMarker[]>([]);
        const obtenerUbicacion = async () =>{
                const datos=await obtenerUbicaciones() as UbicacionMarker[];
                (datos.length===0)? null :
                 setUbicaciones(datos);
                ;
        }
        useEffect( () =>{
            obtenerUbicacion();
        }
        ,[]);



    const{ hasLocation,
initialPosition,getCurrentLocation,followUserLocation ,
userLocation,stopFollowLocation,routeLines} = useLocation();
    const mapViewRef = useRef<MapView>();
    const following= useRef<boolean>(true);
     useEffect  (() => {
           followUserLocation();
           return ()=>{
               stopFollowLocation();
           }
     },[])
     useEffect  (() => {
           if(!following.current) return;
            const {latitude, longitude} = userLocation;
            mapViewRef.current?.animateCamera({
                center:{latitude,longitude}
           });

    },[userLocation])
    const centerPosition = async () => {
         const {latitude,longitude} =await getCurrentLocation();
           following.current = true;
        mapViewRef.current?.animateCamera({
             center:{latitude,longitude}
        });
    }
```

```
    if(!hasLocation){
        return <LoadingScreen/>
    }


    return (
        <>

        <MapView
         ref={(el)=> mapViewRef.current=el!}
           style={{flex:1}}
           showsUserLocation
                initialRegion={{
                latitude: initialPosition.latitude,
                longitude: initialPosition.longitude,
                latitudeDelta: 0.0922,
                longitudeDelta: 0.0421,
                }}
                onTouchStart={() => following.current=false}
            >

                {ubicaciones.length!==0 ? ubicaciones.map((marker, id) => (
                        <Marker
                        key={id}
                        coordinate={{latitude:marker.latitud ,
                                    longitude: marker.longitud}}
                        title={(marker.tipo==='verde')?
                                    'Riesgo Bajo'
                                : ((marker.tipo==='amarilla')?
                                    'Riesgo Medio'
                                : (marker.tipo==='roja')?
                                    'Riesgo Alto'
                                : 'VoyContigo')
                              }
                        description={'Se acciono el día : '+ marker.fecha +
' a las '+marker.hora}
                        pinColor={ (marker.tipo==='verde')? colors.alerta2
                                                          :
((marker.tipo==='amarilla')?
                                                                colors.alerta5
                                                               :
(marker.tipo==='roja')? colors.alerta8
                                                               : 'pink')
```

```
                                }

                        style={{ height:'15%', width:'10%',
alignItems:'center', padding:'2%'}}
                        >


                        <Image source={(marker.tipo==='verde')?
                                        require('../../img/pinV.png')
                                        : ((marker.tipo==='amarilla')?
                                        require('../../img/pinA.png')
                                        : (marker.tipo==='roja')?
                                        require('../../img/pinR.png')
                                        : 'pink')
                                        }
                                        style={{height: 45, width:32,
                                        }}
                                />
                        </Marker>
                    ))
                :null}


                        <Polyline
                            coordinates={ routeLines }
                            strokeColor={colors.primaryDark}
                            strokeWidth={ 4 }
                        />
                </MapView>
            <Fab iconName='locate-outline' onPress={centerPosition}
             style={ {position:'absolute', bottom:'3%',right:10}}/>
            <Fab iconName='refresh-outline' onPress={()=>
obtenerUbicacion()}
             style={ {position:'absolute', bottom:'15%',right:10}}/>
        </>
    )
}
```

Fab.tsx

```
import React from 'react'
import { StyleProp, View, ViewStyle, TouchableOpacity, StyleSheet } from
'react-native';
import Icon from 'react-native-vector-icons/Ionicons';
```

```
import { colors } from '../../theme/colorsTheme';

interface Props {
    iconName: string;
    onPress: () => void;
    style?: StyleProp<ViewStyle>
}



export const Fab = ({iconName, onPress,style={}}:Props) => {
    return (
        <View style={{...style as any}}>
            <TouchableOpacity style={styles.btn}
                              onPress={onPress}>
                <Icon  name={iconName} color='white' size={34}/>
            </TouchableOpacity>

