ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN SISTEMA PROTOTIPO DISTRIBUIDO BASADO EN NOTIFICACIONES PARA TERREMOTOS

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

CRISTIAN JONATHAN RONDA SANGOVALIN

HENRY MAURICIO VILLAVICENCIO CATOTA

DIRECTOR: ING. RAÚL DAVID MEJÍA NAVARRETE, M.Sc.

Quito, enero de 2020

AVAL

Certifico que el presente trabajo fue desarrollado por Cristian Jonathan Ronda Sangovalin y Henry Mauricio Villavicencio Catota, bajo mi supervisión.

ING. RAÚL DAVID MEJÍA NAVARRETE, M.Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Nosotros, Cristian Jonathan Ronda Sangovalin y Henry Mauricio Villavicencio Catota, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejamos constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

_______________________________	_______________________________
CRISTIAN JONATHAN RONDA SANGOVALIN	HENRY MAURICIO VILLAVICENCIO CATOTA

DEDICATORIA

Con todo mi cariño a mí familia, Amelia, Jorge y Zaira. También, a mis amigos que han estado ahí durante mi vida estudiantil.

Cristian

DEDICATORIA

A mis padres por haber sido mi apoyo incondicional.

Henry

AGRADECIMIENTO

Agradezco a mis padres por brindarme su cariño, apoyo y aconsejarme en momentos que me encontraba confundido. Los amo.

A mi hermana Zairita, por los abrazos al llegar a casa y hacerme reír con sus locuras.

A mis amigos del colegio que han estado ahí: Hugo, Isaac, Katherine, Josy, Pamela, Ricardo y Wladimir; por compartir tantos años de amistad. Por su apoyo en mis estudios. Gracias.

A mis amigos de la Poli: Henry, Javier, Johana, Jonathan, Oscar, Victoria y demás amigos; por haber compartido tantos momentos dentro y fuera de la universidad.

Un especial agradecimiento al Ingeniero David Mejía por el tiempo, energía y paciencia invertidos en este Trabajo de Titulación.

Gracias a todos los que me ayudaron y contribuyeron en mi vida universitaria.

Cristian Ronda

AGRADECIMIENTO

A mis padres Ángel y María quienes con su amor, trabajo y sacrificio me han permitido llegar a cumplir un sueño más.

A mis hermanos Alejandro y Camila que con sus palabras me hacían sentir orgulloso. Ojalá algún día me convierta en la persona que sea su ejemplo e inspiración.

A Andrea por ser mi cómplice, mi confidente y mi amiga.

A Oscar, Javier, Johana, Victoria y a todos mis amigos gracias por compartir tantos momentos. Sin ustedes la universidad hubiese sido muy aburrida.

A Cristian, coautor de este documento, gracias por el apoyo y todo lo que hiciste por sacar este proyecto adelante.

Un agradecimiento especial al ingeniero David Mejía por su tiempo, paciencia y ayuda brindada en este proyecto.

Finalmente, gracias a todas aquellas personas que contribuyeron de u otra forma en mi vida universitaria.

Henry Villavicencio

ÍNDICE DE CONTENIDO

AVAL I
DECLARACIÓN DE AUTORÍA II
DEDICATORIA III
DEDICATORIA IV
AGRADECIMIENTO V
AGRADECIMIENTO VI
ÍNDICE DE CONTENIDO VII
RESUMEN IX
ABSTRACT X
1 INTRODUCCIÓN 1
1.1 OBJETIVOS 2
1.2 ALCANCE 2
1.3 MARCO TEÓRICO 3
1.3.1 TECNOLOGÍAS 3
1.3.2 METODOLOGÍA SCRUM 15
1.3.3 HERRAMIENTAS DE DESARROLLO 17
2 METODOLOGÍA 24
2.1 FASE INICIAL 24
2.1.1 VISIÓN DEL PROYECTO 24
2.1.2 ROLES SCRUM 24
2.1.3 ENCUESTAS 25
2.1.4 HISTORIAS DE USUARIO 28
2.1.5 REQUERIMIENTOS 29
2.2 FASE DE PLANEACIÓN 30
2.2.1 ARQUITECTURA DEL PROTOTIPO 30
2.2.2 CASOS DE USO 31
2.2.3 PRODUCT BACKLOG 32
2.2.4 SPRINT BACKLOG 35
2.2.5 CONVENSIONES DEL CÓDIGO FUENTE 36
2.3 FASE DE IMPLEMENTACIÓN 38

2.3.1 SPRINT 1 ... 38
2.3.2 SPRINT 2 ... 43
2.3.3 SPRINT 3 ... 52
2.3.4 SPRINT 4 ... 54
2.3.5 SPRINT 5 ... 58
2.3.6 SPRINT 6 ... 63
2.3.7 SPRINT 7 ... 69
2.3.8 SPRINT 8 ... 76
2.3.9 SPRINT 9 ... 88
3 RESULTADOS Y DISCUSIÓN ... 97
3.1 PRUEBAS DE FUNCIONALIDAD DE LOS SERVICIOS ... 97
3.1.1 PRUEBA DEL SERVICIO DE BASE DE DATOS FEED ... 97
3.1.2 PRUEBA DEL SERVICIO DE BASE DE DATOS MAP ... 97
3.1.3 PRUEBA DEL SERVICIO DE BASE DE DATOS USER ... 98
3.1.4 PRUEBA DEL SERVICIO DE BASE DE DATOS SERVER ... 98
3.1.5 PRUEBA DEL SERVICIO DE SMS ... 99
3.2 PRUEBAS DE INTEGRACIÓN ... 100
3.2.1 AUTENTICACIÓN DEL USUARIO POR NÚMERO CELULAR ... 100
3.2.2 REGISTRO DEL USUARIO ... 102
3.2.3 OBTENCIÓN DE LA POSICIÓN DEL USUARIO ... 105
3.2.4 LECTURA DE RECOMENDACIONES *OFFLINE* ... 105
3.2.5 MAPA COLABORATIVO ... 105
3.2.6 NOTIFICACIÓNDE TERREMOTOS ... 108
3.2.7 ADMINISTRACIÓN DE CONTENIDOS Y CONFIGURACIÓN DEL PROTOTIPO ... 110
3.3 PRUEBAS DE VALIDACIÓN ... 114
3.4 CORRECIÓN DE ERRORES ... 116
4 CONCLUSIONES Y RECOMENDACIONES ... 121
4.1 CONCLUSIONES ... 121
4.2 RECOMENDACIONES ... 123
5 REFERENCIAS BIBLIOGRÁFICAS ... 125
6 ANEXOS ... 128
ORDEN DE EMPASTADO ... 129
ORDEN DE EMPASTADO ... 130

RESUMEN

Este Trabajo de Titulación presenta un prototipo, para la notificación del estado de las personas después de un terremoto, el cual utiliza SMS (Short Message Service) como una alternativa de comunicación.

El prototipo está conformado por las aplicaciones: servidor y los clientes web y móvil, que consumen los microservicios de SMS, base de datos y consulta de terremotos.

La aplicación móvil cuenta con las funcionalidades de registro/autenticación de usuario, lectura de recomendaciones offline, registro de la ubicación en un tiempo configurable, un mapa colaborativo, notificación de terremotos y envío del estado vía SMS.

La aplicación servidor se encarga de notificar los terremotos, recibir el estado del usuario y reenviarlo a sus contactos a través de SMS.

La aplicación web se encarga del manejo de los contenidos del prototipo, además de la configuración de los parámetros para el envió de notificaciones.

En el primer capítulo se presenta el marco teórico sobre: tecnologías utilizadas, metodología de desarrollo Scrum y se detallan las herramientas utilizadas.

En el segundo capítulo se presentan las encuestas, las historias de usuario, y se recogen los requisitos del prototipo para definir su arquitectura y se muestra el diseño e implementación de este.

El tercer capítulo presenta los resultados de las pruebas y las encuestas de validación efectuadas.

En el cuarto capítulo se presentan las conclusiones y recomendaciones obtenidas en el desarrollo de este Trabajo de Titulación.

Finalmente, como anexos se incluyen: las entrevistas, el código del proyecto de la aplicación móvil, web y servidor; y el manual de usuario.

PALABRAS CLAVE: Android, Firebase, microservicios, React, React Native, Redux, SMS, terremoto

ABSTRACT

The current final career project introduces a notification prototype based on SMS (Short Message Service), which is an alternative for communication in an earthquake.

The prototype has several components: server, mobile and web client, which uses the microservices like SMS, database and earthquake consultation.

The mobile application has functionalities as user registration/authentication, offline recommendations reading, location registration in a configurable time, a collaborative map, earthquake notifications and sending the status through SMS.

The server application is responsible for notifying earthquakes, receiving user 's responses and forwarding them to their contacts through SMS. The web application is in charge of manage the contents of the prototype, and the configuration of the parameters to send notifications.

The first chapter introduces the theoretical framework about: technologies, Scrum development methodology and the tools used.

In the second chapter the surveys are introduced also the user stories, and the requirements of the prototype to define its architecture are presented. The design and implementation are shown.

The third chapter presents the results of the tests and the validation surveys carried out. The conclusions and recommendations obtained in the development of this project are presented in the fourth chapter.

Finally, are included: the interviews, the project code of the mobile application, web and server; and the user manual as attachments.

KEYWORDS: Android, earthquake, Firebase, microservices, React, React Native, Redux, SMS

2 INTRODUCCIÓN

Ecuador es un país lleno de bondades por sus características geológicas, topográficas y climáticas, sin embargo, no está exento a la manifestación de eventos que puedan poner en riesgo la vida de su población [1]. De acuerdo con el Instituto Geofísico, entre 2016 y 2017 se registraron 12.049 sismos en todo el territorio nacional [2]. El terremoto de 7.8 grados ocurrido en Pedernales el pasado 16 de abril de 2016, ha sido uno de los más devastadores, el mismo que dejó 671 personas entre fallecidas y desaparecidas [3].

Ecuador está ubicado sobre dos placas tectónicas: Nazca y Sudamericana, por lo cual es propenso a los sismos. Por lo tanto, su población debe aprender a convivir con la alta sismicidad y a su vez es importante que conozcan cómo actuar ante un sismo.

Durante situaciones de emergencia, como los terremotos, las personas quieren conocer el estado de sus familiares lo que causa un uso masivo de llamadas, saturando la red celular. Como alternativa durante emergencias pueden utilizarse los SMS[1] (*Short Message Service*), ya que, al compararlos con los servicios de llamadas y datos estos requieren menos recursos de la red celular para funcionar. Además, compañías de telefonía celular suelen enviar los SMS en un canal de control reservado para operaciones de red, en lugar de uno de los canales designados para el tráfico de voz, por lo que un SMS puede pasar incluso cuando los canales de voz están demasiado sobrecargados [4].

Este Trabajo de Titulación se enfoca en el desarrollo de un prototipo de sistema distribuido, para la notificación del estado de las personas después de un terremoto. El prototipo recopila información sobre la ocurrencia de terremotos mediante el servicio ofrecido por el USGS[2] (*United States Geological Survey*). En caso de tener la información de que un terremoto ocurrió en la zona de Ecuador, se envía un SMS a los usuarios preguntando por su estado. El prototipo utiliza la información recopilada del usuario, tal como: su geolocalización y su respuesta por SMS para notificar de su estado a sus contactos. El usuario también cuenta con una sección de recomendaciones que informa sobre qué hacer antes, durante y después de un terremoto sin necesidad de Internet. Además, el prototipo cuenta con un mapa colaborativo de lugares de importancia como albergues, calles en mal estado y sitios seguros.

---

[1] SMS (*Short Message Service*): es un servicio de mensajes cortos que permite a los teléfonos móviles el envío de mensajes cortos entre ellos.

[2] USGS(*United States Geological Survey*): es una agencia científica de los Estados Unidos controlada por el Centro Nacional de Información Sísmica, que se encarga de detectar la localización y magnitud de terremotos en todo el mundo.

2.1 OBJETIVOS

El objetivo general de este Proyecto Técnico es:

- Desarrollar un prototipo de sistema distribuido basado en notificaciones para terremotos.

Los objetivos específicos del Proyecto Técnico son:

- Analizar el funcionamiento de las herramientas necesarias para el desarrollo de este trabajo de titulación
- Diseñar los componentes que conforman el prototipo
- Implementar de acuerdo con el diseño los componentes del prototipo
- Analizar los resultados de las pruebas realizadas

2.2 ALCANCE

En este Trabajo de Titulación se presenta el desarrollo de un prototipo de sistema distribuido que permita durante un terremoto realizar notificaciones del estado de un usuario. El prototipo cuenta con el módulo de notificaciones mostrado en la Figura 2.1, el cual comunica la aplicación cliente con la aplicación servidor a través de SMS.

1. En caso de existir un terremoto el servidor enviará una notificación de terremoto a través de SMS al cliente.
2. La aplicación servidor esperará un intervalo de tiempo por un SMS con la respuesta del estado del usuario.
3. La aplicación Servidor reenviará la respuesta del usuario a sus contactos registrados en Firebase. En caso de no recibir una respuesta el servidor enviará un SMS a los contactos del usuario con la última posición registrada en Firebase [5].

Como parte del prototipo la aplicación Servidor hace uso de un módem[3] para poder conectarse a la red de telefonía celular.

El prototipo cuenta con un módulo de gestión de datos que usa Internet para el registro y acceso a datos. Estos datos son almacenados en el servicio de Firebase que actúa como base de datos del prototipo.

Una vez obtenida esta información, en el cliente, es posible crear una copia desconectada (*offline*) de los datos almacenados en Firebase. En la Figura 2.2 se muestra un esquema completo del prototipo propuesto.

[3] Módem: dispositivo que convierte señales digitales para poder ser transmitidas a través de redes de telefonía.

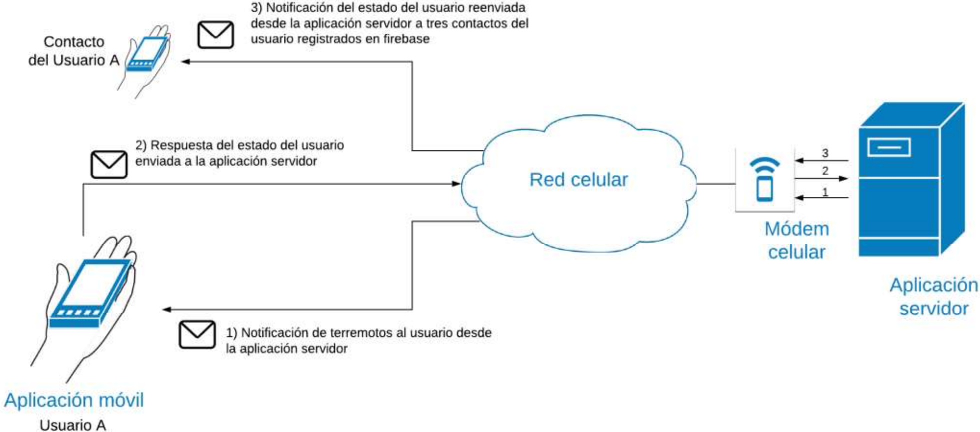

Figura 2.1 Diagrama del módulo de notificaciones

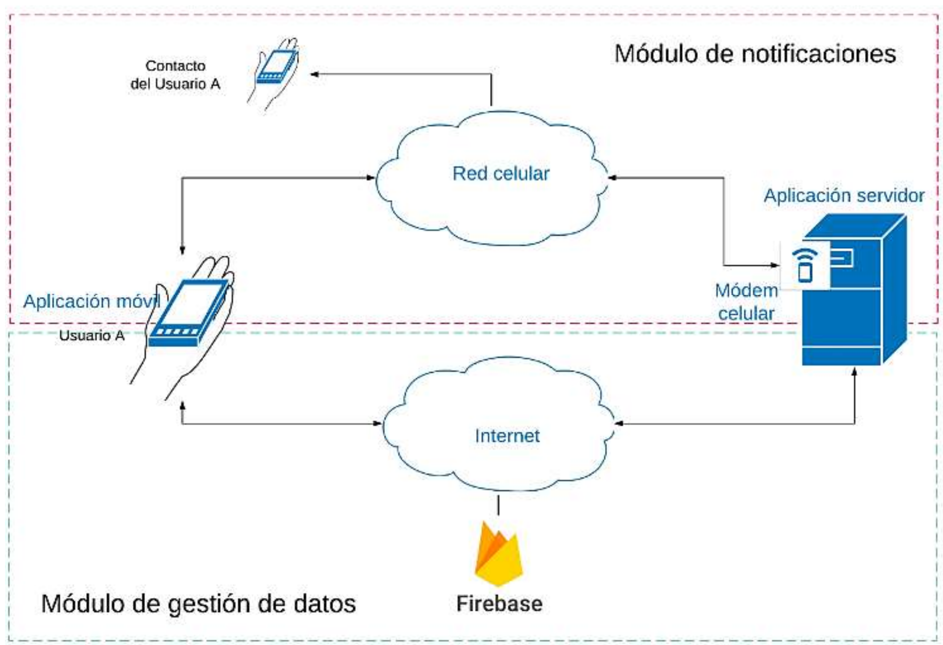

Figura 2.2 Diagrama completo del prototipo

2.3 MARCO TEÓRICO

En esta sección se presentan conceptos clave, adicional a esto, se detalla el conjunto de tecnologías y herramientas usadas para el diseño y desarrollo del prototipo. También se da una corta introducción a la metodología Scrum para el desarrollo de productos software.

2.3.1 TECNOLOGÍAS

2.3.1.1 SMS

Los mensajes de texto o SMS son un servicio de telecomunicaciones que permite el envío de mensajes cortos, que pueden contar con 160 caracteres o menos basados en texto entre teléfonos móviles.

2.3.1.1.1 SMS en situaciones de emergencia

De acuerdo con la Asociación GSM (*Global System for Mobile communications*) el uso de SMS puede proporcionar información oportuna a las comunidades afectadas por un desastre natural y recopilar rápidamente información de estas. Además, es considerado una de las tecnologías más adecuadas para un sistema de comunicación durante emergencias como un terremoto. Estos mensajes pueden enviarse por redes 2G y 3G desde un teléfono celular por lo que se encuentra disponible para un mayor número de usuarios [6].

En Haití durante el terremoto de 2010, se utilizó el servicio de SMS para proporcionar información de la situación a las comunidades, dicha información fue recolectada para mejorar la entrega de ayuda humanitaria [6].

En Ecuador en el terremoto ocurrido el pasado 16 de abril de 2016 en Pedernales, las operadoras de telefonía móvil ofrecieron SMS de manera gratuita a sus clientes en las zonas afectadas para que se comunicarán con sus familiares [7].

2.3.1.1.2 Funcionamiento de envío de SMS

En la Figura 1.3 se presentan los elementos que actúan en el envío de un SMS.

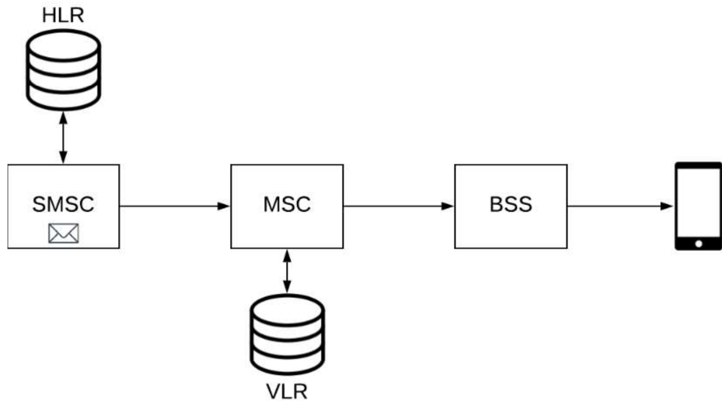

Figura 2.3 Elementos para el envío de SMS

Un SMSC (*Short Message Service Center*) se encarga de almacenar y reenviar los mensajes a otros dispositivos. El SMSC usando el HLR (*Home Location Register*), una base de datos que contiene la información de todos los suscriptores de la red se encarga de buscar el MSC (*Mobile Switching Center*) al cual se encuentra suscrito el destinatario y entregar el SMS a el MSC adecuado.

En cada MSC se tiene un VLR (*Visitor Location Register*) que almacena la información de los suscriptores conectados a ese MSC y ayuda a precisar la ubicación del destinatario. Cuando el MSC tiene la ubicación del destinatario se encarga de cambiar la conexión a la BSS (*Basic Station Subsystem*) correcta.

Finamente, el SMS pasa a un BSS que se encarga de enviarlo hacia la MS (*Mobile Station*) [8]. En caso de que el dispositivo destinatario se encuentre apagado o fuera de alcance, con el uso del SMSC se puede almacenar los SMS y entregarlos cuando se encuentre disponible.

2.3.1.2 Bases de datos NO-SQL

NO-SQL (*Not Only* SQL[4]) es un sistema de gestión de datos que se diferencia del modelo relacional entre entidades y tablas [9]. Las entidades usadas en NO-SQL no se limitan a mantener una estructura, ya que los datos varían para una misma entidad, es decir, no cuentan con un atributo o tienen uno extra. Esta característica es llamada versión de la entidad. Un ejemplo se muestra en la Figura 2.4 con la entidad `Movie`.

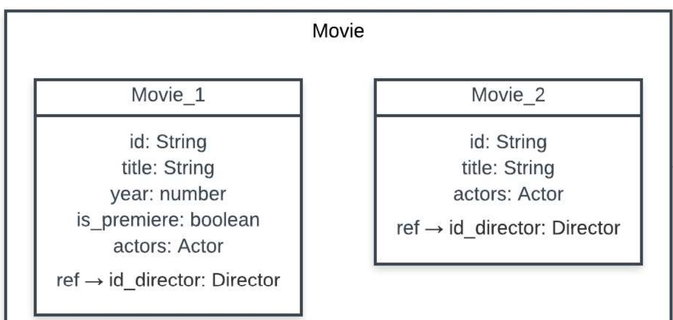

Figura 2.4 Entidad *Movie*

Una entidad puede tener los siguientes tipos de atributos:

- Primitivas: son atributos que representan un único dato que pueden ser del tipo `String`, `Number`, `Boolean`, entre otros que soporte la base de datos
- Referencia: son atributos que relaciona una entidad con otra entidad
- Agregación: son entidades agregadas o embebidas dentro de otra entidad
- Arreglos: datos estructurados dentro de una lista, los cuales pueden ser primitivas, referencias, agregaciones o inclusive otro arreglo

La entidad `Movie` cuenta con los atributos: `id`, `title`, `is_premiere` y `year` como primitivas, `id_director` como referencia a la entidad `Director`, `actors` como un arreglo de agregación de la entidad `Actor`.

En una base de datos NO-SQL los esquemas no suelen ser comunes, sin embargo, suelen ser de utilidad durante el desarrollo. Existen algunas propuestas para la representación

[4] SQL (*Structed Query Language*): es un lenguaje de programación estructurada comúnmente usada para programar, administrar y recuperar información de bases de datos relacionales.

visual de una base de datos NO-SQL. A continuación, se hace una breve descripción de cómo realizar este diagrama.

A causa del versionamiento de las entidades, se opta por modificar el diagrama entidad relación incluyendo todas las versiones de las entidades [9]. La Figura 2.5 muestra el esquema de la base de datos completa de la Figura 2.4

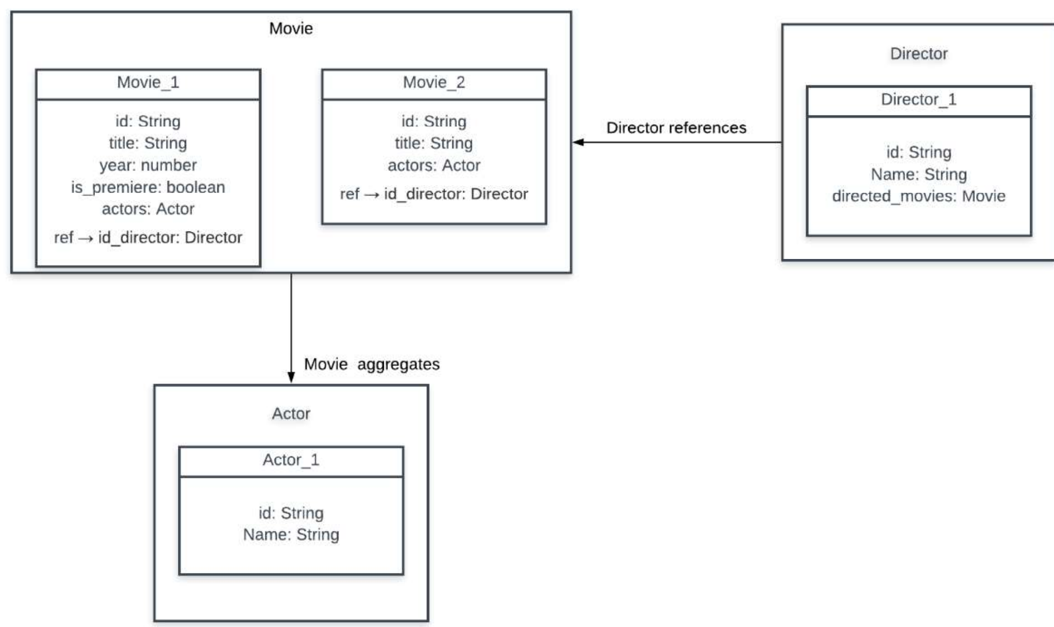

Figura 2.5 Esquema de la base de datos

2.3.1.3 Firebase

Es un servicio BaaS[5] (*Backend-As-A-Service*) ofrecido por Google que proporciona una gama de soluciones para los desarrolladores. El servicio está diseñado para acelerar la integración de funciones basadas en la nube en dispositivos móviles y aplicaciones web. Firebase proporciona muchas características como autenticación, bases de datos en tiempo real, almacenamiento de archivos, *Analytics*[6], *Hosting*[7], entre otros. Firebase cuenta con:

- Firebase *Authentication*: admite la autenticación mediante contraseñas, números de teléfono, proveedores de identidad populares como: Google, Facebook, Twitter, entre otros. Además, permite integrar este servicio con otros propios de Google o con un *backend*[8] personalizado.

[5] BaaS (*Backend-As-A-Service*): es un servicio que proporciona a los desarrolladores de aplicaciones web y móviles una forma de vincular sus aplicaciones al *backend* en la nube.
[6] *Analytics*: es una solución que proporciona estadísticas sobre el uso y desempeño de la aplicación
[7] *Hosting:* es un servicio de Firebase para albergar páginas web.
[8] Backend: es parte de un sistema informático o aplicación a la que no accede directamente el usuario, normalmente responsable del almacenamiento y manipulación de datos.

- Firebase *Realtime Database:* es una base de datos alojada en la nube que almacena los datos en formato JSON. Los clientes de una aplicación que utiliza este servicio reciben actualizaciones de los datos automáticamente.
- Firebase *Cloud Functions*: permite ejecutar de forma automática el código de *backend* en respuesta a eventos activados por las funciones de Firebase y solicitudes HTTPS[9] (*Hypertext Transfer Protocol Secure*). El código se almacena en la nube de Google y se ejecuta en un entorno auto administrado por Firebase, por ende, no es necesario administrar ni escalar servidores propios.
- Firebase *Cloud Storage*: es un servicio de almacenamiento de archivos como imágenes, audio, video y otros tipos de contenido generado por el usuario.
- Firebase *Hosting*: alojamiento seguro y rápido para aplicaciones web. El contenido de la aplicación puede ser dinámico o estático.

Firebase proporciona un SDK[10] (*Software Development Kit*) a los desarrolladores para integrar sus servicios de autenticación, base de datos, entre otros, en sus aplicaciones. Además, el SDK agrega la seguridad a las operaciones de carga y descarga de archivos para las aplicaciones de Firebase, sin importar la calidad de la red.

A continuación, se muestran algunos de los métodos que contiene el SDK y el Código 2.1 muestra un ejemplo de su uso:

- `on`: sirve para obtener datos automáticamente a través de una instantánea del contenido de la ruta de acceso especificada. Es decir que cuando haya un cambio en los datos el método obtendrá el nuevo valor automáticamente sin una consulta generada por el cliente. Este método se muestra de la línea 1 a la 4 donde se obtiene la `Movie` con id 0 de la ruta `movies/0` e imprime por consola sus atributos.
- `once`: sirve para obtener datos una sola vez. A diferencia del método `on`, `once` no recibe actualizaciones automáticamente. El método se ubica de la línea 6 a la 9, donde se obtiene una sola vez el valor de la ruta `movie/1` e imprime por consola sus atributos.
- `set`: sirve para guardar datos en una referencia que se especifique o reemplazarlos si existen. El método se muestra de la línea 11 a la 16 donde se crea una nueva `Movie` con id 2 la ruta `movie/2`.

[9] HTTPS (*Hypertext Transfer Protocol Secure*): es el protocolo de transferencia segura de datos de hipertexto.
[10] SDK (*Software Development Kit*): grupo de herramientas que permiten la programación de aplicaciones móviles.

- `push`: sirve para agregar un nodo nuevo con un id único, este método se muestra en las líneas 19 y 20.
- `update`: sirve para actualizar los valores dentro de una ruta específica. En el Código 2.1 se muestra una función `update` que recibe como parámetros el nombre de un `actor` y titulo de `movie`. En la línea 19 y 20 se hace uso del método `push` para crear un nodo con id único en la ruta `/movies/actor/actors` y se la asigna a la variable `newPostKey`. De la línea 21 a la 24 se crea un objeto donde sus *keys* representan la ruta del nodo y su valor los parámetros a actualizar. Por último, en la línea 25 se envía la variable `updates` actualizar la base de datos en Firebase.
- `remove`: sirve para eliminar datos de una ruta especificada. El método se ubica en la línea 27 la cuál elimina la ruta `/movie/0/actors/0` es decir que elimina al actor con id 0 que pertenece a la `movie` con id 0.
- `signInWithPhoneNumber`: permite la autenticación de un usuario mediante su número de celular, el método se ubica en la línea 29 del Código 2.1

```
1   firebase.database().ref("/movies/0/").on("value", snap => {
2       const values = snap.val()
3       console.log(values)
4   })
5
6   firebase.database().ref("/movies/1/").once("value", snap => {
7       const values = snap.val()
8       console.log(values)
9   })
10
11  firebase.database().ref("/movies/2/").set({
12      title: "Avengers: Endgame",
13      year: 2019,
14      is_premiere: true,
15      actors: [{ id: 10, name: "Rober Downey Jr." }]
16  })
17
18  function update(name, title) {
19      const newPostKey = firebase.database().ref("movies/actor")
20          .child("actors").push().key
21      let updates ={}
22      updates['movies/actor/'+newPostKey+"/id"] = newPostKey
23      updates['movies/actor/'+newPostKey+"/name"]= name
24      updates['movies/0/title']= title
25      return firebase.database().ref().update(updates)
26  }
27  firebase.database().ref("movies/0/actors/0").remove()
28
29  firebase.auth().signInWithPhoneNumber(phoneNumber)
```

Código 2.1 Métodos de ejemplo de Firebase

2.3.1.4 Microservicios

La arquitectura de microservicios se refiere al desarrollo de software donde la aplicación se divide en varios servicios implementables de manera independiente. Estos servicios son modulares, pequeños y cuentan con su propia base de datos. Las bases de datos pueden ser del tipo SQL o NO-SQL y se eligen de acuerdo con las necesidades del proyecto.

Dentro de la arquitectura de microservicios, cada servicio es un componente o proceso de la aplicación que se está desarrollando. Los servicios pueden interactuar entre sí, pero para ello necesitan un mecanismo de comunicación como HTTP usado en REST (*Representational State Transfer*).

2.3.1.4.1 REST

REST es una arquitectura que define un conjunto de principios por los que se pueden diseñar servicios web. Los servicios web basados en esta arquitectura se conocen como: servicios web RESTful.

Un servicio web se identifica mediante su URI (*Uniform Resource Identifier*) y se comunica a través del protocolo HTTP. Responde a métodos como GET, PUT, POST y DELETE, que, de manera sencilla, POST significa crear; GET leer; PUT actualizar y DELETE borrar [10].

Los recursos que se envían entre servidor y cliente pueden tener varios formatos, como: texto plano, XML (*Extensible Markup Language*), JSON, entre otros. A la par se han creado otro tipo de formatos para usos específicos como GeoJSON [11] que se usa para estructuras de datos geográficos.

2.3.1.4.2 Beneficios de la arquitectura de aplicaciones con microservicios

La arquitectura de microservicios ofrece algunos beneficios, como:

- Fácil de mantener: es más fácil para los desarrolladores entender el caso de negocio, de esta forma, se puede empezar el desarrollo más rápido.
- Fácil de escalar: puede escalar de manera fácil creando nuevos servicios independientes.
- Aislamiento a fallas: si falla un servicio no hace que el sistema se caiga.
- Implementación independiente: se puede implementar un microservicio de forma independiente sin afectar a otro microservicio dentro de la arquitectura.

2.3.1.5 ReactJS

Es una librería, desarrollada por Facebook, usada en el *frontend*[11] para la creación de aplicaciones web en JavaScript. ReactJS se basa en el concepto de DOM[12] (*Document Object Model*) usado en los navegadores, que representa los elementos del documento

[11] *Frontend*: es la parte de un sistema informático o aplicación que interactúa con los usuarios.

[12] DOM: es una interfaz de programación para los documentos HTML. Define la estructura de los documentos como un grupo de nodos y objetos estructurados que poseen métodos y características. Así a través del DOM, los programas pueden acceder y modificar el contenido, texto, estructura y estilo de los documentos.

HTML en varios nodos dentro de un árbol. La librería accede a los nodos del DOM y los estructura en un VDOM (*Virtual Document Object Model*) como una copia de este, pero dentro de JavaScript.

ReactJS posee estados internos para cada uno de los nodos del VDOM, cuando estos estados se actualizan ReactJS sabe dónde actualizar el DOM del navegador. En la Figura 2.6 se ilustra como el VDOM calcula los cambios de estado y actualiza el DOM del navegador.

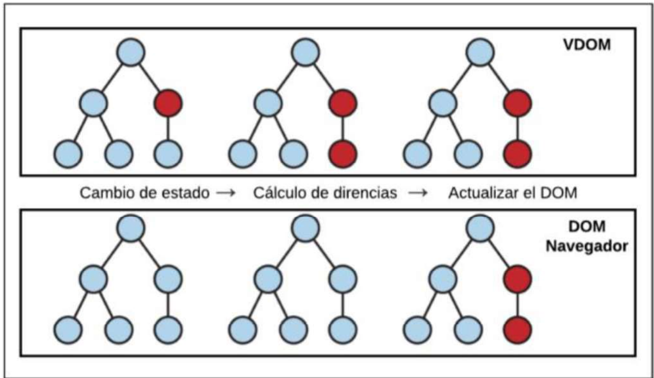

Figura 2.6 VDOM y DOM del navegador, actualización de estados

Dado que ReactJS usa el VDOM no es necesario de un navegador web para su ejecución, lo que permite extender este concepto a otras plataformas como la móvil [12].

2.3.1.6 React Native

React Native es un *framework*[13] basado en ReactJS que permite crear aplicaciones para sistemas iOS y Android. Otra ventaja que da al desarrollador es la de agregar código nativo como: Java, Kotlin o Swift, al proyecto cuando lo necesite.

Las aplicaciones desarrolladas con React Native se ejecutan en tres capas: la capa nativa, la capa de JavaScript y la capa *bridge*. En la Figura 2.7 se muestran las capas mencionadas.

La capa nativa se encarga de manejar los elementos nativos de la UI[14] (*User Interface*) y procesa los gestos o eventos que generan los usuarios. Mientras que la capa de JavaScript es la responsable de ejecutar el código JavaScript, el cual se ocupa de la lógica de la aplicación, funcionalidades y la estructura de la UI. Estas capas no interactúan directamente entre sí, React Native ofrece una capa *bridge* que permite que las capas se comuniquen de forma bidireccional [13].

[13] *Framework*: es una plataforma reutilizable de software que sirve para desarrollar aplicaciones, productos y soluciones basados en un conjunto estandarizado de conceptos, prácticas y criterios.
[14] UI (*User Interface*): también conocida como interfaz de usuario, la cual presenta componentes gráficos dentro de una aplicación.

Figura 2.7 Capas React Native

La capa *bridge* cuenta con las siguientes características:

- Asincrónica: permite la comunicación asíncrona entre las capas y asegura que no se bloqueen entre sí.
- Lotes: envía mensajes de una capa a otra de una manera optimizada.
- Serializable: las capas nunca comparten u operan con los mismos datos, sino que intercambian mensajes serializados.

React Native sigue el siguiente proceso entre capas durante la ejecución de una aplicación. Cuando un evento ocurre, en la capa nativa se recolectan los datos y se envían en un mensaje serializado a la capa *bridge*. La capa *bridge* reenvía el mensaje serializado a la capa JavaScript, la cual es un bucle que procesa el evento e implementa la lógica de la aplicación. Si este evento necesita actualizar el estado de algún componente de la UI o llamar a un método nativo, la capa JavaScript envía al *bridge* mensajes serializados de estas acciones. La capa *bridge* informará con lotes de mensajes serializados a la nativa para que procese los comandos disponibles en los módulos nativos y actualice la UI [14]. Este ciclo se muestra en la Figura 2.8.

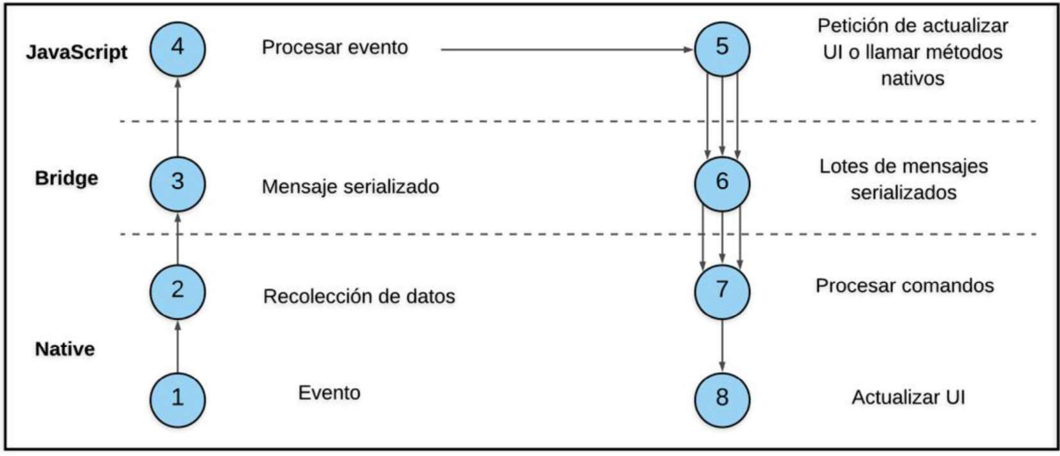

Figura 2.8 Ciclo de ejecución de eventos entre capas de React Native

React Native cuenta con varias librerías complementarias que ayudan a extender su funcionalidad. A continuación, se presentan algunas de las más populares:

- Native Base: es una librería gratuita y *open source* que provee componentes multiplataforma de UI para React Native.
- React Navigation: es una librería que brinda una solución simple y fácil de usar para la navegación, entre pantallas de una aplicación en React Native. React Navigation incluye componentes gráficos para la navegación como *Drawers*[15] y *Tabs*[16]. Además, opciones de personalización en cada pantalla y rutas entre pantallas.
- React Native Map: es una librería que permite incorporar Google Maps a una aplicación construida con React Native.

2.3.1.7 Flux

Es una arquitectura para el manejo del flujo de los datos en una aplicación, particularmente en el *frontend*. Flux define cuatro componentes principales que conforman la arquitectura:

- *Actions*: son métodos que transmiten información al *dispacher.*
- *Store*: actúa como contenedor para el estado de la aplicación para un dominio particular dentro de la aplicación.
- *Dispacher*: recibe *actions* y actúa como el único registro de devoluciones de llamada al *store* dentro de una aplicación.
- *Views*: son la representación de los datos en la interfaz gráfica.

Una característica importante de esta arquitectura es el flujo unidireccional de datos entre componentes. Los datos viajan desde el *view* hasta el *dispacher* por medio de *actions*. El *dispacher* entrega los datos al *store* el cual actualizará el *view* de nuevo en caso de ser necesario [15]. En la Figura 2.9 se muestra el flujo de datos entre los componentes de la arquitectura Flux.

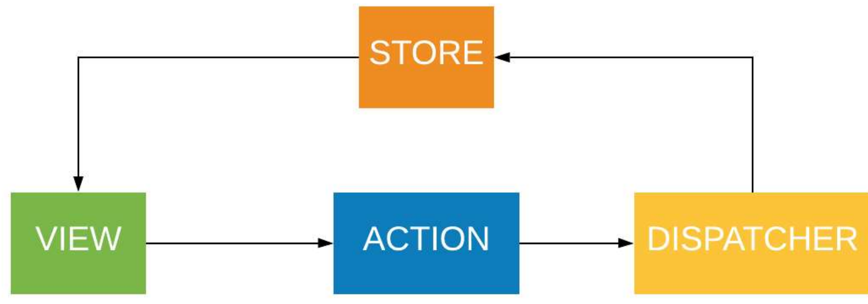

Figura 2.9 Arquitectura Flux

[15] *Drawers*: es un panel de navegación lateral se desliza de izquierda a derecha y contiene los diferentes destinos de navegación dentro de una aplicación.
[16] *Tabs*: es un panel de navegación con un formato horizontal y contiene los diferentes destinos de navegación en forma de pestañas.

2.3.1.7.1 REDUX

Es una librería que implementa la arquitectura Flux, la cual actúa como contenedor de estado previsible para aplicaciones de JavaScript. Redux cuenta con tres principios fundamentales [16]:

1. Una sola fuente de la verdad: el estado (*state)* de la aplicación se guarda en un árbol de objetos JavaScript, el cual se almacena dentro de un único *store*. El *state* tiene la estructura de un árbol de objetos que se puede apreciar en la Figura 1.10.

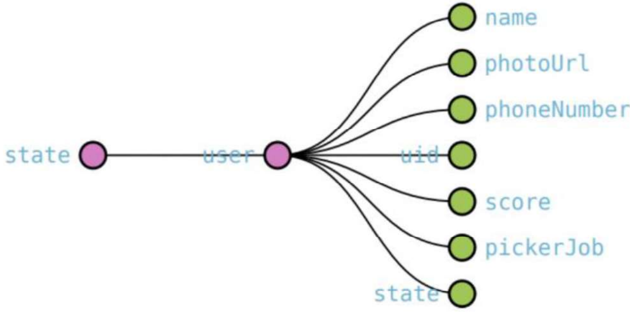

Figura 2.10 Ejemplo de árbol de objetos

2. Estados solo de lectura: esto ayuda a que el *state* no cambie involuntariamente. Existe una sola manera de cambiar el estado y es con la ejecución de un *action* mediante el *dispatch*.

   Un *action* es un método que retorna un objeto simple JavaScript donde se estructura la información que se necesite registrar o almacenar. Este objeto por lo general contiene los atributos `payload` y `type`. El Código 2.2 muestra un ejemplo de un *action*, el cual guarda un usuario y su información se le asigna al `payload`, mientras que `type` indica que tipo de *action* se debe realizar.

```
1  export const setUser = _user => {
2    return {
3      type: user.SET_USER,
4      payload: _user
5    };
6  }
```

Código 2.2 Ejemplo de *action*

3. Los cambios se realizan con funciones puras: un *action* detalla como se actualiza el árbol de objetos en los *reducers*. Los *reducers* son funciones puras que toman el estado anterior y un *action* para retornar el estado actualizado. En el Código 2.3 se muestra un ejemplo de *reducer*, donde se recibe el estado inicial y un *action*, y dependiendo del `type` de la *action* se realizará el cambio en el *state* (línea 5).

```
const reducer = (state = initialState, action) => {
    switch (action.type) {
        case user.SET_USER:
            console.log(action.payload)
            return { ...state, uid: action.payload };
        default:
            return state;
    }
}
```

Código 2.3 Ejemplo de *reducer*

En la Figura 2.11 se presenta el flujo simple que sigue Redux para actualizar el *state* en el *store* y el *view* de la aplicación: un evento en el *view* puede desencadenar un *action*, el cual pasa por un *dispatcher* que ejecuta un *reducer*. El *reducer* actualiza el *state* y lo retorna al *store*. El *view* recibe el nuevo estado y actualiza la UI de ser necesario.

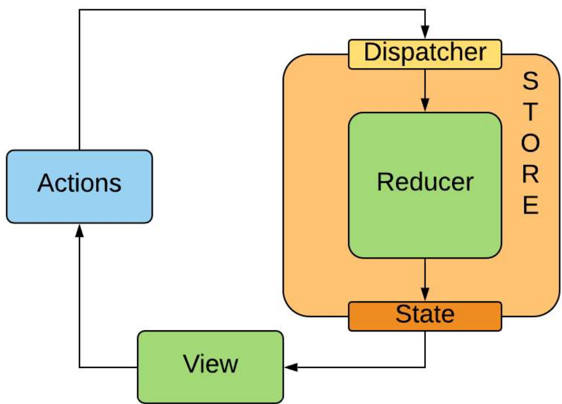

Figura 2.11 Flujo simple de Redux

Redux permite extender sus funcionalidades mediante el uso de *middlewares*. Los *middlewares* son usados para interactuar con las *actions* antes de que el *dispatcher* lo envié al *store*. Por ejemplo: Redux Thunk es un *middleware* que permite realizar operaciones asíncronas. De igual forma se pueden implementar *middlewares* para realizar informe de errores, solicitudes, nuevas *actions*, entre otros [16]. La Figura 2.12 muestra el flujo de Redux que aplica un *middleware* que realiza llamadas a una API.

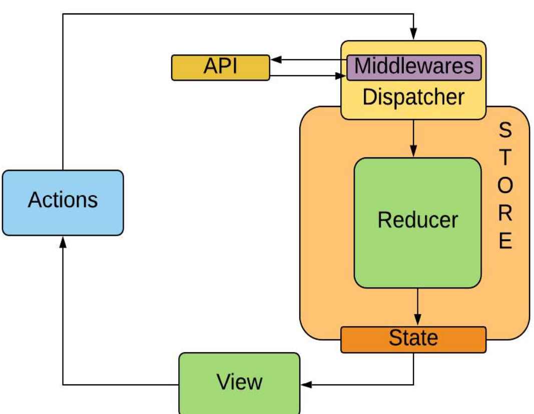

Figura 2.12 Flujo de Redux

Redux en sí es una biblioteca independiente que se puede usar con cualquier *framework* basado en JavaScript incluidos ReactJS y React Native. En el caso de ReactJS y React Native, Redux proporciona la librería denominada `react-redux` que permite a sus componentes de React leer datos y enviar acciones al *store* para actualizar el *state* [17].

2.3.2 METODOLOGÍA SCRUM

Scrum es un marco de trabajo usado para manejar el desarrollo, optimización y mantenimiento de productos. Los usuarios de este marco de trabajo deben conocer los elementos de Scrum. Algunos conceptos de Scrum son:

- *Product Owner*: es el propietario del producto, responsable de dar a conocer los objetivos del proyecto y de garantizar que el *Development Team* entregue los incrementos.
- Scrum *Master*: es el líder del equipo y se encarga de que Scrum sea entendido y aplicado.
- *Development Team*: son los encargados de entregar el incremento del producto y que el producto terminado se pueda poner en producción.
- *Product Backlog*: es una lista de todas las tareas ordenadas y priorizadas que se realizarán durante el desarrollo del proyecto y están visibles para todo el equipo.

Scrum cuenta con algunos de los eventos:

- *Sprint*: es un bloque de tiempo con duración fija, y no puede modificarse. En este bloque se crea un incremento del producto.
- *Sprint Planning Meeting*: reunión de planificación del *Sprint* con todo el equipo.

2.3.2.1 Fases Scrum

Scrum posee procesos fundamentales que se pueden aplicar a proyectos, dichos procesos se dividen en cinco fases: (1) inicio, (2) planeación, (3) implementación, (4) revisión y retrospección, y (5) lanzamiento [18]. La Tabla 2.1 muestra las cuatro fases aplicadas a este proyecto y la descripción de cada uno de sus procesos.

Tabla 2.1 Fases Scrum

Fase	Proceso	Descripción
Inicio	Crear la visión del proyecto	Se revisa el caso de negocio del proyecto para definir la visión que servirá de enfoque durante todo el proyecto.

Fase	Proceso	Descripción
	Asignar los roles Scrum	Se identifica al Scrum *Master, Product Owner* y el *Development Team*.
	Crear el *Product Backlog*	Es una lista de todas las tareas ordenadas y priorizadas que se realizarán durante el desarrollo del proyecto.
Planeación	Crear historias de usuario	Se generan historias de usuario las cuales están diseñadas para garantizar que los requisitos del producto estén claramente descritos.
	Identificar tareas	De acuerdo con las historias de usuario los requisitos se dividen en tareas específicas.
	Crear *Sprint Backlog*	Contiene todas las tareas que se realizarán en un *Sprint.* Estas suelen organizarse en un *Scrumboard*[17], el cual es útil para seguir el trabajo y las actividades que se realizan.
Implementación	Crear entregables	Al concluir con un *Sprint* se genera un entregable.
	Actualizar el *Product Backlog*	Ante cualquier cambio del producto este se discute e incorpora al *Product Backlog*.
Revisión y retrospectiva	Exponer y validar cada *Sprint*	El *Development Team* expone los entregables del *Sprint* al *Product Owner* para asegurar la aprobación y aceptación de este entregable en una reunión.