        </View>
    )
}


const styles = StyleSheet.create({
    btn:{
        zIndex:9999,
        height:50,
        width:50,
        backgroundColor:colors.primary,
        borderRadius:100,
        justifyContent:'center',
        alignItems:'center',
        shadowColor:'#000',
        shadowOffset:{
            width:0,
            height:3,

        },
        shadowOpacity:0.27,
        shadowRadius:4.65,
        elevation:6

    },
});
```

Carpeta Contacto

ContactoAdd.tsx

```
import React, { useContext, useEffect, useState } from 'react';
import { StyleSheet, Text, TextInput, View, TouchableOpacity, Alert} from
'react-native';
import { useForm } from '../../hooks/useForm';
import { AuthContext } from '../../context/AuthContext';
import { colors } from '../../theme/colorsTheme';
import { styles } from '../../theme/appTheme';
import { MultiSelect } from 'react-native-element-dropdown';
import { loginStyles } from '../../theme/loginTheme';
import { widthPercentageToDP as wp, heightPercentageToDP as hp } from
'react-native-responsive-screen';
import { RFValue } from 'react-native-responsive-fontsize';
import  Icon  from 'react-native-vector-icons/Ionicons';
import { Alerta, Contacto } from '../../interfaces/appInterfaces';


interface DataLabel{
    noLabel:number,
    alias:string,
    numero:string,
    idContacto:string,
    idUsuario:string,
    verificaCampos: boolean;
    setVerificaCampos:(bol:boolean) => void;
    habilitaCampo:boolean;
    getContact:() => void;

}

interface DataAlerta{
    idAlerta:number,
    estado:number,
    tipo:string,
}
const data = [
    { label: 'Verde',   value: '1' },
    { label: 'Amarilla',value: '2' },
    { label: 'Roja',    value: '3' },
  ];
```

```
export const ContactoAdd =
({noLabel,alias,numero,idContacto,idUsuario,verificaCampos,setVerificaCampos
,habilitaCampo,getContact}:DataLabel) => {
    const [value, setValue] = useState(null);
    const renderItem = (item: any) => {
        return (
        <View style={stylesC.item}>
            <Text style={stylesC.textItem}>{item.label}</Text>
            {item.value === value && (
            <Icon name='checkmark-outline' size={14} color='green' />
            )}
        </View>
        );
    };

    const
{actualizaContacto,actualizaAlertas,eliminaContactoAlerta,obtenerAlertasCont
acto,errorMessage,removeError} = useContext(AuthContext);
    const [selected, setSelected] = useState<string[]>(['','','']);
    const [selectedPrincipal, setSelectedPrincipal] = useState(false);
    const [despliega,setDespliega] = useState(habilitaCampo);
    const [habilitaCampos,setHabilitaCampos]=useState(habilitaCampo);
    const {nombre,telefono,onChange} =useForm ({
      nombre:alias,
      telefono:numero
    });



    const deleteConfirm =async () => {
        await eliminaContactoAlerta(idContacto);
        await getContact();
        setVerificaCampos(false);
    }
    const eliminaItem =  () =>{
        Alert.alert('Se eliminara el contacto', '¿Estás seguro de querer
eliminar el contacto?',
        [
            {
                text:'Si',
                onPress:()=>deleteConfirm()
```

```
            },
            {
                text:'No',

            }
        ]
        );