[17] *Scrumboard*: es un tablero que consta de filas y columnas en las cuales se organizan las distintas tareas previstas a realizar en un *Sprint.*

En la Figura 1.13 se muestra un resumen de las partes más importantes dentro de las fases Scrum.

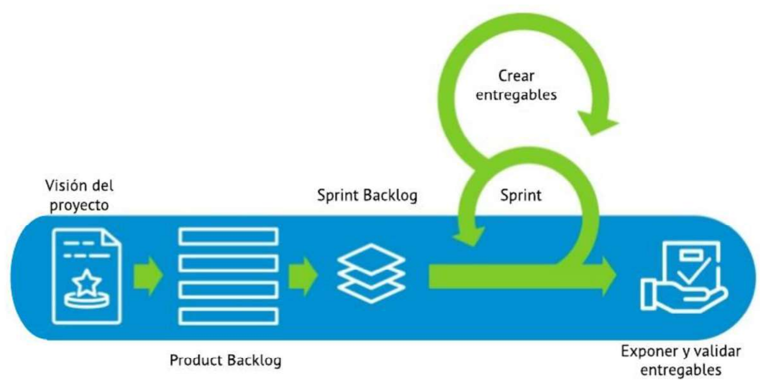

Figura 2.13 Resumen de los procesos Scrum

2.3.3 HERRAMIENTAS DE DESARROLLO

En esta sección se describen las principales herramientas de software y hardware usados para el diseño y desarrollo del prototipo.

2.3.3.1 Node.js

Es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome[18], encargado de suministrar servicios a la aplicación cuando se ejecuta [19]. En el caso de React Native lanza el paquete Metro[19] el cual renderiza los componentes de la aplicación móvil [20].

Para la instalación de esta herramienta se descarga desde su web oficial: https://nodejs.org y se procede con su instalación de manera simple en cualquier sistema operativo. Una captura de la instalación de nodeJS se muestra en la Figura 2.14.

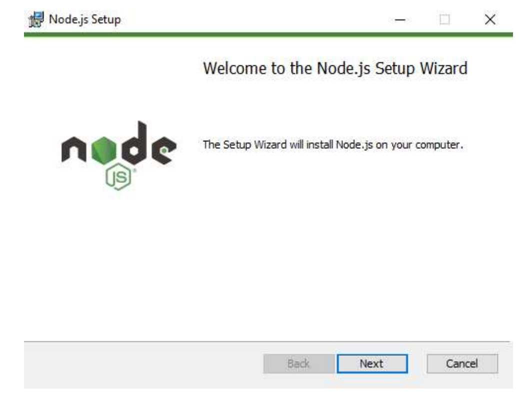

Figura 2.14 Captura de pantalla de instalación de nodeJS

[18] JavaScript V8 de Chrome: es el motor de código abierto de alto rendimiento de JavaScript y WebAssembly de Google, escrito en el lenguaje de programación C ++.
[19] Metro: es un empaquetador de todo el código a un solo archivo JavaScript que se envía al dispositivo móvil.

Node.js cuenta con npm[20] (*Node Package Manager*) como gestor de paquetes, que ayuda a administrar módulos, distribuir paquetes y agregar dependencias a un proyecto. Por ejemplo, el comando `npm install package_name` permite instalar una dependencia dentro un proyecto, mientras que, para desinstalar la misma dependencia se ejecuta el comando `npm uninstall package_name`.

2.3.3.2 React-Native-CLI

Es una herramienta de línea de comandos para interactuar con los proyectos React Native, la cual puede instalarse en Windows, macOs y Linux desde la web en https://facebook.github.io/react-native/docs/getting-started. Permite a los desarrolladores crear proyectos React Native, enlazar dependencias nativas e iniciar la aplicación en un emulador o dispositivo Android conectado, entre otras [21].

El comando: `react-native init nombre_del_proyecto`, permite crear un proyecto nuevo. La Figura 2.15 muestra la estructura de un proyecto, el cual contiene las siguientes carpetas y archivos:

- `android`: es una carpeta que contiene un proyecto Android en el lenguaje de programación Java.
- `ios`: es una carpeta que contiene un proyecto iOS en el lenguaje de programación Swift.
- `App.js`: es un archivo JavaScript que contiene el código de la aplicación.
- `app.json`: es un archivo JSON que contiene información de la aplicación.
- `index.js`: es el archivo de arranque y registro de la aplicación.
- `package.json`: es un archivo JSON que contiene información de los paquetes que se usan dentro del proyecto.

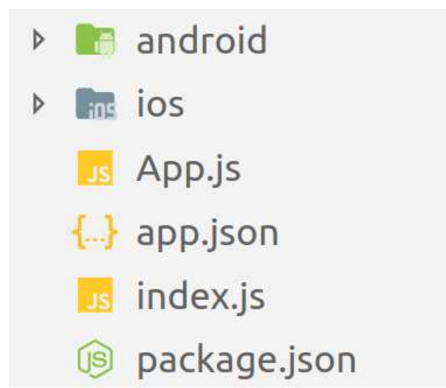

Figura 2.15 Estructura de un proyecto en React Native

[20] Npm (*Node Package Manager*): es el sistema de gestión de paquetes para node.js.

2.3.3.3 Visual Studio Code

Es un editor de código fuente, disponible para Windows, macOS y Linux. Tiene soporte incorporado para JavaScript y Node.js, además tiene un rico ecosistema de extensiones para otros lenguajes de programación. Este editor posee un depurador, una terminal integrada y es muy personalizable. Cuenta con una tienda de extensiones descargables como: *snippets*[21], temas, paquetes de lenguajes de programación, complementos para GitHub[22], entre otras funcionalidades. Una captura de pantalla del editor se muestra en la Figura 2.16 [22].

Figura 2.16 Captura de pantalla de Visual Studio Code

2.3.3.4 Android Studio

Android Studio es el IDE[23] (*Integrated Development Environment*) oficial de la plataforma Android. Está diseñado para acelerar el desarrollo y permite crear aplicaciones para todos los dispositivos que soporta esta plataforma. Incluye herramientas de edición de pantallas y archivos XML, AVD (Android *Virtual Device)* que es un emulador con varias versiones de Android, depuración y pruebas de proyectos mediante el ADB[24] (Android *Debug Bridge*), *snippets* e indentación[25] de código incluido.

Se puede iniciar un AVD desde la línea de comandos, con el comando `emulator -list-avds` que muestra una lista de nombres de los AVD creados, y con el comando `emulator -avd` *`avd_nombre`* se puede iniciar el ADV indicado. La Figura 2.17 muestra un dispositivo Nexus 5x emulado.

[21] *Snippet*: es un término para referirse a partes reusables de código.
[22] GitHub: es una plataforma para alojar proyectos utilizando el sistema de control de versiones Git.
[23] IDE (*Integrated Development Environment*): es un entorno de desarrollo integrado que proporciona herramientas que facilita el desarrollo de software.
[24] ADB (Android *Debug Bridge*): herramienta de línea de comandos que permite la comunicación con un emulador o un dispositivo Android físico conectado.
[25] Indentación: formato del código fuente de un software.

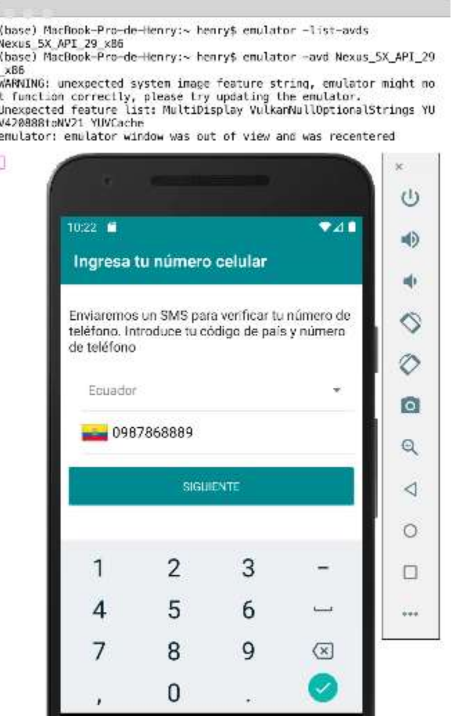

Figura 2.17 Emulador Android, Nexus 5X

2.3.3.5 Firebase Console

Es el *dashboard* o consola que proporciona Google a los desarrolladores para la administración de proyectos en Firebase. El *dashboard* es accesible a través de la web y permite crear proyectos, agregar características específicas a los proyesctos y realizar configuraciones. Algunos de los servicios que se puede configurar son: *Authentication*, *Database*, *Storage*, *Hosting*, *Functions,* entre otros.

En la Figura 2.18 se muestra la captura de pantalla del *dashboard* de Firebase. Agregar un proyecto de Firebase es muy sencillo, simplemente se debe dar *click* sobre añadir proyecto y se abrirá un formulario que solicitará el nombre del proyecto.

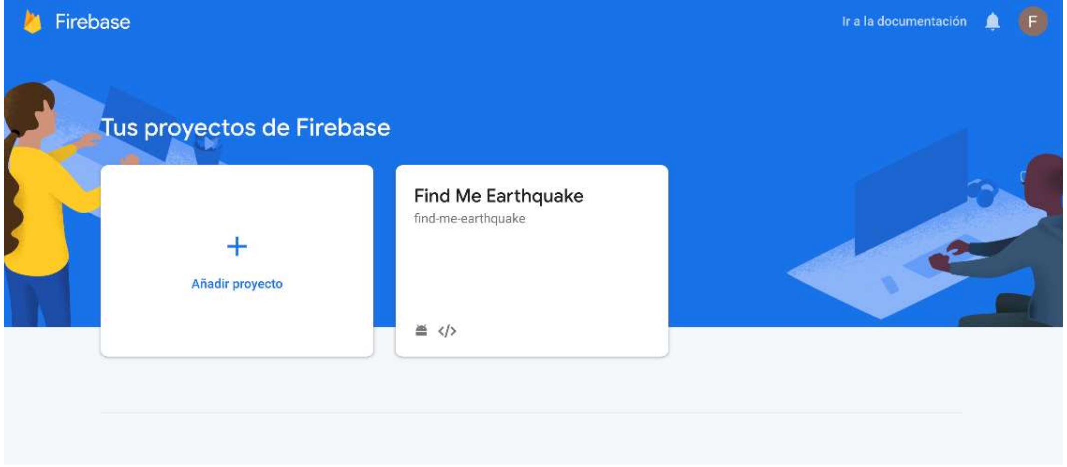

Figura 2.18 Captura de pantalla del *dashboard* de Firebase

Una vez agregado el proyecto se presentará el panel mostrado en la Figura 2.19, desde el cual se puede configurar y añadir los servicios y aplicaciones que estarán vinculados al mismo.

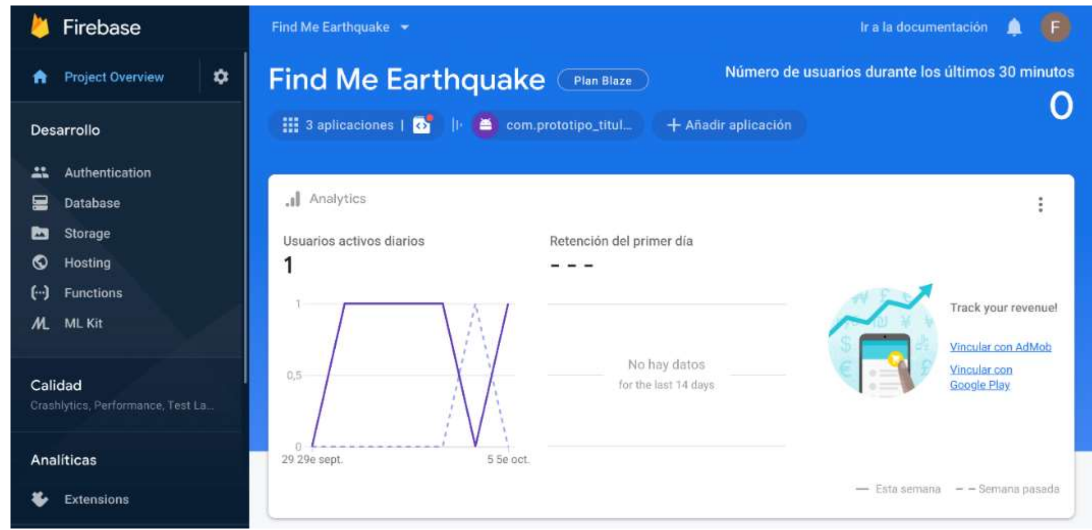

Figura 2.19 Captura de pantalla del *dashboard* de Firebase para el Proyecto *Find Me Earthquake*

2.3.3.6 Lunacy

Es una aplicación para la creación de prototipos de UI de aplicaciones web o móvil. Permite agregar elementos de forma simple como: óvalos, rectángulos, líneas, entre otros, así como dibujar, alinear y unir objetos. Además, admite archivos *sketch*[26], el cual es uno de los formatos más populares en diseño de UI. En la Figura 2.20 se muestra una captura de pantalla de esta herramienta [23].

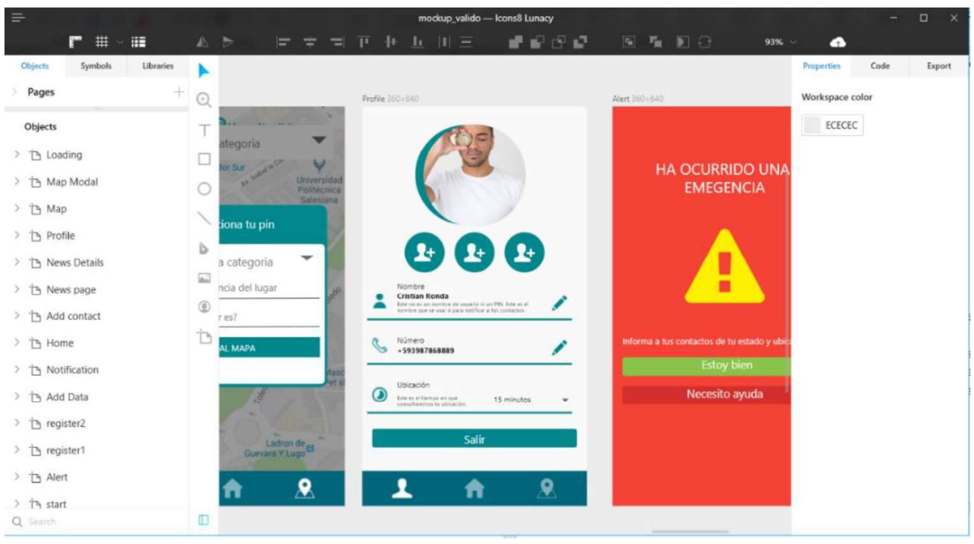

Figura 2.20 Captura de pantalla de la herramienta Lunacy

[26] *sketch*: es una extensión de un archivo de diseño gráfico.

2.3.3.7 Raspberry Pi

Es una placa de circuito impreso que consta de CPU[27], memoria RAM[28], puertos USB[29], entre otros componentes. Haciéndola un computador miniaturizado que puede usar varios sistemas operativos.

Esta plataforma funciona como una interfaz entre hardware y software, es decir, se puede conectar sensores o módulos para obtener información del exterior como sensores de gases, luz, movimiento, módems GSM[30], entre otros. Al ser un computador se puede enviar dicha información por medio de la red local, red celular o Internet. Además, permite implementar y albergar servicios web, correo electrónico, multimedia, IoT[31] y demás servicios que se puedan ejecutar.

La Raspberry Pi Z modelo W cuenta con las siguientes características:

- 1 puerto micro USB
- Puerto mini-HDMI
- Conexión WiFi
- 1 slot para microSD
- 512 MB de memoria RAM

En la Figura 2.21 se muestra una fotografía de la Raspberry Pi Zero modelo W.

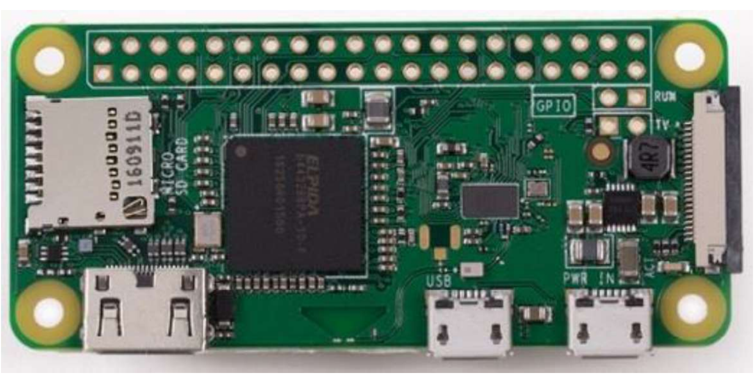

Figura 2.21 Raspberry Pi Zero modelo W

[27] CPU (*Central Processing Unit*): o unidad central de procesamiento es la encargada de realizar todas as tareas de procesamiento y funciones de almacenamiento e información.

[28] RAM (*Random Access Memory*): o memoria de acceso aleatorio se usa como memoria de trabajo en dispositivos como computadores.

[29] USB (*Universal Serial Bus*): es un puerto para conectar dispositivos a un computador.

[30] GSM (*Glocal System for Mobile communications*): o sistema global para las comunicaciones móviles es un sistema estándar para la telefonía móvil digital. Que permite enviar y recibir mensajes por correo electrónico, mensajes de texto y navegar por internet.

[31] IoT (*Internet of Things*): o Internet de las cosas, es la interconexión digital entre dispositivos por medio de Internet.

2.3.3.8 Módem SIM800L

Es un módulo GSM para la Rasberry Pi que permite obtener conexión con la red celular. Las características de este módulo son:

- Interfaz serial UART[32]
- Trabaja en las bandas GSM 850, 900, 1800 y 1900 MHz (redes 2G y 3G)
- Voltaje de Operación: 5V DC

En la Figura 2.22 se muestra una fotografía de este módem con su antena.

Figura 2.22 Módem SIM800L

[32] UART (*Universal Asynchronous Receiver-Transmitter*): o en español Transmisor-Receptor Asíncrono Universal, es un dispositivo que se encarga de manejar la conexión de los puertos y dispositivos serie.

3 METODOLOGÍA

En este Trabajo de Titulación se realizó una investigación aplicada y su desarrollo se basó en la metodología ágil Scrum, la cual ofrece cinco fases, sin embargo, para el presente Trabajo se utilizan solo las cuatro primeras: (1) inicio, (2) planeación, (3) implementación y (4) revisión y retrospectiva.

Siendo la fase inicial donde se crea la visión del proyecto, se asignan los roles Scrum, se crean las historias de usuario en base a las encuestas y se recogen los requerimientos del producto.

Luego, tiene lugar la fase planeación donde se genera la arquitectura del prototipo y casos de uso; se identifican las tareas para crear el *Product Backlog* y el *Sprint Backlog* el cual ayudará a gestionar las tareas del proyecto.

La fase de implementación se dividió en *Sprints*. Se inició con el diseño, implementación y población de los servicios de bases de datos. Después, se generaron los *mockups* y codificó la UI de la aplicación móvil. Luego, se añadieron a la aplicación móvil las funcionalidades de autenticación y registro de usuario; lectura de recomendaciones, mapa colaborativo y notificaciones vía SMS. A continuación, se creó la aplicación servidor donde se implementaron las consultas y notificaciones de terremotos. Por último, se desarrolló la aplicación web para la gestión de contenidos de la aplicación móvil y parámetros del servidor. Cada *Sprint* generó un entregable, que fue evaluado por el *Product Owner* en la fase de revisión y retrospectiva.

El *Product Owner* durante su revisión se encarga de validar los requerimientos del usuario y si existen cambios solicitarlos al equipo. Dicha validación y cambios generan una documentación, la cual se detalla en este documento en el capítulo 3.

Scrum, además, es una metodología flexible, si aparecen cambios o nuevos requerimientos durante el desarrollo del producto pueden ser agregados al *Sprint Backlog* en la fase de implementación.

3.1 FASE INICIAL

3.1.1 VISIÓN DEL PROYECTO

Desarrollar un prototipo que sirva de alternativa de comunicación, a través de SMS, para las personas que desean informar su estado a sus seres queridos inmediatamente después de un terremoto.

3.1.2 ROLES SCRUM

Los roles identificados se muestran en la Tabla 3.1.

Tabla 3.1 Roles Scrum

Rol	Nombre
Scrum *Master*	David Mejía
Product Owner	David Mejía
Development Team	Cristian Ronda, Henry Villavicencio

3.1.3 ENCUESTAS

Para obtener las historias de usuario se generó una encuesta de 8 preguntas. El modelo de la encuesta se encuentra en el ANEXO A y sus resultados se detallan a continuación:

En la Figura 3.1 se muestran los resultados obtenidos de la pregunta 1. Esta pregunta tiene como objetivo conocer que información le resulta más relevante a los usuarios para el registro. Los resultados recogidos muestran que cédula, tipo de sangre y foto de perfil son los datos de interés para el registro del usuario.

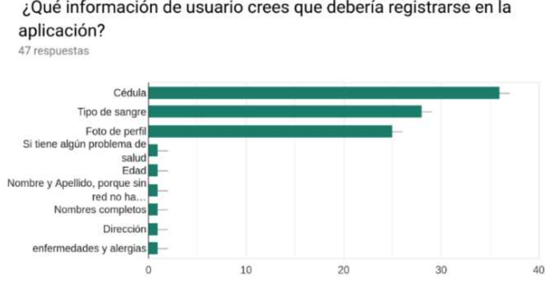

Figura 3.1 Resultados de la pregunta 1

La pregunta 2 tiene como objetivo identificar la cantidad de contactos a los que se les notificará el estado del usuario en caso de terremoto. Los resultados se presentan en la Figura 3.2 y muestran que al 66% de los encuestados les interesa compartir su estado con 3 personas, mientras que, el 38% de los encuestados quieren que sean entre 4 y 5 personas.

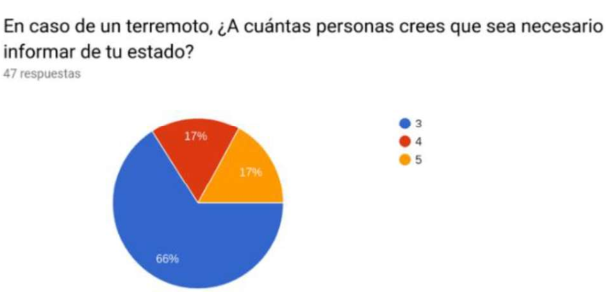

Figura 3.2 Resultados de la pregunta 2

La pregunta 3 tiene como objetivo identificar qué información es de importancia para enviarla a los contactos registrados. Los resultados se presentan en la Figura 3.3 y muestran que el nombre, tipo de sangre y última conexión del sistema son los datos que quieren enviar a los contactos.

Figura 3.3 Resultados de la pregunta 3

La pregunta 4 tiene como objetivo identificar qué tipo de presentación debería tener la posición del usuario, que se enviará en un SMS a sus contactos en caso de un terremoto. Los resultados se presentan en la Figura 3.4 y muestran que, el 63.8% prefiere las coordenadas geográficas y el 36.2% la dirección en la que se encuentre el usuario.

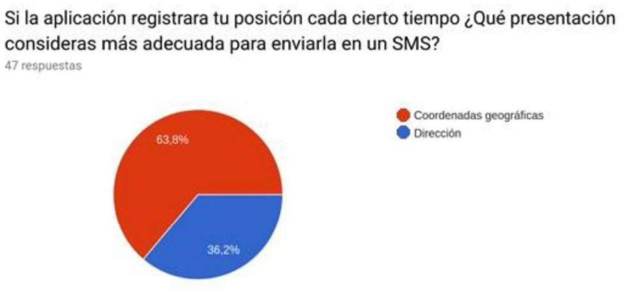

Figura 3.4 Resultados de la pregunta 4

La pregunta 5 y sus resultados se muestran en la Figura 3.5. Esta pregunta tiene como objetivo identificar los lugares que son de importancia para los usuarios luego de un terremoto. Los resultados recogidos muestran que los usuarios tienen como prioridad a sitios seguros, albergues y calles en mal estado.