    }
    const [alertasData,setAlertasData] =useState<DataAlerta[]>([]);
    const obtenerAlertasData = async ()=>{
        const a=await obtenerAlertasContacto(idContacto) as Alerta[];
        const n =[];
        const alertas:DataAlerta[] =[];

        for(let i = 0 ; i<a.length ; i++){
            let s='';
            switch(a[i].tipo){
                case 'verde':
                    if(a[i].estado===1)
                    n[i]='1';
                    s='1';
                    break;
                case 'amarilla':
                    if(a[i].estado===1)
                    n[i]='2';
                    s='2';
                    break;
                case 'roja':
                    if(a[i].estado===1)
                    n[i]='3';
                    s='3';
                    break;
            }
            alertas[i]={idAlerta:a[i].idAlerta,estado:a[i].estado,tipo:s};
        }


        const activos=[];
        let j=0;
        for(let i=0; i<n.length; i++){
            if(n[i]!==undefined){
                activos[j]=n[i];
                j++;
            }
```

```
        }
        setAlertasData(alertas);
        setSelectedPrincipal(activos.length===0? false: true);
        setSelected(activos);


        return n;
    }

    const actualizaCampos= ()=>{

        if(selected.length===0) return Alert.alert('Error al guardar','Es
necesario asignarle una alarma al contacto');
        for(var i=0; i<alertasData.length;i++){
            let estado=0;
            for(var j=0; j<selected.length;j++){
               if(alertasData[i].tipo===selected[j]){
                   estado=1;
               }
            }
            actualizaAlertas(alertasData[i].idAlerta,estado)
            setVerificaCampos(false);

        }
        actualizaContacto(idContacto,nombre,telefono);
        setHabilitaCampos(false);
        setSelectedPrincipal(true);

    }
     useEffect( () =>{
        obtenerAlertasData();

       if(errorMessage.length ===0) return;

       Alert.alert('No se puede agregar', errorMessage,
       [{
           text:'ok',
           onPress:removeError
       }
       ]
       );
   }
   ,[errorMessage])
```

```
   return  (

        <View style={stylesC.container}>
            <View
style={[(despliega)?stylesC.containerHeaderT:stylesC.containerHeaderF,styles
C.headerColor]}>
                <Text style={[stylesC.titleHeader]}>Contacto
{noLabel}</Text>
                <TouchableOpacity style={stylesC.btnHeaderDes}
                                  activeOpacity={0.08}
                                  onPress={()=>(!despliega) ?
setDespliega(true):setDespliega(false) }
                >
                <Icon  name={(despliega) ?'caret-up-outline':'caret-down-
outline'} size={20} color='white' />
                </TouchableOpacity>
            </View>
            {(despliega) ?
                <View style={stylesC.containTarget}>
                    <View style={{flexDirection:'row'}}>
                            <View  style={stylesC.containTargetMitad}>
                                    <Text style={loginStyles.label}>
Nombre/Alias</Text>
                                    <TextInput
                                        editable={(!habilitaCampos) ? false
: true }
                                        placeholderTextColor='rgba(0,0,0,0.4
)'
                                        keyboardType='default'
                                        underlineColorAndroid={colors.primar
yDark}
                                        style={stylesC.inputText}
                                        selectionColor={colors.pink[100]}
                                        autoCapitalize='words'
                                        onChangeText={(value)=>
onChange(value,'nombre')}
                                        value={nombre}


                                    />
                                    <Text
style={stylesC.label}>Número</Text>
                                    <TextInput
```

```
                                        editable={(!habilitaCampos) ? false
: true }
                                        placeholder={telefono}
                                        placeholderTextColor='rgba(0,0,0,0.4
)'
                                        keyboardType='phone-pad'
                                        underlineColorAndroid={colors.primar
yDark}
                                        style={stylesC.inputText}
                                        selectionColor={colors.pink[100]}
                                        autoCapitalize='words'
                                        onChangeText={(value)=>
onChange(value,'telefono')}
                                        value={telefono}

                                    />

                                </View>
                                <View
style={[stylesC.containTargetMitad,{flexDirection:'column'}]}>
                                    <Text style={loginStyles.label}>Designar
Alarma</Text>