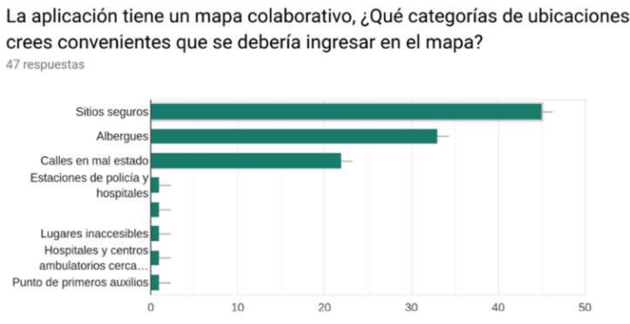

Figura 3.5 Resultados de la pregunta 5

La pregunta 6 tiene como objetivo identificar qué tipo de información le interesa saber al usuario de las ubicaciones que se muestran en el mapa colaborativo. Los resultados se presentan en la Figura 3.6 y muestran que a los usuarios les interesa principalmente el nombre y descripción de la ubicación.

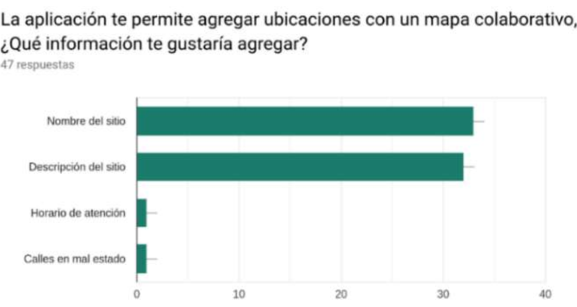

Figura 3.6 Resultados de la pregunta 6

La pregunta 7 tiene como objetivo identificar un tiempo prudencial para esperar una respuesta, de parte del usuario, antes de enviar un SMS de notificación de su estado. Los resultados se presentan en la Figura 3.7 y muestran que un 38.3% cree prudencial un tiempo de 10 o 15 minutos, mientras que, un 23.4% cree que es mejor esperar 30 minutos antes notificar a algún contacto.

Figura 3.7 Resultados de la pregunta 7

La pregunta 8 tiene como objetivo identificar si los usuarios desean que las recomendaciones sean actualizadas. Los resultados se presentan en la Figura 3.8 y muestran que a un 95.7% de los encuestados les gustaría que las recomendaciones sean actualizadas, por otro lado, al 4.3% no le gustaría que las recomendaciones se actualicen.

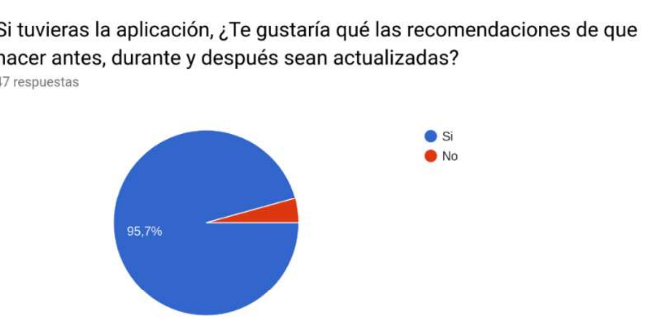

Figura 3.8 Resultados de la pregunta 8

3.1.4 HISTORIAS DE USUARIO

Los resultados de las encuestas permitieron organizar la información y crear las historias de usuario. La Tabla 3.2 presenta estas historias.

Tabla 3.2 Historias de usuario

ID	Título	Descripción
HU_01	Registrar información del usuario	El usuario necesita registrar en el prototipo información de su cedula, tipo de sangre y foto de perfil. Además, de tres contactos a los que se les notificará de su estado en caso de un terremoto.
HU_02	Enviar notificaciones	El usuario necesita que se envíen notificaciones de su estado a través de un SMS en caso de terremoto. El cuál contiene su nombre, tipo de sangre, cedula y su última conexión. En caso de que el prototipo no reciba una respuesta por parte del usuario se enviará la notificación después de un intervalo de tiempo prudente con los datos disponibles.
HU_03	Obtener la ubicación del usuario	El usuario necesita que su ubicación sea enviada en el SMS de notificación.
HU_04	Mostrar lugares de interés en un mapa colaborativo	El usuario desea que se muestren lugares como: albergues, calles en mal estado y sitios seguros.
HU_05	Mostrar información de los lugares de interés	El usuario desea que en los lugares registrados se muestre el nombre y una descripción de este.
HU_06	Actualizar las recomendaciones del prototipo	El usuario quiere que las recomendaciones de que hacer antes, durante y después de un terremoto puedan ser actualizadas.

3.1.5 REQUERIMIENTOS

Con base en las historias de usuario y la visión del proyecto se han identificado los requerimientos del prototipo. El prototipo no contemplaba un administrador que gestione el contenido de este, sin embargo, se lo ha agregado debido a las necesidades del usuario.

3.1.5.1 Requerimientos funcionales

- Autenticar a un usuario por medio de su número celular
- Registrar un usuario con su información personal
- Agregar, cambiar y eliminar tres contactos para notificar el estado del usuario en caso de terremoto
- Cerrar sesión
- Obtener y enviar a Firebase la posición del usuario dentro de un intervalo configurable
- Notificar al usuario vía SMS al ocurrir un terremoto
- Enviar un SMS con el estado del usuario para informar a sus contactos
- Permitir la lectura de manera *offlline* de recomendaciones de que hacer antes, durante y después de un terremoto
- Crear y leer lugares de interés dentro de un mapa colaborativo
- Recibir SMS con el estado del usuario y reenviarlo a sus contactos registrados
- Permitir a un usuario administrador crear, editar y eliminar el contenido de las recomendaciones y lugares de interés
- Permitir a un usuario administrador el cambio de los parámetros para la consulta de terremotos en el servidor

3.1.5.2 Requerimientos no funcionales

- Emplear una arquitectura que use microservicios
- Crear una aplicación Android con el *framework* React Native
- Crear una aplicación web con la librería ReactJS
- Crear una aplicación servidor con nodeJS
- Alojar la base de datos en Firebase
- Enviar SMS a través de un módem GSM
- Consultar terremotos del servicio de USGS

3.2 FASE DE PLANEACIÓN

3.2.1 ARQUITECTURA DEL PROTOTIPO

Se generó la arquitectura del prototipo, la cual se muestra en la Figura 3.9. Esta arquitectura se basa en microservicios y utiliza el BasS de Google Firebase para su implementación.

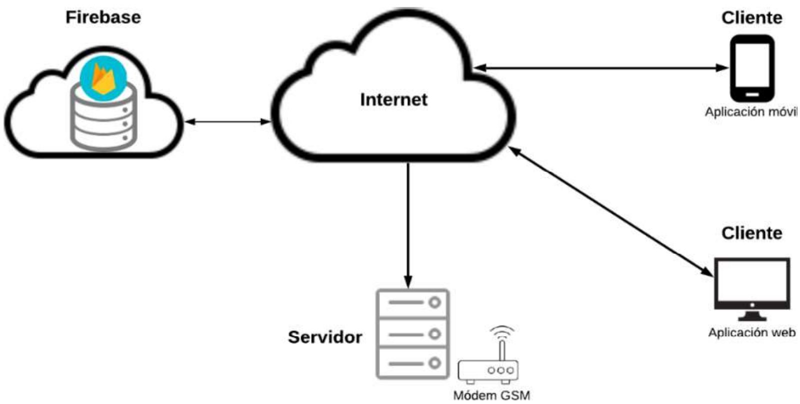

Figura 3.9 Arquitectura del prototipo

El prototipo cuenta con: una aplicación Android y una aplicación web en el lado del cliente. En el lado del servidor se encuentra el módem GSM y el microservicio encargado del envío y recepción de SMS. Además, el prototipo usa Firebase para alberga el servicio de autenticación y los microservicios de base de datos.

Las aplicaciones cliente y servidor se comunicación con el BasS de Firebase, para obtener datos de contenidos y realizar la autenticación, por medio de su API a través de Internet. Mientras que, para la notificación del estado del usuario, el servidor se comunica con el cliente móvil y sus contactos registrados a través de SMS. Esta comunicación se presenta en la Figura 3.10 y se efectúa en tres pasos:

1. El servidor notifica que ocurrió un terremoto a la aplicación móvil
2. La aplicación móvil responde con el estado del usuario
3. El servidor reenvía el estado al contacto del usuario

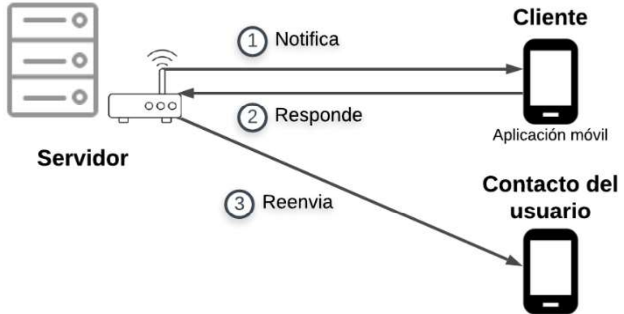

Figura 3.10 Comunicación en caso de terremoto

3.2.2 CASOS DE USO

Los actores que tiene el prototipo son:

- Usuario no registrado
- Usuario registrado
- Administrador

De acuerdo con la arquitectura y los requerimientos obtenidos se han generado los siguientes diagramas de casos de usos, que muestran las interacciones de los diferentes actores identificados.

El usuario no registrado solo puede registrarse. El diagrama se muestra la Figura 3.11.

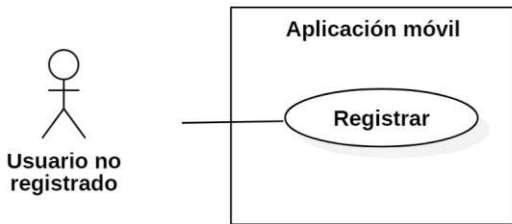

Figura 3.11 Caso de Uso: Usuario no registrado

El usuario registrado puede emplear la mayoría de las funciones del sistema, como: iniciar sesión; ver, agregar y eliminar lugares de interés en un mapa colaborativo; configurar el tiempo de envió de su posición; actualizar datos del perfil; leer recomendaciones; recibir notificaciones, enviar su estado en un SMS durante un terremoto y cerrar sesión. El diagrama se muestra en la Figura 3.12.

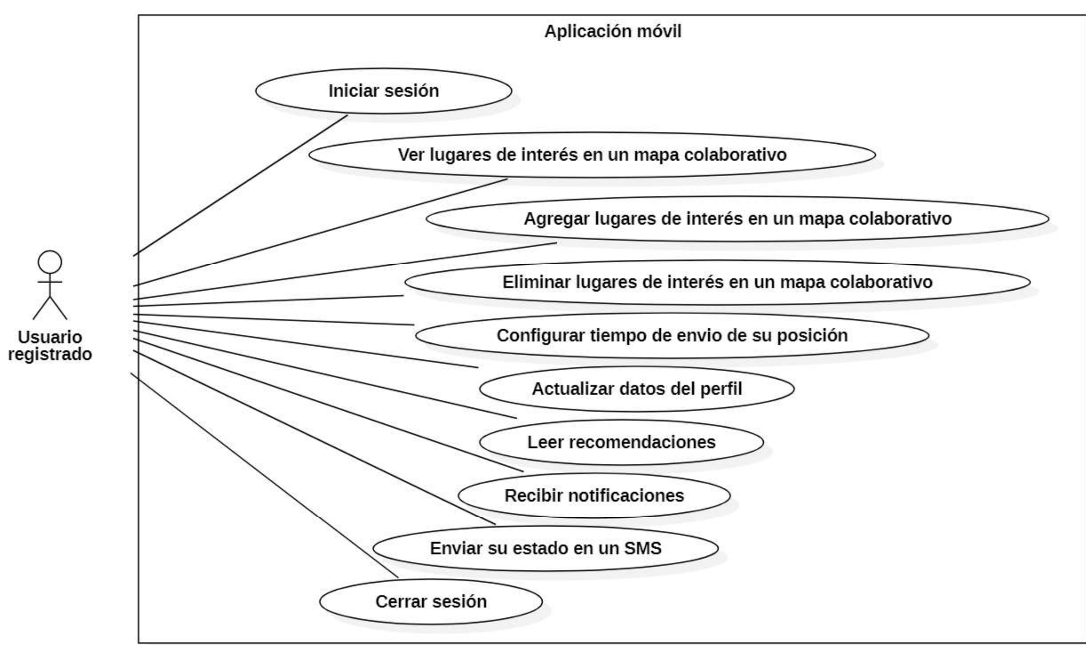

Figura 3.12 Caso de Uso: Usuario registrado

Un administrador puede: iniciar sesión; ver, agregar y eliminar lugares de interés del mapa colaborativo; agregar, leer, actualizar y eliminar recomendaciones; configurar los parámetros del servidor y cerrar sesión. El diagrama se muestra la Figura 3.13.

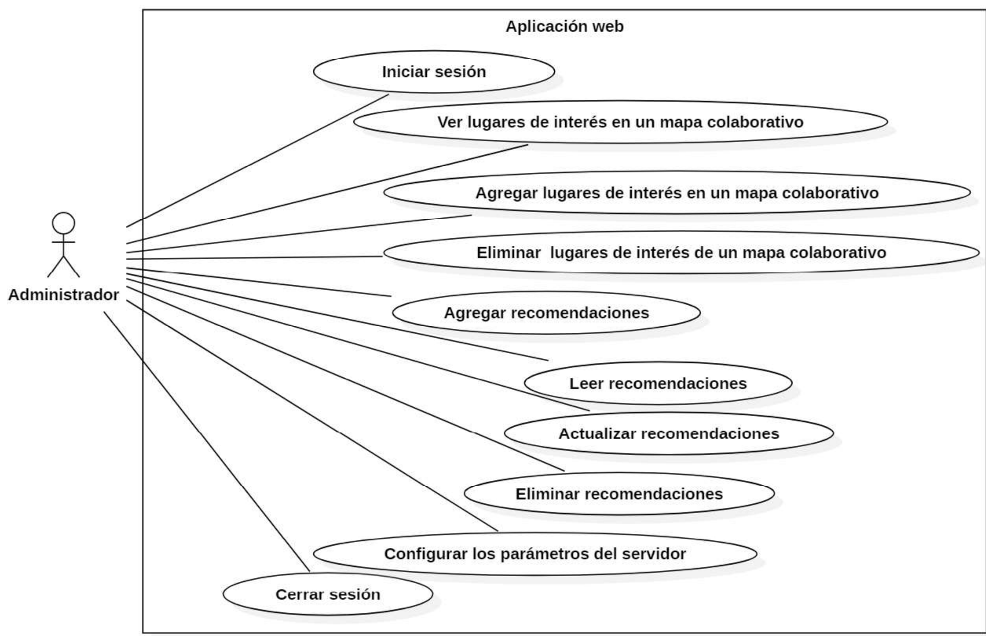

Figura 3.13 Caso de Uso: Administrador

3.2.3 PRODUCT BACKLOG

De acuerdo con los requerimientos del prototipo y teniendo en cuenta su arquitectura se ha generado el *Product Backlog*. El *Product Backlog* muestra las tareas identificadas para cumplir con cada requerimiento (Tabla 3.3).

Tabla 3.3 Product Backlog

Requerimiento	Tareas
1. Levantar la base de datos en Firebase	Modelar la base de datos.
	Codificar la base de datos.
	Alojar la base de datos en Firebase.
2. Autenticar a un usuario por medio de su número celular	Configurar el método de autenticación en Firebase.
	Permitir la autenticación del usuario en la aplicación móvil.
	Codificar la UI para el requerimiento.

Requerimiento	Tareas
3. Registrar un usuario con su información personal	Permitir registrar, leer y actualizar en Firebase el nombre, tipo de sangre, número de cédula y contactos del usuario.
	Configurar Firebase *Storage*.
	Permitir almacenar la foto de perfil del usuario en Firebase *Storage*.
	Enlazar la URI de la fotografía a la referencia del usuario en la base de datos.
	Codificar la UI para el requerimiento.
4. Obtener y enviar a Firebase la posición del usuario dentro de un intervalo configurable	Permitir la lectura de la posición del GPS del dispositivo en un intervalo de tiempo.
	Codificar la funcionalidad para enviar la posición del usuario a Firebase.
	Codificar la UI para el requerimiento.
5. Notificar al usuario vía SMS al ocurrir un terremoto	Consultar terremotos a USGS en el servidor.
	Enviar notificaciones SMS desde el servidor a los usuarios.
	Recibir notificaciones SMS en el cliente.
	Mostrar notificación de terremoto en el cliente.
	Codificar la UI para el requerimiento.
6. Recibir SMS con el estado del usuario y reenviarlo a sus contactos registrados	Enviar un SMS con el estado del usuario al servidor.
	Recibir un SMS con el estado del usuario en el servidor.

Requerimiento	Tareas
	Reenviar el SMS con el estado a los contactos del usuario registrados en el servidor.
	Enviar un SMS desde el servidor a los contactos del usuario con la información disponible de este, en caso de no recibir una respuesta.
7. Permitir la creación, lectura y eliminación de lugares de interés	Agregar a Firebase pines de lugares de interés en el mapa colaborativo.
	Permitir la lectura de pines en el mapa colaborativo en tiempo real en el cliente.
	Permitir la eliminación de pines que el usuario agregó en el mapa colaborativo.
	Codificar la UI para el requerimiento.
8. Permitir la lectura de manera *offline* de recomendaciones de que hacer antes, durante y después de un terremoto	Agregar recomendaciones en la base de datos alojada en Firebase.
	Permitir la persistencia de datos en el cliente.
	Codificar la UI para el requerimiento.
9. Cerrar sesión	Eliminar datos del dispositivo.
	Eliminar datos en Firebase.
	Codificar la UI para el requerimiento.
10. Permitir a un usuario administrador crear, editar y eliminar el contenido en las recomendaciones y sitios de interés	Permitir crear, editar y eliminar recomendaciones de antes, durante y después de terremotos.
	Permitir leer, editar y eliminar pines de lugares de importancia en el mapa colaborativo.

Requerimiento	Tareas
	Generar la UI para el requerimiento.
11. Permitir a un usuario administrador el cambio de los parámetros para la consulta de terremotos en el servidor	Leer los parámetros de configuración en el servidor.
	Codificar la UI para el requerimiento.

3.2.4 SPRINT BACKLOG

Al agrupar los requerimientos del *Product Backlog* se establecen los *Sprint* los cuales generan un entregable. Todos los *Sprint* forman parte del *Sprint Backlog*, este elemento se muestra en la Tabla 3.4 con su nombre y requerimientos.

Tabla 3.4 *Sprint Backlog*

Sprint	Nombre	Requerimiento
Sprint 1	Implementación de la base de datos	1
Sprint 2	Implementación de UI y navegación	2, 3, 4, 5, 7, 8, 9, 10
Sprint 3	Acceso a la cámara y contactos	3
Sprint 4	Implementación del registro y autenticación del usuario	2, 3
Sprint 5	Implementación de la lectura de recomendaciones *offline*	8
Sprint 6	Implementación del mapa colaborativo	7
Sprint 7	Implementación de módulos nativos	4, 5
Sprint 8	Implementación del servidor	5, 6, 11
Sprint 9	Implementación del administrador	10, 11

3.2.4.1 Tareas

Para trabajar en las tareas de los requerimientos se usó el tablero Scrum online de la herramienta Trello[33]. La Figura 3.14 presenta un ejemplo de esta herramienta. Este tablero cuenta con las cuatro columnas denominadas: “por hacer”, que contiene las tareas por

[33] Trello: plataforma de gratuita para la gestión de proyectos con metologías agiles.

realizar; “en progreso”, las tareas que se están realizando; “verificar”, las tareas que están por verificar por parte del *Product Owner* y “hecho”, las tareas ya realizadas.

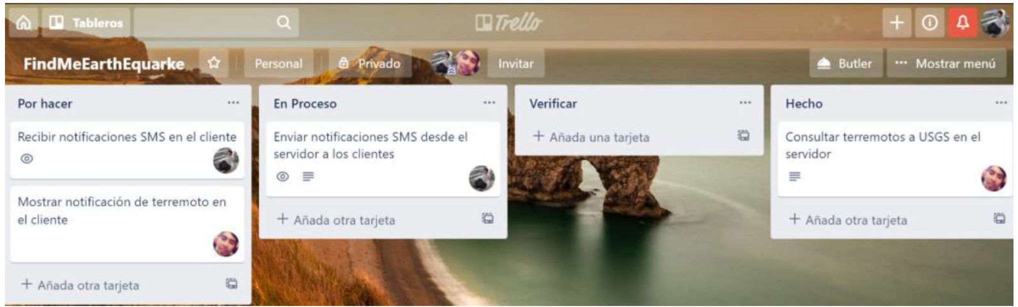

Figura 3.14 Tablero Scrum en Trello

3.2.5 CONVENSIONES DEL CÓDIGO FUENTE

3.2.5.1 Estructura de directorios

Con el objetivo de tener un código escalable y limpio se definió una estructura de directorios para el desarrollo. Las aplicaciones clientes contienen un directorio principal llamado `App`. En este directorio se utilizó la estructura mostrada en la Tabla 3.5.

Tabla 3.5 Estructura general del proyecto

Carpeta	Descripción
`Components`	Contiene los componentes reutilizables de la aplicación tales como: botones, listas, iconos, entre otros.
`Firebase`	Incluye archivos para la conexión con Firebase y los métodos para el uso del SDK de Firebase.
`Images`	Guarda imágenes que se usan en la aplicación.
`Lib`	Es un directorio de misceláneos, el cual contiene el archivo `strings.js` donde se definen los textos de la aplicación.
`Navigation`	Contiene archivos para la navegación entre pantallas de la aplicación.
`Redux`	Abarca los directorios `Actions` y `Reducer` dentro de los cuales se definen las *actions* y *reducers* de la aplicación. También, contiene el archivo `enhancer-redux.js` que alberga los *middlewares* que usa la aplicación. Finalmente contiene el archivo `index.js` en el cual se define el *store* de información de la aplicación.

Carpeta	Descripción
`Styles`	Es un directorio que contiene los estilos de los componentes del proyecto.
`Themes`	Alberga el archivo `colors.js` que contiene los colores de la aplicación.
`Views/Pages`	Contiene directorios de todas las pantallas de la aplicación. Además, un archivo `index.js` donde se exportan las mismas.

Cada pantalla de la aplicación se define dentro de un directorio que contiene dos archivos:

- `Nombre_de_la_pantalla.js`: archivo que contiene la lógica necesaria, es decir, métodos y datos de la interfaz de usuario presentada.
- `Nombre_de_la_pantalla.layout.js`: archivo que define la estructura de la interfaz gráfica de la pantalla.

3.2.5.2 *Backup* y versionamiento del código fuente

Para mantener un control de versionamiento y respaldo del código fuente del prototipo, se levantó un repositorio privado en GitHub y se enlazó el proyecto por medio de comandos al repositorio. En la Figura 3.15 se muestra una captura de pantalla del repositorio creado en GitHub.

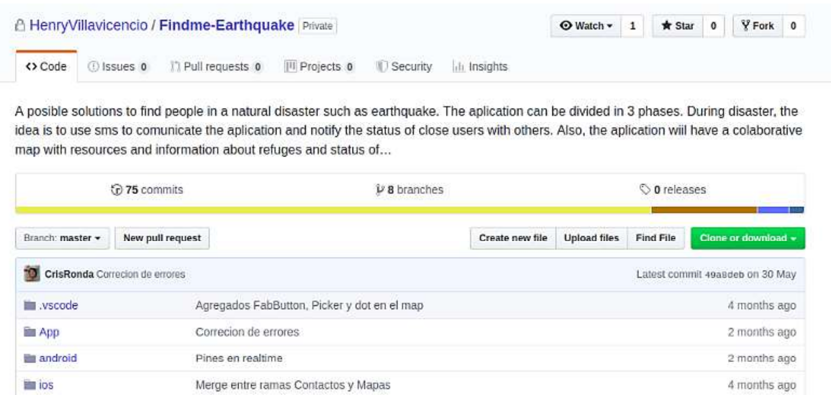

Figura 3.15 Repositorio levantado en GitHub

Para enlazar el proyecto con el repositorio se inició sesión y se agregó el repositorio con el comando mostrado en el Código 3.1 y se instaló el plugin `Git History` en VSCode para visualizar el avance del proyecto.

```
Findme-Earthquake henry$ git remote add origin git@github.com:HenryVillavicencio/Findme-Earthquake.git
```

Código 3.1 Comando de `git` para agregar el repositorio

En la Figura 3.16 se muestra la captura de pantalla de `Git History` de un *push*[34] realizado por el usuario HenryVillavicencio y los archivos que ha modificado.