                                    <MultiSelect
                                        disable={(habilitaCampos)?
false:true}
                                        style={stylesC.dropdown}
                                        placeholderStyle={stylesC.placeholde
rDD}
                                        selectedTextStyle={stylesC.selectedT
extDD}
                                        selectedStyle={stylesC.selectedDD}
                                        activeColor={colors.primaryLight}
                                        iconColor={'#000'}
                                        data={data}
                                        labelField="label"
                                        valueField="value"
                                        placeholder="Selecciona Alerta"
                                        searchPlaceholder="Buscar..."
                                        value={selected}
                                        onChange={item => {
                                            setSelected(item );
                                            setValue(item.value);
                                        }}
                                        renderItem={renderItem}
```

```
                                    />
                            </View>
                    </View>

                    <View style={stylesC.containerBtns}>

                            {habilitaCampos?

                            <TouchableOpacity
style={[styles.btn,styles.btnDark,stylesC.btnDimension]}
                            activeOpacity={0.08}
                            onPress={()=>{actualizaCampos()}}

                            >
                            <Icon name='checkmark' size={RFValue(20)}
color='white'  />
                            </TouchableOpacity>
                            :null}
                           {(!habilitaCampos)?
                                <TouchableOpacity style={[styles.btn,
styles.btnDark,stylesC.btnDimension]}
                                                    activeOpacity={0.08}
                                                    onPress={()=>{verificaCampos
?
                                                         Alert.alert('Modificar
contacto','Para poder modificar otro contacto salve los cambios del contacto
que se encuentre modificando e intentelo nuevamente')
                                                    :setHabilitaCampos(true)
,setVerificaCampos(true)}}
                                                    >

                                <Icon name='create-outline'
size={RFValue(20)} color='white'  />
                                </TouchableOpacity>
                            :
                                (selectedPrincipal)?

                                    <TouchableOpacity
style={[styles.btn,styles.btnDark,stylesC.btnDimension]}
                                    activeOpacity={0.08}
                                    onPress={()=>{setHabilitaCampos(false),s
etVerificaCampos(false)}}
                                    >
                                    <Icon name='close-outline'
size={RFValue(20)} color='white'  />
```

```
                                    </TouchableOpacity>
                                :
                                null

                            }

                             <TouchableOpacity
style={[styles.btn,styles.btnDark,stylesC.btnDimension]}
                                                activeOpacity={0.08}
                                                onPress={()=>{verificaCampos ?
                                                            Alert.alert('Elimina
r contacto','Para poder eliminar el contacto salve los cambios del contacto
que se encuentre modificando e intentelo nuevamente')
                                                            :eliminaItem()}
                                                        }
                            >
                            <Icon name='trash-outline' size={RFValue(20)}
color='white'  />
                            </TouchableOpacity>
                    </View>
                </View>
            : null}

        </View>

    )

}


const stylesC = StyleSheet.create({
    container:{
        marginVertical:'1%'
    },
    containerHeaderT:{
        borderTopLeftRadius:10,
        borderTopRightRadius:10
    },
    containerHeaderF:{
        borderRadius:8,
    },
    headerColor:{
        backgroundColor:colors.primary
    },
    titleHeader:{
```

```
        fontSize:RFValue(18),
        fontWeight:'bold',
        color:'white',
        textAlign:'center',
        margin:2,
    },
    btnHeaderDes:{
        position:'absolute',
        right:'1%',
        marginVertical:'1%'
    },
    containTarget:{
        backgroundColor:colors.secondary,
        borderBottomEndRadius:15,
        borderBottomStartRadius:15,
    },
    containTargetMitad:{
        width:'50%',
        paddingHorizontal:'5%',
        paddingVertical:'1%'
    },
    inputText:{
        color:'#000',
    },
    label:{
        color:colors.ThemeText,
        fontWeight:'bold',
    },
    containerBtns:{
        flexDirection:'row',
        justifyContent:'flex-end'
    },

    btnDimension:{
        paddingHorizontal:hp(3),
        paddingVertical:hp(1),
        borderRadius: 10,
        margin:hp(1.04),
    },
    buttonText:{
        color:colors.primaryDark,
        fontSize:hp(2.4),
    },
    dropdown: {
        borderBottomColor: colors.primaryDark,
```