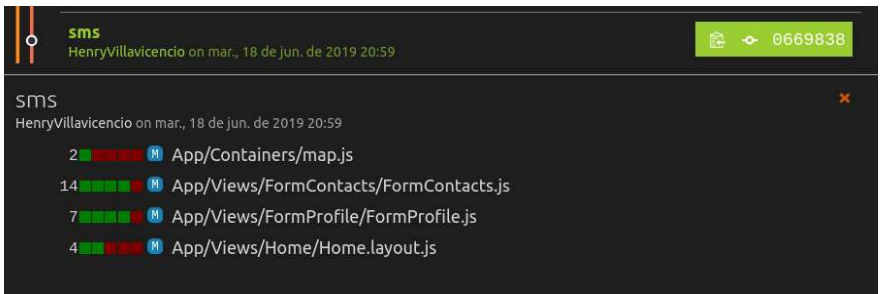

Figura 3.16 Historial de versionamiento

3.3 FASE DE IMPLEMENTACIÓN

3.3.1 SPRINT 1

El prototipo cuenta con información que se obtiene mediante microservicios. Esta información se almacena, en formato JSON, en Firebase. Para el modelado de la base de datos se utilizará un esquema que muestra las entidades y versiones que se alojarán en la base de datos.

De acuerdo con los requerimientos del prototipo y el uso de microservicios se generan las siguientes bases de datos:

- `users`: almacena los datos del usuario
- `map`: alberga los datos del mapa colaborativo
- `feed`: guarda los datos del *feed*[35] recomendaciones
- `server`: almacena las configuraciones del servidor

La base de datos `users`, se muestra en la Figura 3.17, y la posee de las entidades `User`, que almacena la información del usuario; `LastConnection`, que guarda la última posición del usuario y `Contact`, que alberga los contactos del usuario.

La entidad `User` almacena la siguiente información del usuario:

- `id`: identificador del usuario
- `name`: nombre y apellido
- `ci`: número cedula
- `blood`: tipo de sangre

[34] *Push*: comando en Git que envía los cambios al repositorio remoto en GitHub.
[35] *Feed*: es una lista de contenido.

- `phone`: número de teléfono
- `lastConnection`: almacenará la información de la última posición del usuario y mantiene una relación de agregación con la entidad `LastConnection`
- `listContacts`: arreglo de agregación de la entidad `Contact` que almacenará los tres contactos del usuario

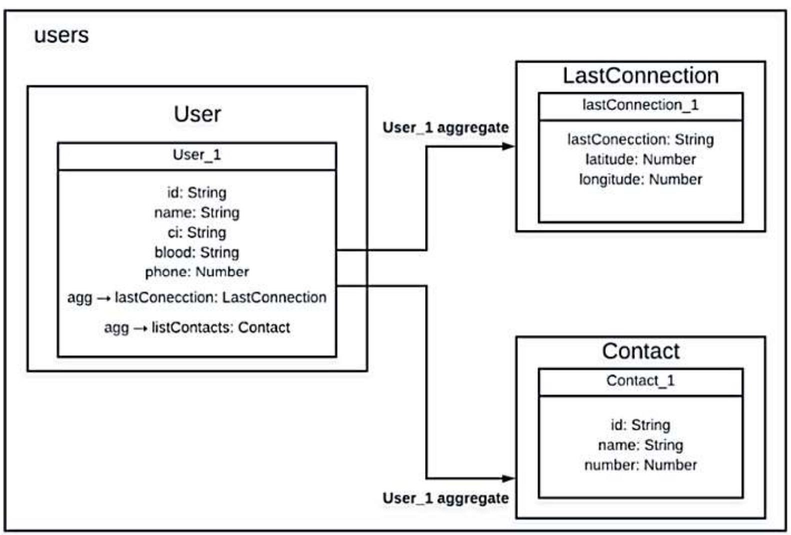

Figura 3.17 Esquema de la base de datos `users`

El esquema de la base de datos `map` se muestra en la Figura 3.18, el cual contiene las entidades `Pin` y `CategoryMap`. La entidad `Pin` tiene dos versiones `Pin_1` que contiene una referencia al `id` del usuario que lo agregó y `Pin_2` es un *pin* creador durante el desarrollo del prototipo y no hace referencia ningún usuario.

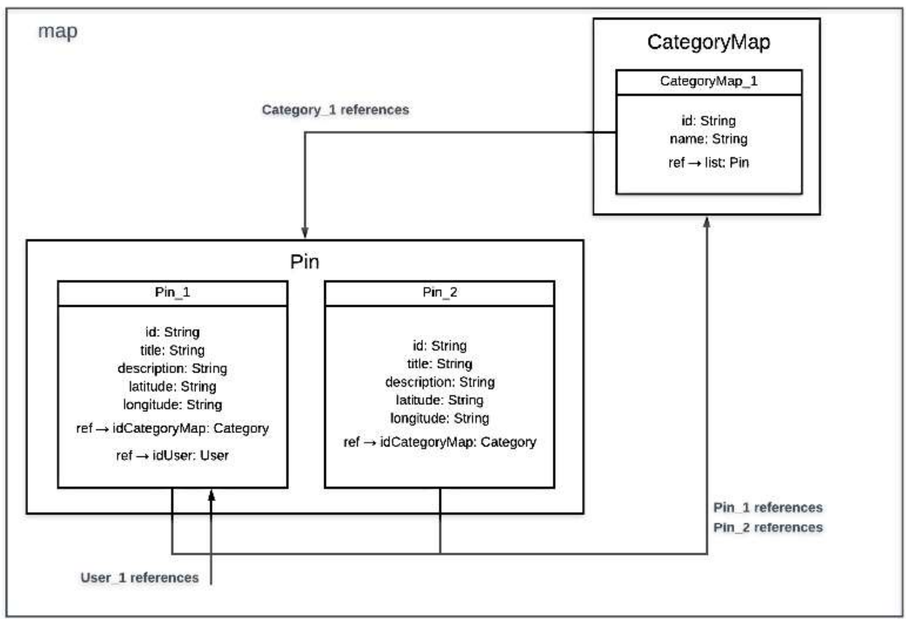

Figura 3.18 Esquema de la base de datos `map`

La entidad `Pin` se relaciona con `CategoryMap` haciendo referencia al `id` de la categoría a la cual el *pin* pertenece. Además, la entidad `CategoryMap` contiene una lista de referencia de los `id` de los *pins* que pertenecen a esa categoría.

`Pin` almacena la siguiente información del lugar de interés, en las dos versiones de la entidad:

- `id`: identificador del *pin*
- `title`: título del *pin*
- `description`: descripción del *pin*
- `latitude` y `longitude`: latitud y longitud de la ubicación del *pin*

El esquema de la base de datos `feed` se muestra en la Figura 3.19 y cuenta con las entidades: `Category`, `Article`, `ContentArticle`, `Content` y `Value`.

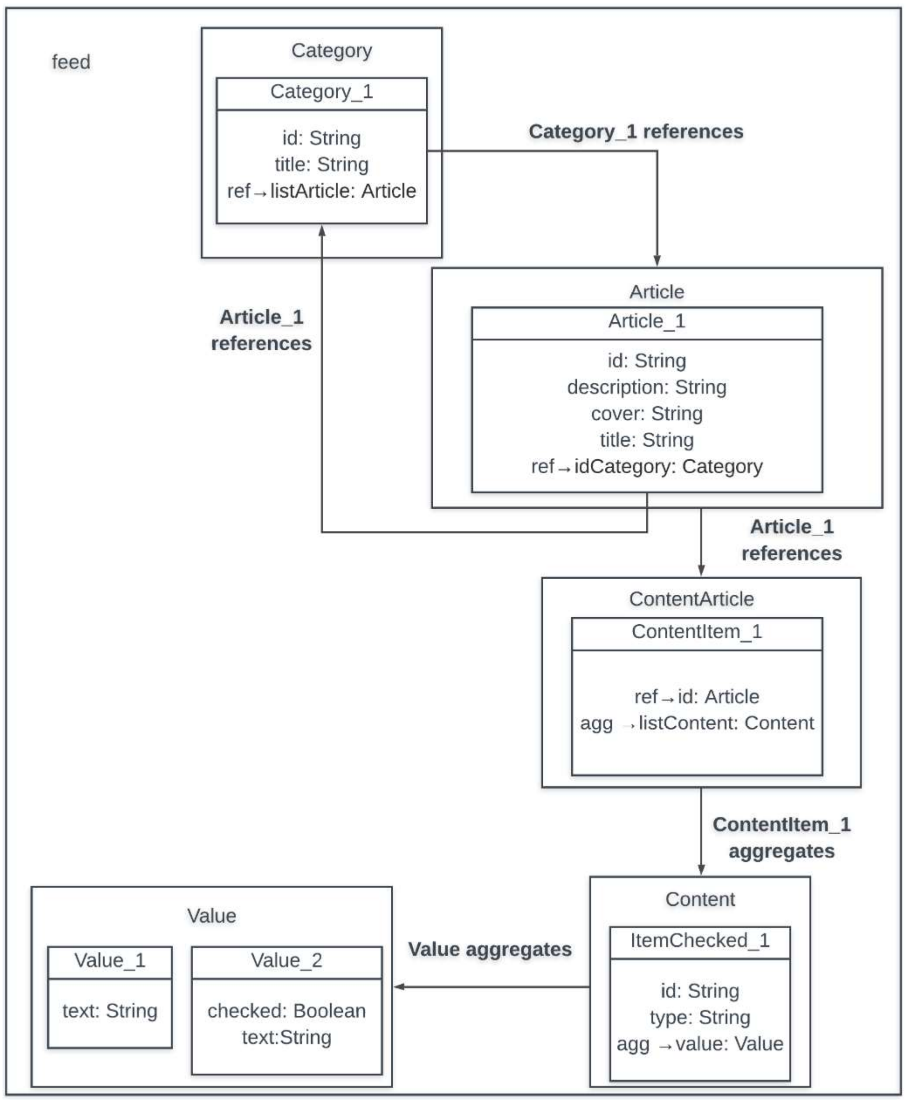

Figura 3.19 Esquema de la base de datos `feed`

La entidad `Article` se relaciona con la entidad `Category` en el nodo `idCategory`, mientras que, la entidad `ContentArticle` referencia el `id` de `Article`. La entidad `ContentArticle` contiene una lista de agregación de la entidad `Content` que a su vez contiene una agregación de la entidad `Value`.

La entidad `Article` almacenará la siguiente información de las recomendaciones:

- `id`: identificador de la recomendación
- `description`: descripción corta de lo que trata la recomendación
- `idCategory`: referencia a la categoría que pertenece
- `title`: título de la recomendación

La entidad `ContentArticle` almacena un arreglo del contenido del artículo y se relaciona con la entidad `Article` mediante el `id` de esta. La entidad `Content` describe el contenido con el nodo `type` que indica el tipo de elemento agregado en un `String` y `value` indica los parámetros de este elemento.

El esquema de la base de datos `server` se muestra en la Figura 3.20 y contiene la entidad `Settings` que almacenará los parámetros de la aplicación servidor:

- `minMagnitude`: representa la magnitud mínima para notificar a los usuarios de un terremoto
- `offsetTime`: intervalo de tiempo en milisegundos previos a la fecha y hora de la consulta de los terremotos
- `requestTime`: intervalo de tiempo entre consultas de terremotos
- `waitTime` tiempo que el servidor espera por una respuesta de los usuarios antes de notificar a sus contactos su estado

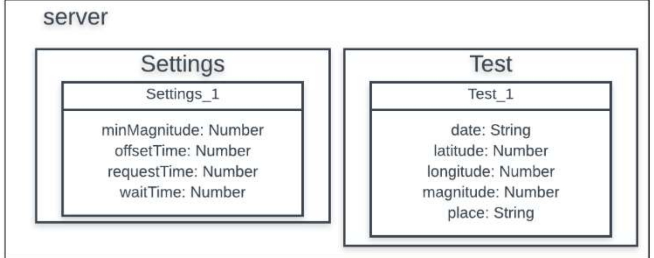

Figura 3.20 Esquema de la base de datos de `server`

Además, se agregó la entidad `Test` para realizar pruebas del prototipo y cuenta con los parámetros:

- `date`: representa la fecha de la prueba

- `latitude` y `longitude`: representan las coordenadas del epicentro del terremoto de la prueba
- `magnitude`: representa la magnitud del terremoto de la prueba
- `place`: representa el lugar de ocurrencia del terremoto de la prueba

3.3.1.1 Implementación

Para la implementación se codificó un objeto JSON para cada una de las bases de datos. En el Código 3.2 se muestra un fragmento escrito en formato JSON para la base de datos `map`, cuyo esquema se puede observar en la Figura 3.18. El nodo `map` es el nodo raíz y contiene las entidades `CategoryMap` y `Pin` con sus respectivos atributos. El ANEXO B contiene los archivos con los objetos JSON para generar las bases de datos.

```
2    "map": {
3      "categoryMap": [
4        {
5          "id": "0",
6          "list": [
7            "0",
8            "1"
9          ],
10         "name": "Centros de acopio"
11       }
12     ],
13     "pin": [
14       {
15         "description": "La Fundación Muchachos Solidarios apoyo a niños.",
16         "id": "0",
17         "idCategory": "0",
18         "idUser": " ",
19         "latitude": -0.223585,
20         "longitude": -78.508768,
21         "title": "La Fundación Muchachos Solidarios "
22       },
```

Código 3.2 Fragmento del objeto JSON `map`

Una vez que se han codificado los objetos JSON, se los importa a Firebase en la base de datos a la que pertenecen. En la Figura 3.21 se muestran los datos importados del Código 3.2.

```
find-me-earthquake-map
- map
  - categoryMap
    - 0
      id: "0"
      - list
        0: "€
        1: "1
        LsuUCMnEojNBuYaehGF: "LsuUCMnEojl
      name: "Alber
  - pin
    - 0
      description: "La Fundación Muchachos Sol
      id: "€
      idCategory: "€
      idUser: "
      latitude: -0.22
      longitude: -78.5(
      title: "La Fundación Mucha
```

Figura 3.21 Objeto JSON `map` importado en Firebase

3.3.1.2 Entregable

Este *Sprint* tiene como entregable las bases de datos alojadas en Firebase. Los datos que contiene cada una de las bases pueden ser consumidos a través de su API. La Figura 3.22 muestra las bases de datos:

- `find-me-earthquake (predeterminado)`: contiene la base de datos `users` y alberga la información de los usuarios
- `find-me-earthquake-article`: contiene la base de datos `feed` con la información de las recomendaciones
- `find-me-earthquake-map`: alberga la base de datos `map` con la información de los lugares de importancia del mapa colaborativo
- `find-me-earthquake-server`: aloja la base de datos `server` con la configuración del servidor y los parámetros para el *test*

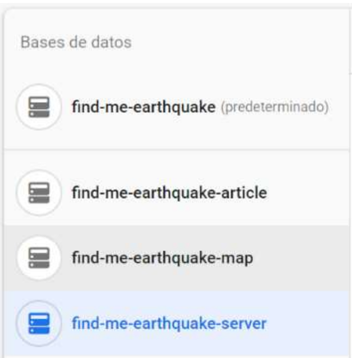

Figura 3.22 Bases de datos en Firebase

3.3.2 SPRINT 2

El prototipo cuenta con una aplicación Android la cual será desarrollada con el *framework* React Native. Esta aplicación deberá brindar una UI para que los usuarios puedan interactuar con el prototipo.

3.3.2.1 Diseño

El diseño de la UI de la aplicación Android se generó con la herramienta Lunacy con base en los componentes de Native Base. A continuación, se presentan los *mockups* de las pantallas de la aplicación y se describe de forma breve la interacción que tendrá el usuario con cada una de ellas.

Los usuarios deberán ingresar su número de celular en el formulario de autenticación que se muestra en la Figura 3.23 (a). Una vez que han ingresado se les enviará un SMS para que se puedan autenticar y continúen con el registro. El *mockup* de la pantalla de verificación se muestra en la Figura 3.23 (b).

Una vez realizada la autenticación, al usuario se le presentará la pantalla de registro de datos mostrada en la Figura 3.23 (c). Esta pantalla consta de un formulario donde deberá ingresar sus datos personales. Finamente, se podrán agregar tres contactos a la interfaz de usuario, en la aplicación móvil, que se muestra en la Figura 3.23 (d) y se culminará con el registro.

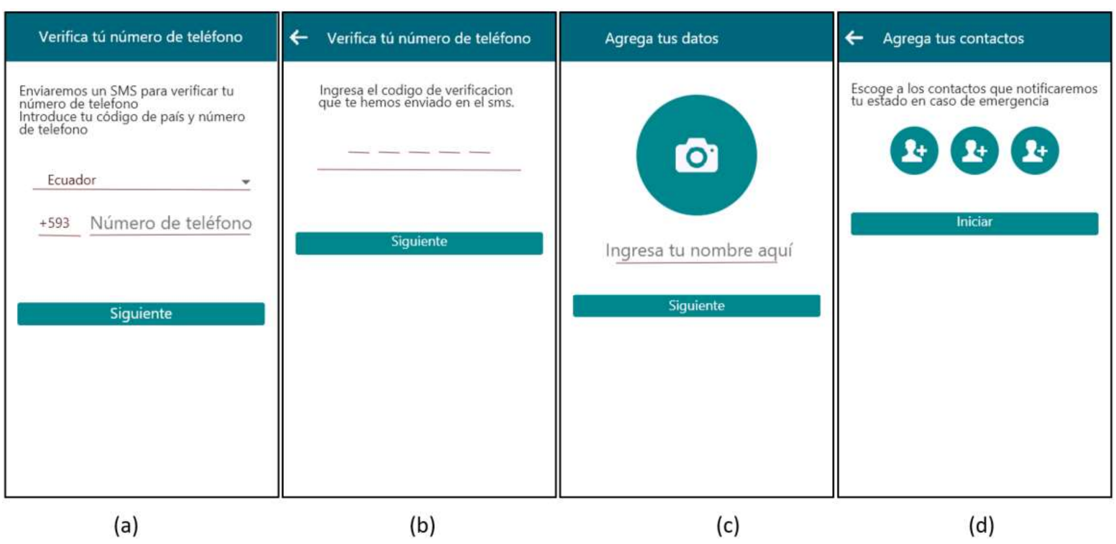

Figura 3.23 *Mockup* de las pantallas de (a) autenticación, (b) verificación, (c) registro de datos del usuario y (d) selección de contacto

La aplicación contará con un *feed* de recomendaciones que consta de tres pantallas que se presentan en la Figura 3.24:

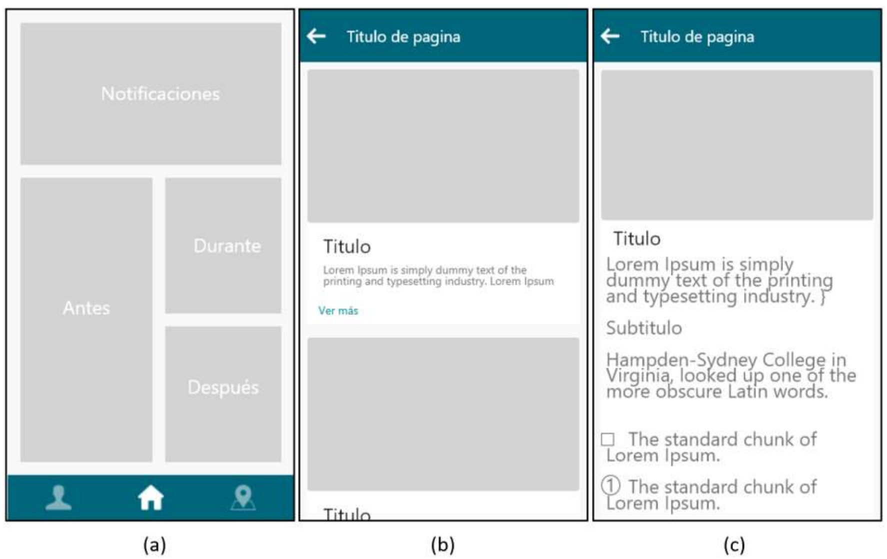

Figura 3.24 *Mockup* de las pantallas (a) *Home*, (b) recomendación por categoría y (c) recomendación

- *Home*: contiene las recomendaciones por categoría y una opción para ver las notificaciones

- Recomendación por categoría: contiene las recomendaciones de una determinada categoría agrupadas en una lista
- Recomendación: es el contenido de una recomendación en específico

La aplicación cuenta con un mapa colaborativo, la interacción del usuario con este se da en dos pantallas:

- Mapa: alberga la pantalla del mapa colaborativo y marcadores de sitios de interés
- ModalMap: contiene la pantalla del modal[36] de registro de un nuevo sitio de interés

Las pantallas mencionadas se muestran en la Figura 3.25.

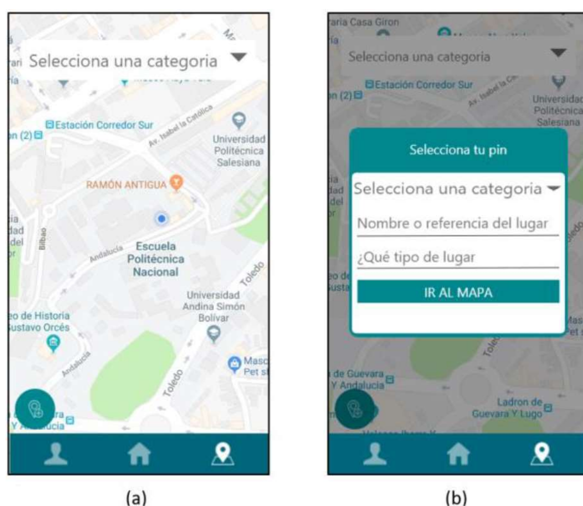

Figura 3.25 *Mockup* de las pantallas (a) Map y (b) ModalMap

La visualización y actualización de los datos del usuario se podrá realizar desde el apartado de perfil mostrado en la Figura 3.26.

Figura 3.26 *Mockup* de la pantalla de perfil

[36] Modal: es una ventana sobre la pantalla que concentra el foco en una acción particular.

3.3.2.2 Implementación

Para la implementación de este *Sprint* se creó un proyecto en React Native y se añadieron las siguientes librerías al archivo `package.json` del proyecto:

- `native-base`: contiene componentes de UI para React Native
- `react-native-flags`: alberga componentes de UI de banderas del mundo
- `react-navigation`: librería para la navegación entre pantallas

3.3.2.2.1 Interfaz gráfica

La implementación de las pantallas se realizó con los componentes gráficos que ofrece Native Base para React Native. El Código 3.3 muestra un ejemplo de cómo se realiza la importación de algunos módulos para el desarrollo de la UI.

```
1   import React from 'react'
2   import {
3       nextTextButton,
4       textNumberRegister,
5       laberPickerNumberRegister,
6       placeholderInputNumberRegister
7   } from '../../Lib/strings'
8   import {
9       Picker,
10      Form,
11      Item,
12      Input,
13  } from 'native-base'
14  import Flag from 'react-native-flags';
15  import { Container, Button, TextHeader } from '../../Components'
```

Código 3.3 Ejemplo de importación de librerías, archivos y variables

En la línea 1 se importa `React` desde la librería `react`, de la línea 2 a la 7 se importan las variables que contienen los textos de la pantalla, de la línea 8 a la 13 se importan los componentes de UI de la librería `native-base`. En la línea 14 se importa un componente `Flag` desde la librería `react-native-flags`. Por último, se importan los componentes de UI personalizados desde la capeta `Components`.

Un ejemplo del componente `Button` de Native Base se muestra en el Código 3.4 y su visualización de la interfaz puede apreciarse en la Figura 3.27.

```
1   <Button
2       text="Home"
3       onPress={() => alert("Presionaste el boton")}
4       child={<Icon name='home' />}
5       style={{ backgroundColor: "blue", textAlign: "center" }} />
```

Código 3.4 Ejemplo de componente en React Native

Figura 3.27 Visualización de la interfaz del botón

Los componentes de React cuentan con propiedades conocidas como `props` que permiten agregarle otros componentes, datos, métodos, estilos, entre otros. El componente `Button` del Código 3.4 cuenta con las siguientes `props`:

- `text`: texto del `Button`, que en este caso es `Home`.
- `child`: componente hijo que se ubicará dentro del `Button`, en este ejemplo se ubica un componente `Icon`.
- `onPress`: método que se ejecutará cuando se presione el componente, en este ejemplo se ejecuta una alerta con el mensaje `"Presionaste el botón"`.
- `styles`: estilos del componente. Este se define en formato JSON y es similar en sintaxis a CSS[37] (*Cascading Style Sheets*).