```
        borderBottomWidth: 1,
    },
    placeholderDD: {
        color:colors.ThemeText,
        fontSize:RFValue(14),

    },
    selectedTextDD: {
        fontSize:RFValue(14),
        color:colors.primaryDark
    },

    selectedDD: {
        borderRadius: 12,
        borderColor:colors.primaryDark,

    },
    item: {
        padding: '5%',
        flexDirection: 'row',
        alignItems: 'center',
    },
    textItem: {
        color:'black',
        fontSize:RFValue(14),

    },

});
```

Carpeta Alerta

sendAlerta.tsx

```
import React from 'react';
import {NativeModules,PermissionsAndroid, Alert} from 'react-native';
import { TextAlertVerde,TextAlertAmarilla ,TextAlertRoja} from
'../../textos/AlertMessages';

var DirectSms = NativeModules.DirectSms;


export const sendAlerta = async (numero:string,
alerta:string,latitude:number,longitude:number) => {
    let mensaje= '';
```

```
    let
ubicacionTxt='http://maps.google.com/maps?q='+latitude+','+longitude;
    const sendDirectSms = async () => {
        if (numero) {
        try {
            const granted = await PermissionsAndroid.request(
            PermissionsAndroid.PERMISSIONS.SEND_SMS,
            {
                title: 'Envió de Mensajes',
                message:
                'Para poder enviar el mensaje es necesario que otorgue los
permisos ' +
                '¿Desea que VoyContigo pueda enviar mensajes?.',
                buttonNeutral: 'Pregúntame Despues',
                buttonNegative: 'Cancelar',
                buttonPositive: 'OK',
            },
            );
            if (granted === PermissionsAndroid.RESULTS.GRANTED) {
                switch (alerta) {
                    case 'verde':
                        {
                            mensaje=TextAlertVerde+ubicacionTxt+ '  |
'+alerta;
                        }
                        break;
                    case 'amarilla':
                       {
                           mensaje=TextAlertAmarilla+ ubicacionTxt + '  |
'+alerta;
                       }

                       break;
                    case 'roja':
                        {
                            mensaje=TextAlertRoja+ubicacionTxt+ '  |
'+alerta;
                        }
                        break;

                    default:
                        break;
                }

                DirectSms.sendDirectSms(numero, mensaje);
```

```
                Alert.alert('Alerta enviada','La alerta '+alerta+' fue
accionada con exito');
            } else {
            Alert.alert('Envio Denegado','Los permisios para el envio de
mensajes ha sido denegado, activelos para poder mandar las alertas');
            }
        } catch (error:any) {
            console.warn(error);
            Alert.alert(' '+error);
        }
        }
    };

      return sendDirectSms() ;

};
```

Carpeta API

voyAPI.tsx

```
import axios from 'axios';
import AsyncStorage from '@react-native-async-storage/async-storage';

const baseURL= 'http://10.0.0.4:8000/api';

const voyApi = axios.create({baseURL});

voyApi.interceptors.request.use(
    async(config:any) => {
        const token = await AsyncStorage.getItem('token');
        if(token){
            config.headers['voy-token']=token;
        }
        return config;
    }
);

export default voyApi;
```

Carpeta Header

BackGround.tsx

```
import React from 'react'
import { Image, ImageBackground, StyleSheet, View } from 'react-native';
import { colors } from '../../theme/colorsTheme';
import { heightPercentageToDP as hp, widthPercentageToDP as wp } from
'react-native-responsive-screen';

interface Props {
  section:string;
  size?:number;

}

export const BackGround = ({section}:Props) => {
    return (
        <>
            <ImageBackground source={require ('../../img/header.jpg')}
resizeMode="cover"
            style={{position:'absolute',height:'100%', width:'100%'}}/>
        </>
    )
}

const styles = StyleSheet.create({
  container:{
    position:'absolute',
    height:'100%',
    width:'100%'
  },

});
```

LogoHeader.tsx

```
import React from 'react'
import { Image, StyleSheet, Text, View } from 'react-native';
import { RFPercentage, RFValue } from 'react-native-responsive-fontsize';
import { heightPercentageToDP as hp, widthPercentageToDP as wp } from
'react-native-responsive-screen';
import { BackGround } from './BackGround';
```

```
interface Props {
    section:string;
    size?:number;

}
export const LogoHeader = ({section}:Props) => {
       let title= (section === 'Login') ? 'VoyContigo' : 'Registro';
    return (
      <>
      <BackGround section={title}/>
      <View style={(section === 'Login') ? stylesL.containerImgLogin :
stylesR.containerImgLogin }>

            <Image source={require ('../../img/logo.png')} /
                  style={ (section === 'Login') ? stylesL.imgLogo :
stylesR.imgLogo }
            />
            <Text style={(section === 'Login') ? stylesL.tituloTxt
:stylesR.tituloTxt }>
                {title}
            </Text>



        </View>
    </>
    )

}

const stylesL = StyleSheet.create({
    containerImgLogin:{
        height:  '100%',
        justifyContent:'center',
        alignItems:'center',
        padding:'5%',
    },
    imgLogo:{
        width: wp('35%'),
        height: hp('19%'),
        marginHorizontal:'5%',
        marginVertical:'2%',
    },
    tituloTxt:{
        color:'#FFF',
```

```
        fontSize: RFValue(30),
        fontWeight:'bold',
        fontFamily:'Hello Viktoria'
    },

});


const stylesR = StyleSheet.create({
    containerImgLogin:{
        height:  '100%',
        justifyContent:'center',
        alignItems:'center',
        padding:'5%',
    },
    imgLogo:{
        width: wp('24%'),//35
        height: hp('13%'), //19
        marginHorizontal:'5%',
        marginVertical:'2%',
    },
    tituloTxt:{
        color:'#fff',
        fontSize: RFValue(30),
        fontWeight:'bold',
        fontFamily:'Hello Viktoria'
    },

});
```

Java\com\voycontigo

ModuloSendSms.java

```
package com.voycontigo;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.Callback;
import com.facebook.react.bridge.ReactMethod;
import com.facebook.react.uimanager.IllegalViewOperationException;

import android.telephony.SmsManager;

public class ModuloSendSms extends ReactContextBaseJavaModule {
```

```
    public ModuloSendSms(ReactApplicationContext reactContext) {
        super(reactContext);
    }

    @Override
    public String getName() {
        return "DirectSms";
    }

    @ReactMethod
    public void sendDirectSms(String phoneNumber, String msg) {

        try {
            SmsManager smsManager = SmsManager.getDefault();
            smsManager.sendTextMessage(
              phoneNumber, null, msg, null, null
            );
        } catch (Exception ex) {
            System.out.println("No se pude enviar el mensaje.");
        }
    }
}
```

PaqueteSendSms.java

```
package com.voycontigo;
import com.facebook.react.ReactPackage;
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.uimanager.ViewManager;
import com.voycontigo.ModuloSendSms;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class PaqueteSendSms implements ReactPackage {

    @Override
    public List<ViewManager> createViewManagers(
      ReactApplicationContext reactContext
    ) {
        return Collections.emptyList();
    }

    @Override
```

```
    public List<NativeModule> createNativeModules(
            ReactApplicationContext reactContext) {
        List<NativeModule> modules = new ArrayList<>();
        modules.add(new ModuloSendSms(reactContext));
        return modules;
    }
}
```