El Código 3.5 muestra la implementación del *mockup* de la Figura 3.23 (a).

```
16  export default (props) => {
17      const { countryCode, onCountryCodeChange, onChangePhoneNumber, onPress } = props
18      return < Container >
19          <TextHeader
20              text={textNumberRegister}
21          />
22          <Form>
23              <Item>
24                  <Picker
25                      note
26                      mode='dialog'
27                      selectedValue={countryCode}
28                      onValueChange={onCountryCodeChange}
29                  >
30                      <Picker.Item label={laberPickerNumberRegister} value='' />
31                      <Picker.Item label='Ecuador' value='EC' />
32                  </Picker>
33              </Item>
34              <Item >
35                  {countryCode && <Flag code={countryCode} size={32} />}
36                  <Input
37                      maxLength={10}
38                      keyboardType='numeric'
39                      placeholder={placeholderInputNumberRegister}
40                      onChangeText={onChangePhoneNumber}
41                  />
42              </Item>
43          </Form>
44          <Button
45              text={nextTextButton}
46              onPress={onPress}
47          />
48      </Container >;
49  }
```

Código 3.5 Código de la UI para pantalla de autenticación

En la línea 17 se muestran las propiedades que recibe el componente de la interfaz gráfica y se detallan a continuación:

- `countryCode`: es el código de telefonía celular del país.
- `onCountryCodeChange`: es un método que ejecuta cuando el código de país cambia.

[37] CSS (*Cascading Style Sheets*): lenguaje que describe el estilo de un documento HTML.

- `onChangePhoneNumber`: método que se ejecuta cuando el valor dentro del componente `Input` cambia.
- `onPress`: método que se ejecuta cuando se presiona el botón.

Un `Container` (línea 18) envuelve los elementos que componen la pantalla, el cual permite realizar un *scroll*[38] en caso de que los componentes superen el tamaño de la pantalla. `TextHeader` (línea 19) es un componente de texto que muestra un mensaje al usuario en la pantalla.

El formulario `Form` de registro se encuentra definido de la línea 22 a la 43 y está compuesto de los siguientes componentes:

- `Item` especifica los elementos que componen el `Form`. Se define en las líneas 23 y línea 34.
- `Picker` (línea 24) es un componente con una lista desplegable que cuenta con propiedades como: `note` que es un estilo predefinido del componente; `mode` es el modo mediante el cual se presentan las opciones al usuario, en este caso en forma de dialogo`(dialog)`; `selectedValue`, es el valor seleccionado en la lista; `onValueChange`, es el método que se ejecuta cuando se selecciona un nuevo valor de la lista desplegable.
- `Picker.Item` es un ítem de la lista desplegable del componente `Picker`, este se define en la línea 30 y línea 31.

En la línea 35 se revisa si existe el código del país en la variable `countryCode` para mostrar la bandera del país correspondiente, mediante el componente `Flag`.

De forma similar se codificaron las UI restantes de la aplicación móvil, que por motivos de espacio no se muestran en este documento, pero se encuentran en el ANEXO C.

3.3.2.2.2 Navegación entre pantallas

En la aplicación móvil se han identificado 3 tipos diferentes de navegación, definidos por la librería React Navigation:

- *stack* define una pila de pantallas entre las cuales se realiza la navegación.
- *tab* define un conjunto de pestañas entre las que realiza la navegación.
- *switch* muestra una pantalla dependiendo de un caso predefinido y no permite acciones de retroceso.

---

[38] *Scroll:* denominado desplazamiento o deslizar la pantalla

La aplicación cuenta con los siguientes *stacks* de pantallas:

- Registro (`StackRegister`): contiene las pantallas para la autenticación y el registro mostradas en la Figura 3.23
- Recomendaciones (`StackFeed`): abarca las pantallas del *feed* de recomendaciones mostradas en la Figura 3.24
- Mapa (`StackMap`): alberga las pantallas del mapa colaborativo mostradas en la Figura 3.25

El Código 3.6 muestra el *stack* de Registro. En la línea 1 se importa la función `createStackNavigator` desde `react-navigation`. En la línea 2 se importan las pantallas `Auth`, `Verify`, `FormProfile` y `FormContacts` de la carpeta `Views`.

La función `createStackNavigator` recibe un objeto formado por objetos con las pantallas y propiedades que forman el *stack*. Un ejemplo se puede ver de la línea 5 a la 11, donde se crea el objeto `ScreenNumberRegister` con las siguientes propiedades: `screen` que tiene como valor la pantalla `Auth` y el objeto `navigationOptions` que tiene el título y el estilo del encabezado de la pantalla. De forma similar se codificaron los *stacks* restantes.

```
1     import { createStackNavigator } from "react-navigation";
2     import { Auth, Verify, FormProfile, FormContacts } from "../Views";
3     import { header as headerStyle } from "../Styles";
4     const StackRegister = createStackNavigator({
5       ScreenNumberRegister: {
6         screen: Auth,
7         navigationOptions: {
8           title: "Ingresa tu número celular",
9           ...headerStyle
10        }
11      },
12 >    ScreenNumberVerify: {⋯
18      },
19 >    ScreenFormProfile: {⋯
26      },
27 >    ScreenFormContacts: {⋯
33      }
34    });
```

Código 3.6 *Stack* de navegación para la autenticación

La navegación entre pantallas se lo realiza con el método `navigate` definido en la librería `react-navigation`. Este método recibe el nombre del objeto de la pantalla a la cual se quiere navegar. El Código 3.7 muestra un ejemplo del uso de este método.

```
navigation.navigate('ScreenFormProfile')
```

Código 3.7 Método `navigation`

Para la creación del *tab* se utiliza la función `createBottomTabNavigator` de `react-navigation`, la cual recibirá un objeto formado por objetos con las pantallas o *stack* que

conforman las pestañas de navegación. Un ejemplo de este objeto se puede ver en el Código 3.8 de la línea 10 a la 21. El objeto `TabHome` contiene los siguientes parámetros: `screen` que recibe el *stack* de recomendaciones y el objeto `navigationOptions` que contiene la propiedad `tabBarIcon` que define el icono de la pestaña.

```
1   import { createBottomTabNavigator } from "react-navigation";
2   import StackFeed from "./mainStack";
3   import Profile from "./profileStack";
4   import StackMap from "./mapStack";
5   import { Icon } from "native-base";
6   import React from "react";  8K (gzipped: 3.3K)
7   import colors from "../Themes/colors";
8
9   const MainTab = createBottomTabNavigator(
10    {
11      TabHome: {
12        screen: StackFeed,
13        navigationOptions: {
14          tabBarIcon: ({ tintColor }) => (
15            <Icon
16              type="Entypo"
17              name="home"
18              style={{ fontSize: 20, color: tintColor }}
19            />
20          )
21        }
22      },
23 >    TabMap: {…
34      },
35 >    TabProfile: {…
46      }
```

Código 3.8 Fragmento para la navegación en pestañas `BottomTabNavigator`

Para manejar el flujo de navegación se utiliza un `SwitchNavigator`, el cual permite seleccionar entre una pantalla u otra dependiendo del caso definido en una pantalla inicial.

La Figura 3.28 muestra el diagrama de actividades del *switch* de navegación. El flujo empieza cuando el usuario abre la aplicación, la cual mostrará una pantalla de `Loading` por defecto y esta verificará si el usuario se encuentra registrado. Si el usuario está registrado se mostrará la pantalla principal de la aplicación, que en este caso es el *tab* de navegación. Mientras que, si el usuario no está registrado se le mostrará la pantalla de registro. La aplicación luego de procesar el registro mostrará la pantalla principal finalizando el flujo de actividades.

Para la implementación del *switch* de navegación se utilizará la función `createSwitchNavigator`. De la misma forma que en los casos anteriores está función puede recibir *stacks* de navegación y pantallas, sin embargo, de forma adicional recibe un objeto extra en el que se define la pantalla inicial.

El Código 3.9 muestra un ejemplo de un *switch* de navegación. De la línea 7 a la 9 se agrega el *stack* de registro, el *tab* de navegación y la pantalla de `Loading`

respectivamente. En la línea 12 se define la pantalla que se muestra por defecto al iniciar la aplicación, la cual en este caso es la pantalla de `Loading`.

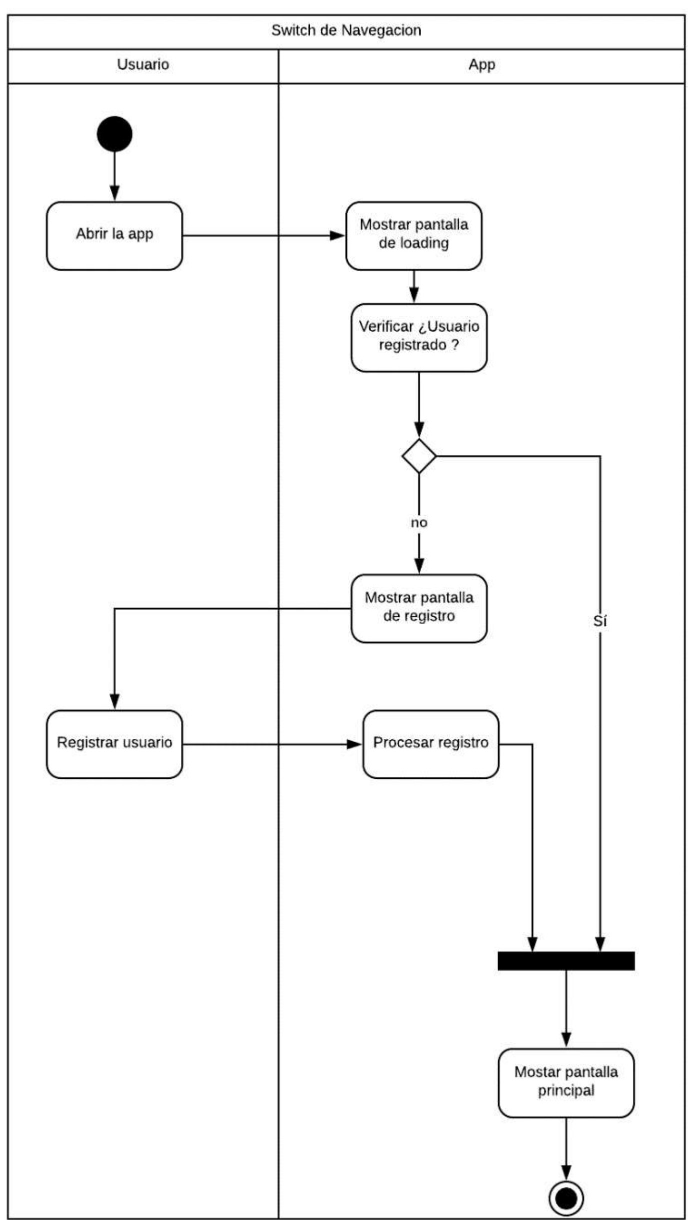

Figura 3.28 Diagrama de actividades *switch* de navegación

```
1  import { createSwitchNavigator } from "react-navigation";
2  import StackRegister from "./stackRegister";
3  import MainTab from "./mainTab";
4  import { Loading } from "../Views";
5  const SwitchNavigator = createSwitchNavigator(
6    {
7      Register: StackRegister,
8      Main: MainTab,
9      Loading,
10   },
11   {
12     initialRouteName: 'Loading',
13   }
14 );
15 export default SwitchNavigator;
```

Código 3.9 Fragmento de código para la navegación en un `SwitchNavigator`

3.3.2.3 Entregable

En este *Sprint* se tiene como entregables los componentes de UI de la aplicación móvil y su respectiva navegación.

3.3.3 SPRINT 3

Este *Sprint* se enfocará en desarrollar las funcionalidades que le permitirán a un usuario escoger su foto de perfil y seleccionar sus contactos desde el dispositivo. Para esto la aplicación necesita hacer uso de ciertas funcionalidades nativas del dispositivo como la cámara, la galería y los contactos. En Android es necesario solicitar ciertos permisos al usuario para que una aplicación pueda acceder a estas funcionalidades.

3.3.3.1 Implementación

Para la implementación de este *Sprint* se añadieron las siguientes librerías al archivo `package.json` del proyecto:

- `react-native-image-picker`: sirve para cargar imágenes desde la galería o la cámara del dispositivo
- `react-native-select-contact`: permite acceder a los contactos del dispositivo

Debido a que las librerías requieren permisos para poder acceder a las funcionalidades de cámara, galería y contactos del dispositivo durante la instalación se debe editar el archivo `AndroidManifest.xml` y agregar los permisos indicados en el Código 3.10.

```
11    <uses-permission android:name="android.permission.CAMERA" />
12    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
13    <uses-permission android:name="android.permission.READ_CONTACTS" />
```

Código 3.10 Fragmento de agregado en `AndroidManifest.xml`

El Código 3.11 muestra el método `request` para solicitar los permisos al usuario con el uso de `PermissionsAndroid` dentro del código JavaScript.

```
PermissionsAndroid.request(
    PermissionsAndroid.PERMISSIONS.READ_CONTACTS
)
```

Código 3.11 Pedido de permiso en Android

La implementación de la funcionalidad de selección de contactos se muestra en el Código 3.12. En la línea 10 se crea la función `_onPress` que recibe como parámetro el `_id` del componente de UI del contacto seleccionado. En la línea 14 se verifica que la aplicación tenga el permiso `PERMISSIONS_READ_CONTACTS`. Verificar los permisos es una acción asíncrona, por lo que es necesario controlarla. Las sentencias `async` y `await` (línea 10 y 13) bloquean la ejecución de la función hasta obtener el resultado o error de la verificación.

Si se tiene el permiso `garanted` (línea 16) para la lectura de los contactos se ejecuta el método `selectContactPhone` (línea 17) de la librería `react-native-select-contact`.

```
10    _onPress = async (_id) => {
11        const { onPress } = this.props;
12
13        const garanted = await PermissionsAndroid.check(
14            PermissionsAndroid.PERMISSIONS.READ_CONTACTS
15        )
16        garanted ?
17            selectContactPhone().then(selection => {
18                if (selection) {
19                    const { contact, selectedPhone } = selection;
20                    const { name } = contact;
21                    const { number } = selectedPhone;
22                    if (onPress) onPress({ id: _id, name, number })
23                }
24            })
25            :
26            PermissionsAndroid.request(
27                PermissionsAndroid.PERMISSIONS.READ_CONTACTS
28            )
29
30    }
```

Código 3.12 Método `_onPress` para la selección de un contacto

Esto inicia la aplicación de contactos y permite seleccionar uno. El método `selectContactPhone` cuenta con una Promesa[39] que retorna la variable `selection` que contiene la información del contacto seleccionado, en caso de que el usuario cancele la selección la variable será nula. Los campos `name` y `number` (línea 20 y 21) se extraen del contacto seleccionado y se envían a un componente externo, a través de la propiedad `onPress` (línea 22) si esta existiera. Si no se tiene el permiso (línea 26) se solicitará el mismo al usuario.

La implementación de la funcionalidad de selección de una foto se muestra en el Código 3.13.

```
17  const _onPress = () => new Promise(done => {
18    ImagePicker.showImagePicker(options, (response) => {
19      if (response.error) Toast.show({ text: imageProfileString.errorToast })
20      else if (response.didCancel) Toast.show({ text: imageProfileString.selectedCancel })
21      else done(response.uri)
22      done('');
23    });
24  })
```

Código 3.13 Función `_onPress` para la selección de imagen

En la línea 17 se crea la función `_onPress` que retorna una Promesa denominada `done`, la cual es una función que se ejecutará en algún momento de la selección de la imagen.

[39] Promesa: es un objeto que representa la terminación o el fracaso eventual de una operación o método asíncrono.

En la línea 18 se usa el método `showImagePicker`, del objeto `ImagePicker` definido en la librería `react-native-image-picker`, para poder acceder a la galería y cámara del dispositivo. En caso de que ocurra algún error (línea 19), se mostrará en la pantalla un mensaje indicando el mismo, por otra parte, si el usuario cancela la selección se muestra un mensaje de que se ha cancelado esta acción (línea 20). En ambos casos la promesa retornará una cadena vacía (línea 22). Por último, si el usuario selecciona una imagen desde su dispositivo con éxito se ejecuta la promesa que devolverá la URI de la imagen seleccionada (línea 21).

Cabe recalcar que la librería maneja la solicitud de los permisos internamente, por lo cual no es necesario solicitarlos en la función `_onPress`.

3.3.3.2 Entregable

En este *Sprint* se tiene como entregables las funcionalidades de seleccionar imagen y contactos en la aplicación móvil.

3.3.4 SPRINT 4

Durante el proceso de registro del usuario, para validar que el dispositivo puede recibir notificaciones por SMS, se autenticará el número de teléfono celular mediante el servicio de Firebase.

El diagrama de secuencia para la autenticación se muestra en la Figura 3.29.

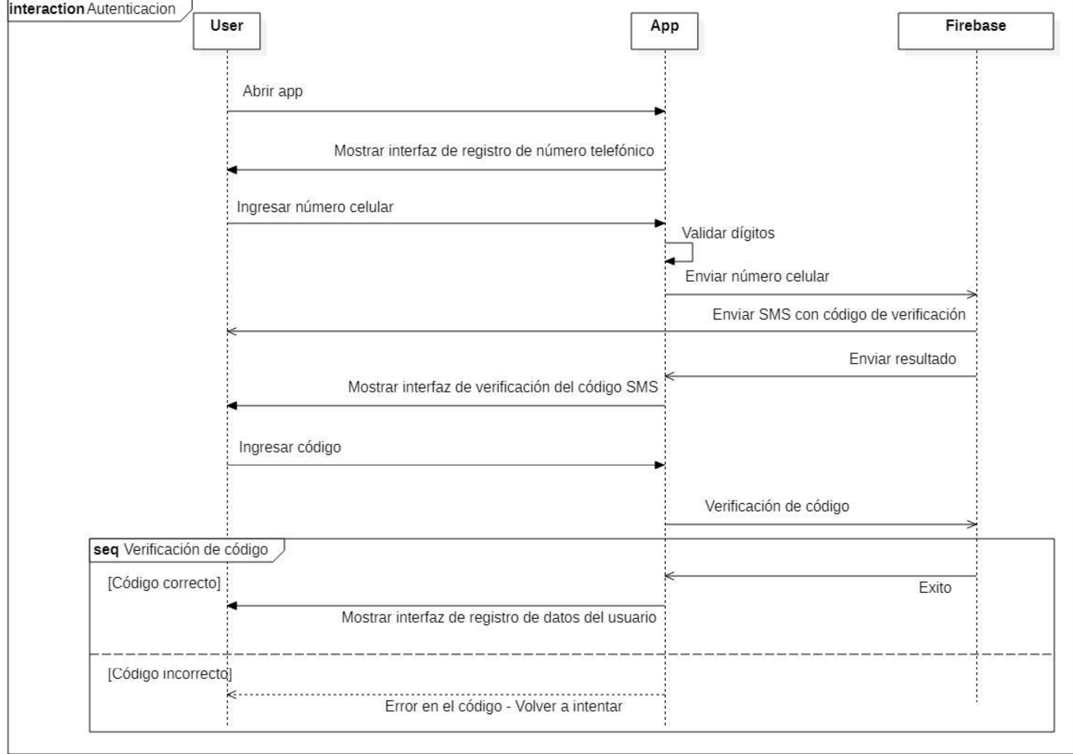

Figura 3.29 Diagrama de Secuencia: Autenticación

El proceso empieza una vez que el usuario abre la aplicación (*app*), quien tendrá que registrarse con su número, el cual se enviará a Firebase a través de un servicio web. Firebase enviará una respuesta desde el servicio y un SMS con un código, mismo que deberá ser ingresado en la aplicación móvil para autenticar al usuario. Si el código ingresado es correcto se continuará con el registro de datos, caso contrario se mostrará un mensaje de error. El usuario tiene la posibilidad de volver a intentar la autenticación en caso de que no haya recibido el código por algún motivo.

3.3.4.1 Implementación

Para la implementación de este *Sprint* se debe añadir la siguiente librería al archivo `package.js` del proyecto:

- `react-native-firebase`: cuenta con el SDK de Firebase para acceder a los servicios de autenticación, bases de datos y almacenamiento de archivos

3.3.4.1.1 Instalación de Firebase en un proyecto React Native

La instalación de esta dependencia requiere la edición de varios archivos en la carpeta `Android` del proyecto.

En el archivo `build.gradle` se agregan las dependencias de Firebase para autenticación, bases de datos, almacenamiento, entre otras necesarias para el uso del API. Las líneas agregadas se muestran en el Código 3.14.

```
173    // Firebase dependencies
174    implementation 'com.google.android.gms:play-services-location:15.0.0'
175    implementation "com.google.android.gms:play-services-base:16.0.1"
176    implementation "com.google.android.gms:play-services-maps:16.0.0"
177    implementation "com.google.firebase:firebase-core:16.0.6"
178    implementation "com.google.firebase:firebase-auth:16.1.0"
179    implementation "com.google.firebase:firebase-firestore:17.1.5"
180    implementation "com.google.firebase:firebase-database:16.1.0"
181    implementation "com.google.firebase:firebase-storage:16.0.4"
```

Código 3.14 Archivo `build.gradle` a nivel de app

En el archivo `build.gradle` a nivel de proyecto Android se agregaron las líneas del Código 3.15 necesarias para usar los servicios de `google` como Firebase.

```
17        dependencies {
18            classpath 'com.android.tools.build:gradle:3.1.4'
19            classpath 'com.google.gms:google-services:4.0.1'
20        }
21    }
22
23    allprojects {
24        repositories {
25            google()
26            mavenLocal()
27            jcenter()
28            maven {
29                url "$rootDir/../node_modules/react-native/android"
30            }
```

Código 3.15 Archivo `build.gradle` a nivel de proyecto Android

En el archivo `MainApplication.java` se debe agregar los paquetes necesarios para acceder a los servicios de Firebase. El Código 3.16 muestra los paquetes agregados para hacer uso del servicio de almacenamiento, autenticación y base de datos de Firebase.

```
34    @Override
35    protected List<ReactPackage> getPackages() {
36      return Arrays.<ReactPackage>asList(
37          new MainReactPackage(),
38            new RNFirebasePackage(),
39            new RNFirebaseAuthPackage(),
40            new RNFirebaseDatabasePackage(),
41            new RNFirebaseStoragePackage(),
```

Código 3.16 Archivo `MainApplication.java`

Para conectar Firebase un proyecto de React Native se deben especifica la clave de acceso al servicio provisto por este. En el Código 3.17 se muestra la configuración de las claves de acceso e inicialización de Firebase en un proyecto React Native. En la línea 1 se importa la librería `react-native-firebase`, de las líneas 2 a 10 se establece la clave provista por Firebase asignada a un objeto `config` y en la línea 11 se inicializa la conexión con Firebase.

En las líneas 12 y 13 se instancian los servicios de autenticación en `auth` y de almacenamiento de archivos en `st`. De la línea 14 a la 16 se instancian los servicios de las bases restantes del proyecto `articleDB`, `mapDB` y `userDB`. Finalmente se exportan las instancias creadas.

```
1   import firebase from 'react-native-firebase'
2   const config = {
3       apiKey: 'AIzaSyCLodGVAitkJe56UnHncS2N1omWJJvlb4o',
4       appId: '1:668510197057:android:31e62b44550e9e76',
5       authDomain: 'find-me-earthquake.firebaseapp.com',
6       databaseURL: 'https://find-me-earthquake.firebaseio.com/',
7       projectId: 'find-me-earthquake',
8       storageBucket: 'find-me-earthquake.appspot.com',
9       messagingSenderId: '668510197057'
10  }
11  const app = firebase.initializeApp(config)
12  const auth = firebase.auth();
13  const st = firebase.storage();
14  const articleDB = app.database('https://find-me-earthquake-article.firebaseio.com')
15  const mapDB = app.database('https://find-me-earthquake-map.firebaseio.com')
16  const userDB = app.database('https://find-me-earthquake.firebaseio.com/')
17  export {
18      auth,
19      st,
20      articleDB,
21      mapDB,
22      userDB
23  }
```

Código 3.17 Inicialización de Firebase en un proyecto React Native

3.3.4.1.2 Métodos de Firebase para la autenticación y registro

Por motivos de espacio se presentan los métodos más representativos que se utilizaron para la aplicación móvil. El Código 3.18 muestra la función de autenticación del usuario.

En la línea 3 se crea la función `authPhoneNumber`, la cual recibe como parámetro un objeto `phoneNumber` que representa el número telefónico del dispositivo. La función retorna el método `signInWithPhoneNumber(phoneNumber)` de `auth` para realizar la autenticación.

```
3   const authPhoneNumber = (phoneNumber) => auth()
4       .signInWithPhoneNumber(phoneNumber)
```

Código 3.18 Función `signInWithPhoneNumber`

El Código 3.19 muestra la función `setUser` que guarda los datos del usuario en Firebase. Esta función recibe como parámetro los datos del usuario como son nombre, número celular, id del usuario, contactos, URI de la imagen de perfil, tipo de sangre y su cedula de identidad (líneas 10 y 11). En la línea 12 se agregan los datos en la ruta `/users/${uid}` con los parámetros de este usuario.

```
10  const setUser = (name, phoneNumber, uid,
11    contacts, photoUrl, blood, ci) => {
12    return userDB
13      .ref(`/users/${uid}`)
14      .set({
15        name,
16        phoneNumber,
17        uid,
18        contacts,
19        photoUrl,
20        blood,
21        ci
22      });
23  };
```

Código 3.19 Función `setUser`

El Código 3.20 muestra la función `uploadImage` que almacenará la foto de perfil en el *storage* de Firebase. En la línea 5 se recibe un objeto `uriImage` que representa la URI de la imagen en el dispositivo y se obtiene el `uid` del usuario en la línea 6. En la línea 7 se retornar el resultado del método `putFile` que almacenará la imagen en la ruta `users/uid` del *storage* de Firebase.

```
5   const uploadImage = (uriImage) => {
6       const { uid } = getUser();
7       return st().ref('users/' + uid).putFile(uriImage)
8   }
```

Código 3.20 Función `uploadImage`

Por último, en el Código 3.21 se muestra la función `getUser` que permite obtener el usuario actual de Firebase en la aplicación. En caso de que no exista este usuario la función retorna un objeto vacío. Esta funcionalidad ayuda a verificar en la pantalla de `Loading` si un usuario se ha registrado o no, permitiendo navegar a la pantalla correspondiente.

```
5    const getUser = () => auth.currentUser;
```

Código 3.21 Función `getUser`

3.3.4.2 ENTREGABLE

En este *Sprint* se tiene como entregables las funcionalidades de registro y autenticación del usuario incorporadas a la aplicación móvil.

3.3.5 SPRINT 5

De acuerdo con los requerimientos, la aplicación cliente debe contar con un *feed* de recomendaciones alojado en Firebase. En caso de un terremoto lo más probable es no tener acceso a Internet, por esta razón, la información de recomendaciones debe estar disponible de manera *offline*.

3.3.5.1 Implementación

Para la implementación de este *Sprint* se añadieron las siguientes librerías al archivo `package.json` del proyecto:

- `react-redux`: permite hacer uso de Redux dentro de React Native.
- `redux`: incluye un contenedor de estado predecible para aplicaciones JavaScript
- `immutable`: incluye estructuras y formatos de datos inmutables.
- `redux-persist`: es un *middleware* para persistir y rehidratar datos en el *store* de Redux.
- `redux-action-buffer`: es un *middleware* para Redux que almacena todas las *actions* en una cola.
- `redux-thunk`: es un *middleware* para Redux que maneja datos de métodos asíncronos.
- `redux-persist-immutable`: es un contenedor para `redux-persist` que proporciona soporte para la librería `immutable`.
- `redux-immutable`: soporta Redux e `immutable`.

Para el uso de Redux en el *feed* de recomendaciones se han creado 4 archivos:

- `actionsTypes.js:` contiene un objeto con constantes que especifican los *actions type* del *reducer*.
- `feedReducer.js:` contiene el *reducer* del *feed* de recomendaciones.
- `feedActions.js:` contiene las *actions* del *feed* de recomendaciones.

- `Redux/index.js:` alberga el *store* de la aplicación.

3.3.5.1.1 Métodos de Firebase para el feed de recomendaciones

El Código 3.22 muestra la obtención de las recomendaciones desde Firebase. En la línea 36 se crea la función `getArticles` que recibe como parámetro un `callback`[40], que se ejecutará cuando se obtengan los datos desde Firebase.

```
36  const getArticles = callback => {
37    articleDB
38      .ref("/feed/")
39      .on("value", snap => {
40        callback(snap.val());
41      });
42  };
```

Código 3.22 Función `getArticles`

3.3.5.1.2 Redux para el feed de recomendaciones

El Código 3.23 muestra un fragmento del archivo `actionsTypes.js.` En la línea 7 se muestra un objeto `feed` con las constantes de los tipos de *actions* que se realizarán en el *reducer*:

- `SET_DATA` (línea 8) *action* que guardará los datos del *feed* de recomendaciones en el *state* del *store*.
- `SET_CATEGORY` (línea 9) *action* que guardará la categoría de las recomendaciones en el *state* del *store*.
- `SET_ARTICLE` (línea 10) *action* que guardará la recomendación seleccionada y la guardará en el *state* del *store*.
- `SET_CHECK` (línea 22) *action* que guardará si un *checkbox*, de la recomendación, esta seleccionado o no y la guardará en el *state* del *store*.

```
7   export const feed = {
8       SET_DATA: 'FEED_SET_DATA',
9       SET_CATEGORY: 'FEED_SET_CATEGORY',
10      SET_ARTICLE: 'FEED_SET_ARTICLE',
11      SET_CHECK: 'FEED_SET_CHECK'
12  }
```

Código 3.23 Fragmento de código del archivo `actionsTypes.js`

El Código 3.24 muestra el contenido del archivo `feedReducer.js`. En la línea 1 se importa el objeto `feed` desde `actionsTypes.` De las líneas 4 a la 8 se declara el estado

[40] *Callback*: en JavaScript un *callback* es una función que se pasa como argumento a otra función.

inicial en el objeto `initialState` y se le aplica `fromJS`, que es una utilidad de `immutable` que convierte los objetos en un mapa inmutable.

En la línea 10 se declara el `reducer` que recibe como parámetro un objeto `state`, que tiene como valor por defecto el objeto `initialState`, y un `action` que especifica como se actualizará el *state*. En la línea 11 se usa un `switch` para encontrar el tipo de *action* que se quiere ejecutar. Cada case del `switch` representa un caso de actualización del `state`. Por ejemplo: En la línea 12 se define el `case` para cuando el `action.type` es igual `SET_DATA`, en este caso se actualizará la `data` de las recomendaciones con el `action.payload`. Los casos restantes del *switch* se implementan de manera similar. Como buena práctica se define un caso `default` el cual regresa el mismo objeto `state` sin ninguna actualización. Por último, se exporta el `reducer` creado.

```
1   import { feed } from '../Actions/actionsType';
2   import { fromJS } from 'immutable';  61.3K (gzipped: 16.7K)
3
4   const initialState = fromJS({
5       data: null,
6       selectedCategory: '',
7       selectedArticleId: ''
8   });
9
10  const reducer = (state = initialState, action) => {
11      switch (action.type) {
12          case feed.SET_DATA:
13              return state.set('data', action.payload);
14          case feed.SET_CATEGORY:
15              return state.set('selectedCategory', action.payload);
16          case feed.SET_ARTICLE:
17              return state.set('selectedArticleId', action.payload);
18          case feed.SET_CHECK:
19              return state.setIn(
20                  [
21                      'data', 'contentArticle',
22                      action.payload.idArticle, action.payload.id,
23                      'value', 'checked'
24                  ],
25                  action.payload.check);
26          default:
27              return state;
28      }
29  }
30  export default reducer;
```

Código 3.24 Archivo `feedReducer.js`

El Código 3.25 muestra un ejemplo de un *action*. En la línea 5 se declara la función `setData` que recibe como parámetro `data`. Está función retorna un objeto JavaScript (línea 6) con dos propiedades `type` y `payload`. El objeto `type` contiene el caso para que

el *reducer* pueda actualizar el *state*, en este ejemplo `feed.SET_DATA`, mientras que, `payload` contiene los datos del *action*, en este ejemplo `data` representa la información de los artículos.

De forma similar se realizan las *actions* para los demás, casos definidos en el archivo `feedReducer.js`.

```
5   export const setData = data => {
6       return {
7           type: feed.SET_DATA,
8           payload: data
9       };
10  }
```

Código 3.25 Función `setData`

El Código 3.26 muestra un ejemplo de *dispatch* para `feedActions`. El ejemplo muestra la función `dispatchGetArticles`, la cual retorna una nueva función que recibe como parámetro la función `dispatch`. En la línea 14 se ejecuta el método `getArticles`, el cual una vez que obtenga una respuesta ejecutará el `dispatch` del *action* `setData` que actualizará el *state* del *store* con el objeto `articles` obtenido desde Firebase.

```
12  export const dispatchGetArticles = () => {
13      return dispatch =>
14          getArticles(articles => dispatch(setData(articles)));
15  }
```

Código 3.26 *Action* `feedAction`

La persistencia de datos se realiza con la ayuda de *middlewares*. El Código 3.27 muestra los *middlewares* agregados en el archivo `enhancer-redux.js`:

```
App > Redux > enhancer-redux.js > ...
1   import { applyMiddleware, compose } from 'redux'  3K (gzipped: 1.2K
2   import { autoRehydrate } from 'redux-persist-immutable'  130.3K (gz
3   import thunkMiddleware from 'redux-thunk'  1.1K (gzipped: 546)
4   import createActionBuffer from 'redux-action-buffer'  1.7K (gzipped
5   import { REHYDRATE } from 'redux-persist/constants'  996 (gzipped:
6
7
8   let middlewares = [thunkMiddleware, createActionBuffer(REHYDRATE)]
9   let enhacers = [autoRehydrate()]
10  export default compose(
11    applyMiddleware(...middlewares),
12    ...enhacers
13  )
```

Código 3.27 Código `enhancer-redux.js`

- `autoRehydrate`: se importa desde la librería `redux-persist-immutable` y sirve para guardar los datos en el dispositivo y agregarlos al *state* cuando la aplicación inicia.

- `thunkMiddleware`: es un *middleware* que maneja los datos de métodos asíncronos.
- `createActionBuffer`: es un *middleware* que almacena todas las *actions* en una cola.
- `REHYDRATE`: es una herramienta de `redux-persist` para obtener todo el *state* guardado en el dispositivo de manera automática cuando se abre la aplicación.

El `store` se muestra en el Código 3.28. En la línea 1 se importa el `reducer`, mientras que en línea 2 desde la librería `redux` se importa `createStore`. En línea 3 se importa la función `Map` con el nombre `map` el cual sirve para crear colecciones en formato clave-valor, y en la línea 8 se crea el `initialState` que guardará el `reducer` con su estado inicial. En la línea 9 se declara y se crea el `store` con el `reducer` y el `initialState` y por último los *middlewares* definidos en el `enhancer-redux` (línea 9). En la línea 10, se agrega la sentencia `persistStore` para hacer uso de la persistencia de datos en la memoria del dispositivo.

```
App > Redux > index.js > ...
1   import reducer from './Reducer'
2   import { createStore } from 'redux'  4.5K (gzipped: 1.8K)
3   import { AsyncStorage } from 'react-native'
4   import { persistStore } from 'redux-persist-immutable'  130.3K (gzi
5   import composedEnhacers from './enhancer-redux'
6   import { Map as map,  fromJS } from 'immutable'  61.3K (gzipped: 16
7
8   const initialState = fromJS(map())
9   const store = createStore(reducer, initialState, composedEnhacers)
10  persistStore(store, { storage: AsyncStorage })
11  export default store;
```

Código 3.28 `store` con persistencia de datos

Para el leer y modificar el estado dentro de un componente de la aplicación se utiliza el método `connect` definido en la librería `react-redux`, para enviar el *state* o parte de este y los *dispatch* de los *actions* al componente como propiedad, en `mapStateToProps` y `mapDispatchToProps` respectivamente. El Código 3.29 muestra un ejemplo del uso de estos. En la línea 30 se conecta a la clase `Category` con `mapStateToProps` y `mapDispatchToProps`. La descripción de la función y del objeto son las siguientes:

- `mapStateToProps`: lee el nodo `feed` y obtiene la categoría seleccionada en `selectedCategory` desde el `state` y lo retorna dentro de un objeto JavaScript (línea 22).
- `mapDispatchToProps`: es el objeto que alberga un *action* para poder utilizarlo dentro de una clase (línea 26).

```
20  const mapStateToProps = (state) => {
21      const idCategory = state.getIn(['feed', 'selectedCategory']);
22      return { idCategory }
23  }
24
25  const mapDispatchToProps = {
26      dispatchSetSelectedArticle: feedActions.dispatchSetSelectedArticle
27  }
28
29
30  export default connect(mapStateToProps, mapDispatchToProps)(Category);
```

Código 3.29 Conexión de una clase con el *store* de Redux

3.3.5.2 Entregable

En este *Sprint* se tiene como entregables la funcionalidad que muestra las recomendaciones de que hacer antes, durante y después de un terremoto, de manera *offline*.

3.3.6 SPRINT 6

De acuerdo con los requerimientos, la aplicación cliente, debe contar con un mapa colaborativo que muestre sitios de interés, donde el usuario también podrá crear y eliminar un sitio. Un usuario solo podrá eliminar un sitio si fue agradado por él.

El diagrama de actividades para añadir/eliminar sitios, representados como *pins*, al mapa se muestra en la Figura 3.30.

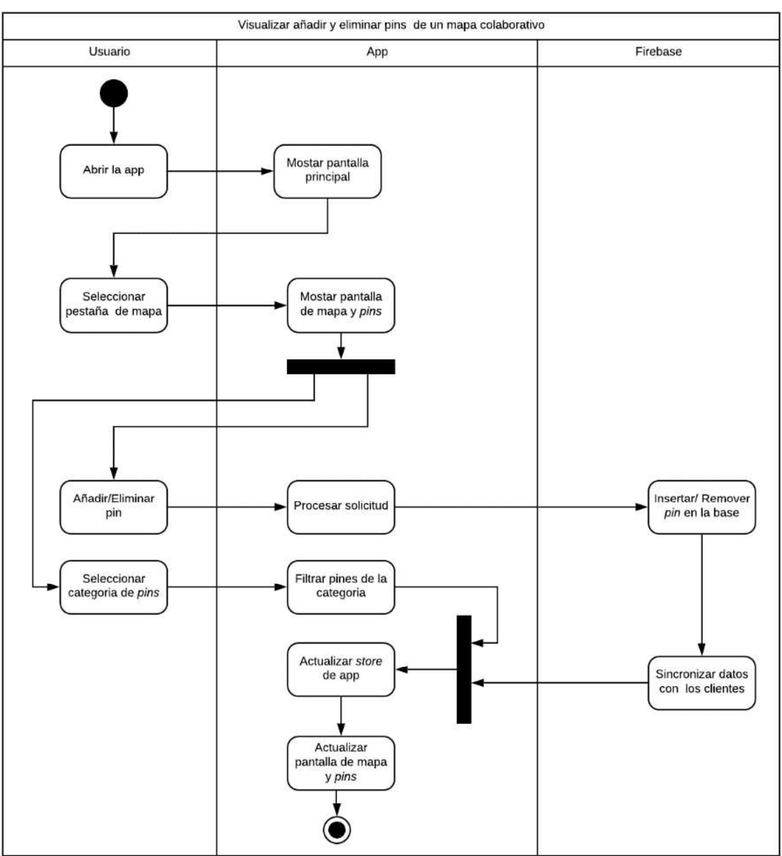

Figura 3.30 Diagrama de actividades: añadir y eliminar *pins* del mapa colaborativo

El usuario deberá abrir la aplicación y seleccionar la pestaña de mapa, donde una vez que la aplicación le muestre la pantalla tendrá dos opciones que puede realizar:

- Añadir/eliminar *pin*: la *app* procesará la solicitud y en Firebase se insertará o eliminará el *pin* de la base de datos.
- Seleccionar categoría de *pins*: la *app* filtrará los *pins* de la categoría.

En ambos casos se actualiza el *store* de la *app* y la pantalla de mapa y *pins*.

3.3.6.1 Implementación

Para la implementación de este *Sprint* se añadieron las siguientes librerías al archivo `package.json` del proyecto:

- `react-native-maps`: contiene componentes como `MapView`, `Maker`, entre otros, que consumen el servicio de Google Maps

3.3.6.2 Instalación de React Native maps

Para la instalación de esta dependencia se requiere la edición de varios archivos del proyecto en la carpeta `android`.

El Código 3.30 muestra las líneas que se han agregado en el archivo `build.gradle`.

```
android > app > build.gradle
157
158    dependencies {
159        implementation project(':react-native-select-contact')
160        implementation project(':react-native-image-picker')
161        implementation project(':react-native-firebase')
162        implementation project(':react-native-gesture-handler')
163        implementation (project(':react-native-maps')){
164            exclude group: 'com.google.android.gsm', module: 'play-services-base'
165            exclude group: 'com.google.android.gsm', module: 'play-services-maps'
166        }
```

Código 3.30 Fragmento del archivo `build.gradle`

En el archivo `MainApplication.java` se han agregado los paquetes necesarios para acceder a los servicios de Google Maps. El Código 3.31 muestra el paquete `MapsPackage` agregado (línea 38) en el método `getPackages`, el cual permite acceder a los métodos de la librería desde el proyecto de React Native.

```
34    @Override
35    protected List<ReactPackage> getPackages() {
36      return Arrays.<ReactPackage>asList(
37          new MainReactPackage(),
38            new MapsPackage(),
```

Código 3.31 Fragmento del archivo `MainApplication.java`

Finalmente se requiere la edición del archivo `AndroidManifest.xml` para especificar el `API_KEY` para el uso del servicio de Google Maps denominado *Maps SDK for Android*, el

cual se puede obtener en la web de Google *Cloud Plataform*. La configuración para el `API_KEY` se muestra en la Figura 3.31 y el Código 3.32 muestra un fragmento del archivo `AndroidManifest.xml`, donde se ha agregado el `API_KEY`.

Figura 3.31 Obtención del `API_KEY` del sitio web Google *Cloud Plataform*

```
27      <meta-data
28    android:name="com.google.android.geo.API_KEY"
29    android:value="AIzaSyDEV4hqJqp69ZrrOAc1WjcTWMnpf422XCk"/>
```

Código 3.32 Fragmento del archivo `AndroidManifest.xml`

Además, se agregaron las líneas del Código 3.33, para obtener acceso a la posición del usuario, en el archivo `AndroidManifest.xml`.

```
6   <uses-permission android:name="android.permission.INTERNET" />
7   <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
8   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
9   <uses-feature android:name="android.hardware.location.gps" />
```

Código 3.33 Permisos para obtener la posición del usuario

3.3.6.2.1 Componentes del mapa

La librería `react-native-maps` cuenta con componentes para hacer uso del mapa de Google, para la implementación de este *Sprint* se utilizaron los componentes `Marker`, el cual muestra un *pin* sobre una ubicación en el mapa y `MapView`, es el mapa como tal.

El Código 3.34 muestra el componente `Marker`, considerado como *pin*, que cuenta con las propiedades:

- `coordinate`: propiedad que representa la latitud y longitud del *pin* (línea 27)
- `title`: título del marcador que se mostrará cuando se presione sobre el *pin* (línea 28)

- `pinColor`: color del *pin* (línea 29)
- `description`: descripción del sitio que se mostrará cuando se presione sobre el *pin* (línea 30)
- `onCalloutPress`: método que se ejecutará cuando se presione sobre la descripción del *pin* seleccionado (línea 31)
- `onPress`: método que se ejecutará cuando se presione sobre el *pin* (línea 32)

```
26  <Marker
27    coordinate={{ latitude, longitude }}
28    title={title}
29    pinColor={colors.darkPrimaryColor}
30    description={description}
31    onCalloutPress={() => onCalloutPress(marker)}
32    onPress={_showGoogleMaps}
33  />
```

Código 3.34 Componente `Marker`

El Código 3.35 muestra el componente `MapView` que recibe las siguientes propiedades:

```
43  <MapView
44    showsUserLocation={true}
45    showsMyLocationButton
46    toolbarEnabled={true}
47    onMapReady={_onMapReady}
48    provider="google"
49    region={region}
50    onRegionChangeComplete={_onRegionChangeComplete}
51    style={[{ ...StyleSheet.absoluteFillObject }, { bottom: _bottom }]}
52    loadingEnabled
53    loadingIndicatorColor={colors.darkPrimaryColor}
54    loadingBackgroundColor={colors.whiteColor}
55  >
56    {_markers}
57  </MapView>
```

Código 3.35 Componente `MapView`

- `showUserLocation`: propiedad tipo `boolean` para activar la posición del usuario en tiempo real (línea 44)
- `showMyLocationButton`: propiedad tipo `boolean` que muestra un botón de ubicación del usuario (línea 45)
- `toolbarEnable`: propiedad para mostrar herramientas para abrir la aplicación de Google Maps o navegar al sitio escogido (línea 46)
- `onMapReady`: método que se ejecuta cuando el mapa ha terminado de cargar (línea 47)
- `provider`: proveedor del mapa (línea 48)
- `region`: propiedad para especificar la región que va a mostrar el mapa (línea 49)
- `onRegionChangeComplete`: método cuando la región cambia (línea 50)

- `style`: propiedad para especificar el estilo del mapa (línea 51)
- `loadingEnable`: propiedad que muestra un indicador de carga hasta que el mapa esté listo (línea 52),
- `loadingIndicator`: color del indicador `loadingEnable` (línea 53)
- `loadingBackgroud`: color del fondo de la pantalla de carga (línea 54)

En la línea 56 se agregan los marcadores al mapa.

3.3.6.2.2 Redux para el mapa colaborativo

De manera similar, al *Sprint* anterior, se implementó Redux en el mapa colaborativo con sus *actions* y *reducer.* Redux acepta un único *reducer*, por este motivo se utiliza `combineReducer` (línea 1) mostrado en el Código 3.36, para combinar todos los *reducer* dentro de uno solo. Los *reducers* `mapReducer`, `feedReducer` y `userReducer` se importan desde la línea 2 a la 4 y se agregan al `combineReducers` de la línea 6 la 10.

```
1   import { combineReducers } from 'redux-immutable';
2   import map from './mapReducer';
3   import feed from './feedReducer';
4   import user from './userReducer';
5
6   export default combineReducers({
7       map,
8       feed,
9       user,
10  });
```

Código 3.36 Código del `combineReducer`

3.3.6.2.3 Métodos de Firebase para el mapa colaborativo

El Código 3.37 muestra la función `getPins` el cual recibe como parámetro una función `callback`. En la línea 57 se obtienen los datos, de la base de datos `mapDB`, en la referencia `/map` y el `callback` se ejecutará cuando se obtengan dichos datos (línea 59).

```
56  const getPins = callback => {
57    mapDB
58      .ref("/map/")
59      .on("value", snap => callback(snap.val()));
60  };
```

Código 3.37 Función `getPins`

El Código 3.38 muestra la función `uploadPin` que recibe como parámetros `pinCategory`, `pinName`, `pinDescription` y `region`. En la línea 73 se obtiene el usuario actual de la aplicación y en la línea 74 se extrae su `id`. De la línea 76 a la 78 se añade una clave única en la base de datos del mapa en la referencia `/map/pin`. De la línea 79 a 87 se crea el objeto `pin` con las propiedades establecidas en la base de datos. En la línea 88 se crea la variable `updatesMap` para agregar las actualizaciones en las referencias `/map/pin/` y `/map/categoryMap/`.

En las líneas 89 a 92 se agregan las rutas con sus respectivas actualizaciones a `updatesMap`. Por último, se envían las actualizaciones a la base de datos `mapDB` con el método `update` (línea 96).

```
72  const uploadPin = (pinCategory, pinName, pinDescription, region) => {
73    const user = getUser();
74    let { uid } = user;
75
76    var newPostKey = mapDB
77      .ref("/map/pin")
78      .push().key;
79    const pin = {
80      id: newPostKey,
81      idUser: uid,
82      idCategory: pinCategory,
83      title: pinName,
84      description: pinDescription,
85      latitude: region.latitude,
86      longitude: region.longitude
87    };
88    var updatesMap = {};
89    updatesMap["/map/pin/" + newPostKey] = pin;
90    updatesMap[
91      "/map/categoryMap/" + pinCategory + "/list/" + newPostKey
92    ] = newPostKey;
93
94    mapDB
95      .ref()
96      .update(updatesMap);
97  };
```

Código 3.38 Función `uploadPin`

El usuario puede visualizar todos los sitios agregados en el prototipo. Pero, puede eliminar solo los *pins* de los lugares de interés que hayan sido agregados por él. Esta validación se realiza con el id del usuario registrado en cada *pin*.

El Código 3.39 muestra la función `removePin` que recibe como parámetro el objeto `pin` que contiene los datos de referencia del *pin* a eliminar. De la línea 146 a 147 se obtiene el `uid` del usuario actual de la aplicación. En la línea 148 se obtiene el `id`, `idCategory` y `idUser` del objeto `pin`.

```
145 const removePin = pin => {
146   const user = getUser();
147   let { uid } = user;
148   const { id, idCategory, idUser } = pin;
149   if (uid === idUser) {
150     return mapDB
151       .ref("/map/categoryMap/" + idCategory + "/list/" + id)
152       .remove()
153       .then(() =>
154         mapDB
155           .ref("/map/pin/" + id)
156           .remove()
157       )
158   } else {
159     throw new Error('No tiene permisos para eliminar este pin');
160   }
161 };
```

Código 3.39 Función `removePin`

En la línea 149 se verifica que el `uid` del usuario sea el mismo que el `idUser` del `pin`, si son iguales se retorna el resultado de borrado del `pin` de la categoría (línea 151 a 157). Si no coinciden se lanzan un error indicando que el usuario no tiene permisos para borrar el `pin` seleccionado.

3.3.6.3 Entregable

En este *Sprint* se tiene como entregable la incorporación del mapa colaborativo a la aplicación móvil.

3.3.7 SPRINT 7

En caso de terremoto se le informa al usuario, a través de SMS, de que ha existido un terremoto. Se abrirá automáticamente la aplicación y preguntará el estado del usuario para reenviarla a sus contactos. Por otro lado, cabe la posibilidad de que el usuario no pueda informar de su estado, por lo que, como alternativa se puede obtener la posición guardada previamente en Firebase y enviarla a sus contactos. La obtención de la posición se realiza por intervalos de tiempo configurables en 15, 30 o 60 minutos y se envía a Firebase en *background* (segundo plano) desde el dispositivo.

React Native no posee la capacidad de realizar tareas en *background* [24], pero si permite a los desarrolladores crear dichas tareas de forma manual y con código nativo, en este caso con el lenguaje de programación Java.

3.3.7.1 Implementación

El uso de módulos nativos en React Native puede ayudar en tareas donde el *framework* aún no cuenta con un módulo, por ejemplo: tareas en segundo plano, procesamiento de imágenes, escucha de SMS entrantes, optimización de datos, entre otros [24].

3.3.7.1.1 Módulo nativo escucha de SMS

La Figura 3.32 muestra el diagrama de la clase que permite recibir los SMS de notificación de terremotos.

Figura 3.32 Diagrama de clases de `SMSReceiver`

El Código 3.40 presenta la clase `SMSReceiver`, derivada de la clase `BroadcastReceiver`[41] para acceder a los eventos que ocurren dentro del dispositivo

[41] BroadcastReceiver: es una clase base que maneja eventos de difusión que se envían dentro del Sistema Operativo Android.

Android. En las líneas 13 y 14 se declaran las variables `SMS_RECIVED` que sirve para hacer referencia a la acción de SMS recibido a través del proveedor del dispositivo y `TAGSMS` que funciona como identificador del SMS que ha llegado. En las líneas 15 y 16 se sobre escribe el método `onReceive` de la clase `BroadcastReceiver`, la cual recibe el contexto de la aplicación y un `Intent`[42] para ejecutarse. En la línea 17 se revisa si en el `Intent` existe la acción `android.provider.Telephony.SMS_RECEIVED`. En la línea 18 se declara un `String` llamado `smsBody`, que almacenará el contenido de los SMS recibidos. En la línea 19 se itera en los mensajes obtenidos en el `Intent`, se obtiene su contenido y se los agrega al objeto `smsBody`. En la línea 23 se revisa si el mensaje empieza con el identificador definido en `TAGSMS` si es así se abre la aplicación en la pantalla de aviso de terremoto.

```
11  public class SMSReceiver extends BroadcastReceiver {
12
13      private static final String SMS_RECIVED= "android.provider.Telephony.SMS_RECEIVED";
14      private static final String TAGSMS = "TERREMOTO";
15      @Override
16      public void onReceive(Context context, Intent intent) {
17          if (intent.getAction().equals(SMS_RECIVED)) {
18
19              String smsBody = "";
20              for (SmsMessage smsMessage : Telephony.Sms.Intents.getMessagesFromIntent(intent)) {
21                  smsBody += smsMessage.getMessageBody();
22              }
23              if (smsBody.startsWith(TAGSMS)) {
24                  Intent warningIntent = new Intent(Intent.ACTION_VIEW,
25                          Uri.parse("apptitulacion://alert?id=com.prototipo_titulacion"));
26                  context.startActivity(warningIntent);
27              }
28          }
29      }
30  }
```

Código 3.40 Clase `SMSReceiver`

Para la notificación de un terremoto se debe iniciar la aplicación automáticamente en la pantalla de alerta. El *mockup* de esta pantalla se muestra en la Figura 3.33.

Figura 3.33 *Mockup* de pantalla de alerta de terremoto

Esta pantalla debe añadirse al `switchNavigator` de la aplicación, las líneas agregadas se muestran en Código 3.41.

[42] *Intent:* o actividad la cual inicia una pantalla en una *app* en Android.

```
Alert: {
  screen: Alert,
  path: "alert"
}
```

Código 3.41 Pantalla de alerta agregada en el `switchNavigator`

Para abrir la aplicación desde la clase `SMSReceiver` se utilizó la propiedad `path` de la librería `react-navigation`. En el Código 3.42 se muestra la etiqueta `data` que se agrega como identificador de la aplicación, esta línea se agregó en el archivo `AndroidManifest.xml` del proyecto Android.

```
<data android:scheme="apptitulacion" />
```

Código 3.42 Código agregado en `AndroidManifest.xml`

Con esto en mente se puede iniciar la aplicación mediante un `Intent` dentro de la aplicación móvil en el código nativo, o desde una aplicación externa, como se muestra en el Código 3.43 que recibe la acción `ACTION_VIEW` la cual permite mostrar la pantalla en primer plano y una URI con el nombre de la aplicación `apptitulacion` seguido del `path` de la pantalla que se desea que se muestre y por último el nombre del paquete.

```
Intent warningIntent = new Intent(Intent.ACTION_VIEW,
        Uri.parse("apptitulacion://alert?id=com.prototipo_titulacion"));
```

Código 3.43 Ejemplo de `Intent` para abrir una aplicación

3.3.7.1.2 Módulo nativo obtención de la posición

Para implementar el requerimiento del envío de la posición se necesita un `JobService`[43] el cual estará recogiendo la ubicación del usuario y guardándola en Firebase. Esta clase no se puede implementar con algún módulo de React Native existente, por lo que se necesita un módulo nativo.

La Figura 3.34 muestra el diagrama de clases del módulo nativo que se describe a continuación:

- `GPSJobService`: es el `JobService` que obtendrá la posición del usuario cada cierto tiempo.
- `SharedPreferencesModule`: es el encargado de hacer uso de `GPSJobService`.
- `SharedPreferencesPackage`: es el encargado de registrar el módulo creado en `SharedPreferencesModule` y compartirlo en el proyecto de React Native.

[43] JobService: Esta es la clase base que maneja las solicitudes asincrónicas que fueron programadas en un JobScheduler.

Figura 3.34 Diagrama de clases para la obtención de la posición

Para implementar el módulo nativo se siguió la guía de React Native para la creación de módulos nativos especificados en [25], donde se indican que las clases `SharedPreferencesModule` y `SharedPreferencesPackage` deben ser creadas para conectar el código nativo con React Native.

El Código 3.44 muestra el método `doBackgound` de la clase `GPSJobService`.

```java
37    private void doBackground(JobParameters jobParameters) {
38        if (ActivityCompat.checkSelfPermission(getApplicationContext(),
39                Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED
40                && ActivityCompat.checkSelfPermission(getApplicationContext(),
41                        Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
42            Toast.makeText(getApplicationContext(), "No tenemos el permiso a recoger tu ubicación", Toast.LENGTH_LONG)
43                    .show();
44            return;
45        }
46        long interval = jobParameters.getExtras().getLong("inteval");
47        String uid = jobParameters.getExtras().getString("uid");
48        LocationManager locationManager = (LocationManager) getApplicationContext()
49                .getSystemService(Context.LOCATION_SERVICE);
50
51        LocationListener locationListener = new LocationListener() {
52            public void onLocationChanged(Location location) {
53                String latitude = Double.toString(location.getLatitude());
54                String longitude = Double.toString(location.getLongitude());
55                String lastConnection = DateFormat.format("dd-MM-yyyy hh:mm:ss", location.getTime()).toString();
56                Map<String, Object> position = new HashMap<>();
57                position.put(LATITUDE, latitude);
58                position.put(LONGITUDE, longitude);
59                position.put(LASTCONNECTION, lastConnection);
60                db.child("users").child(uid).child("lastPosition").setValue(position)
61                        .addOnSuccessListener(new OnSuccessListener<Void>() {
62                            @Override
63                            public void onSuccess(Void aVoid) {
64                                Toast.makeText(getApplicationContext(), "EXITO ", Toast.LENGTH_LONG).show();
65
66                            }
67                        }).addOnFailureListener(new OnFailureListener() {
68                            @Override
69                            public void onFailure(@NonNull Exception e) {
70                                Toast.makeText(getApplicationContext(), "FALLO", Toast.LENGTH_LONG).show();
71
72                            }
73                        });
74            }
75        };
76        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, interval, 0, locationListener);
77        jobFinished(jobParameters, true);
78    }
```

Código 3.44 Método `doBackgound`

El método `doBackgound` (línea 37) realiza la obtención de la posición y recibe como parámetro un objeto `JobParameter` que contiene los objetos `interval` y `uid` que representan el intervalo de ejecución de la tarea y el id del usuario respectivamente. En las líneas 38 a 45 se revisa si se tienen los permisos para acceder a la posición del dispositivo si es así, se continua con el código, caso contrario se muestra un mensaje al usuario con la leyenda `"No tenemos permiso a recoger tu ubicación"`.

En la línea 46 y 47 se obtienen los parámetros `interval` y `uid`. En la línea 48 se utiliza un objeto del tipo `LocationManager` para obtener el servicio `LOCATION_SERVICE` del dispositivo. De las líneas 51 a 75 se declara el objeto `locationListener` del tipo `LocationListener`, el cual sirve para escuchar los cambios de la posición del dispositivo con el método `onLocationChange`. Este método recibe como parámetro un objeto del tipo `Location` que contiene las coordenadas (`latitude` y `longitude`) de la ubicación del dispositivo. En la línea 55 se guarda la fecha actual en la variable `lastConnection`.

En la línea 56 se declara la variable `position` del tipo `Map`, que permite representar los datos en una estructura clave-valor, muy similar a JSON. En las líneas 57 a 59 se agregan a la variable `position` los datos de `latitude`, `longitude` y `lastConnection`.

En la línea 60 se envían los datos a Firebase en la ruta`/users/uid/lastPosition/` y se guarda la posición. Si los datos se han guardado con éxito se ejecutará el método `onSuccess`, el cual muestra un mensaje al usuario y se da por finalizado el servicio.

En la Línea 76 se piden las actualizaciones de la posición con el método `requestLocationUpdates` con los parámetros:

- Proveedor: proveedor de la posición, en este caso el `LocationManager.GPS_PROVIDER` del dispositivo.
- Tiempo mínimo: es el tiempo mínimo de actualización de la ubicación, en milisegundos, en este caso es el objeto `interval` especificado.
- Distancia mínima: es la distancia mínima entre actualizaciones de ubicación, en metros, en este caso es `0`.
- Escuchador: es el objeto que recibe las actualizaciones de la ubicación, en este caso es el `locationListener`.

En la línea 77 se vuelve a reprogramar el servicio para que la posición se siga enviando continuamente a Firebase.

El Código 3.45 muestra un fragmento del archivo `SharedPreferencesModule.java`. En la línea 30 se especifica que el método `setTimeJob` se podrá invocar como un método

en React Native que recibe `_uid` que representa el id del usuario e `_interval` que representa el intervalo para enviar la posición. Para activar el servicio se revisa que `_interval` no sea nulo ni que este vacío (línea 31). En caso de que no se reciban estos parámetros se cancelará el `JobSchedualer`[44] y se mostrará un mensaje al usuario que se ha cancelado el envío de la posición.

```
29     @ReactMethod
30     public void setTimeJob(String _uid, String _inteval) {
31         if (_inteval≠null && !_inteval.isEmpty()) {
32             long inteval = Long.parseLong(_inteval, 10);
33             PersistableBundle bundle = new PersistableBundle();
34             bundle.putString("uid", _uid);
35             bundle.putLong("inteval", inteval);
36             ComponentName componentName = new ComponentName(getReactApplicationContext(), GPSJobService.class);
37             JobInfo info;
38             if (Build.VERSION.SDK_INT ≥ Build.VERSION_CODES.N) {
39                 info = new JobInfo.Builder(99, componentName)
40                         .setRequiresCharging(false)
41                         .setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY)
42                         .setPersisted(true)
43                         .setMinimumLatency(inteval)
44                         .setExtras(bundle)
45                         .build();
46             } else {
47                 info = new JobInfo.Builder(99, componentName)
48                         .setRequiresCharging(false)
49                         .setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY)
50                         .setPersisted(true)
51                         .setPeriodic(inteval)
52                         .setExtras(bundle)
53                         .build();
54             }
55             JobScheduler scheduler = (JobScheduler) mContext.getSystemService(JOB_SCHEDULER_SERVICE);
56             int result = scheduler.schedule(info);
57             if (result == JobScheduler.RESULT_SUCCESS) {
58                 Toast.makeText(getReactApplicationContext(), "Se establecio el tiempo con exito", Toast.LENGTH_LONG).show();
59             } else {
60                 Toast.makeText(getReactApplicationContext(), "Error, vuelve a intentar", Toast.LENGTH_LONG).show();
61             }
62         } else {
63             JobScheduler scheduler = (JobScheduler) mContext.getSystemService(JOB_SCHEDULER_SERVICE);
64             scheduler.cancel(99);
65             Toast.makeText(getReactApplicationContext(), "Se cancelo el envío de tu posición", Toast.LENGTH_LONG).show();
66         }
67     }
```

Código 3.45 Fragmento del archivo `SharedPreferencesModule.java`

En la línea 32 se transforma el intervalo al tipo `long` en base 10. En línea 33 se declara la variable `bundle` del tipo `PersistableBundle`[45] que permite almacenar persistentemente los parámetros de este método. En las líneas 34 y 35 se añaden las variables `_uid` e `Interval` al `bundle`. En la línea 36 se declara la variable `componentName` del tipo `ComponentName`[46] y se identifica al `GPSJobService` dentro del contexto de la aplicación de React Native [26]. En la línea 37 se declara `JobInfo`, el cual es un contenedor para enviar información a un servicio.

En la línea 38 se revisa la versión del SDK de Android, ya que para versiones superiores o iguales a la versión Nougat (Android 5.0) cambia la codificación del servicio.

---

[44] JobSchedular: es una clase que sirve para programar tareas repetitivas en un tiempo establecido.
[45] PersistableBundle: es una clase que permite persistir variables en formato clave-valor en la memoria del dispositivo.
[46]ComponentName : sirve como identificador de un componente de aplicación.

En la línea 39 se inicializa el JobInfo con el Buider que recibe un entero para identificar al servicio y ComponentName y se definen los siguientes métodos:

- setRequiresCharging: especifica si el servicio requiere que el dispositivo este cargando, en este caso no se necesita.
- setRequireNetworkType: especifica qué tipo de red debe utilizar el servicio, en este caso utilizará cualquiera tipo de red.
- setPersisted: especifica si el servicio es persistente, es decir, que persista el servicio a pesar del reinicio del dispositivo.
- setMinimumLatency: especifica el retardo para que el servicio se realice cada cierto tiempo, una vez que se ha realizado el servicio se volverá a hacer en cualquier momento del intervalo especificado.
- setExtras: especifica las variables que se pasarán al servicio GPSJobService.

En caso de que el dispositivo sea de menor versión cambian el método:

- setPeriod: de manera similar a setMinimumLatency, define el intervalo para que el servicio se realice de manera periódica en un intervalo de tiempo.

En ambos casos se construyen el objeto con el método build.

En la línea 55 se declara un objeto JobScheduler. En la línea 56 se define el resultado de si se ha establecido el JobScheduler. Si se ha establecido se informa al usuario con un mensaje "Se ha establecido el tiempo con exito", caso contrario se muestra el mensaje "Error, vuelve a intentar".

Por último, en el Código 3.46 se registra el servicio en el archivo AndroidManifest.xml especificando la clase del servicio, en este caso GPSJobService y el permiso correspondiente para enlazar el servicio.

```
<service android:name=".GPSJobService"
    android:permission="android.permission.BIND_JOB_SERVICE"/>
```

Código 3.46 Fragmento del archivo AndroidManifest.xml

Para hacer uso del módulo nativo en la aplicación React Native se implementó, en una lista desplegable, un componente PickerJob. El Código 3.47 muestra el componente implementado, el método onValueChange (línea 13 a 21) recibe el valor seleccionado de la lista, lo guarda en el *state* local y se emplea NativeModules de la librería react-native para acceder al método setTimeJob que ejecuta el servicio especificado en el Código 3.45.

```
13        onValueChange = (value) => {
14            this.setState({ selected: value, })
15            NativeModules.SharePreference.setTimeJob(uid, value)
16        }
```

Código 3.47 Fragmento del componente `PickerJob`

3.3.7.2 Entregables

En este *Sprint* se tiene como entregable las funcionalidades para recibir notificaciones de terremotos y enviar la posición en segundo plano a Firebase. El ANEXO C contiene el código de la aplicación móvil.

3.3.8 SPRINT 8

Los usuarios reciben notificaciones cuando sucede un terremoto, esta notificación se envía desde el servidor. Además, el servidor se encarga de reenviar el estado de los usuarios.

El diagrama de secuencia de notificación se presenta en la Figura 3.35.

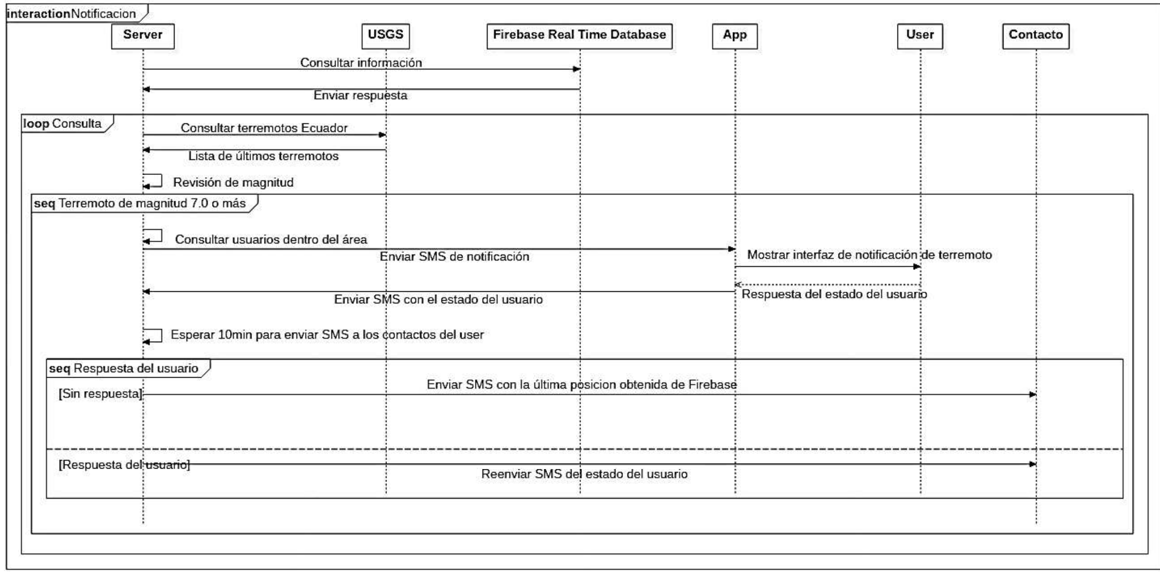

Figura 3.35 Diagrama de secuencia: notificación y reenvío del estado del usuario

El flujo comienza cuando el servidor consulta la información del sistema en Firebase. Se realiza un lazo para consultar a USGS sobre terremotos ocurridos en Ecuador, en caso de obtener registros de terremotos el servidor discriminará por magnitud, notificando a los usuarios vía SMS cuando la magnitud sea mayor o igual a un umbral configurable por el administrador. Esta notificación desencadenará un aviso en la aplicación móvil preguntando el estado del cliente. El servidor espera 10 minutos por esta respuesta y la reenviará vía SMS a sus contactos registrados. En caso de no obtener respuesta de parte del cliente el sistema enviará un SMS con la última posición y fecha registrada en Firebase a sus contactos.

3.3.8.1 Implementación

Para implementar los requerimientos del presente *Sprint*, se ha montado un circuito de acuerdo con el diagrama de la Figura 3.36, el cual muestra la conexión de la Raspberry Pi, que ejecutará el servidor, y el módulo GSM 800L.

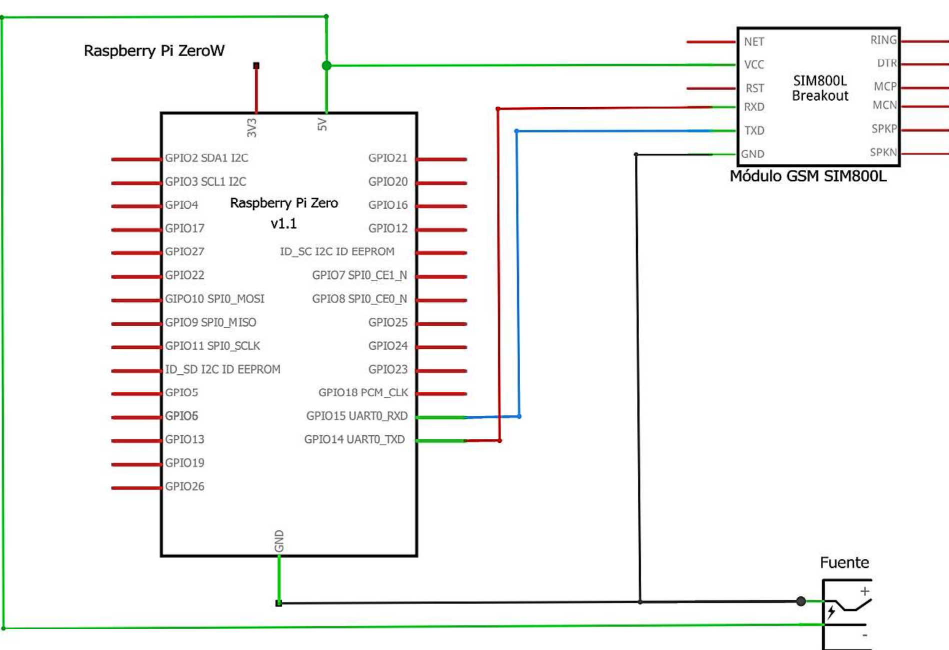

Figura 3.36 Diagrama de conexión para el servidor de SMS

Para crear un proyecto en nodeJS, se ejecuta el comando mostrado en el Código 3.48, el cual requiere de los archivos mostrados en la Figura 3.37: `index.js` contiene el código a ejecutarse, y `package.json` contiene información de los paquetes que se usan dentro del proyecto.

```
Findme-Earthquake henry$ npm init -y
```

Código 3.48 Creación de un proyecto en nodeJS

Figura 3.37 Archivos de un proyecto en nodeJS

Para la implementación de este *Sprint* se añadieron las siguientes librerías al archivo `package.json` del proyecto:

- `express`: es un *framework* para crear aplicaciones o servicio web y cuenta con manejadores de rutas, manejadores de datos, entre otros [27].
- `serialport-gsm`: permite acceder al modem celular y sus funciones.
- `axios`: realiza peticiones HTTP basado en promesas.

- `body-parse`: es un *middleware* para `express`, que analiza las cabeceras de las peticiones HTTP y obtiene el *body* (cuerpo) representado en objetos JSON.
- `firebase`: cuenta con el SDK de Firebase para acceder a los servicios de autenticación, bases de datos y almacenamiento de archivos.

La librería `express` posee varios métodos, la Tabla 3.6 muestra los métodos usados en este *Sprint*.

Tabla 3.6 Métodos de la librería `express`

Método	Acción
`use`	Agrega un *middleware* a la aplicación creada en `express`.
`listen`	Vincula y escucha las conexiones en el *host* y el puerto especificado.
`get`	Ejecuta un método `HTTP GET`.
`post`	Ejecuta un método `HTTP POST`.
`delete`	Ejecuta un método `HTTP DELETE`.

La librería `axios` permite realizar peticiones HTTP, entre ellas las más importantes son:

- `GET`: petición para obtener cualquier tipo de información.
- `POST`: petición para solicitar al servidor almacene algún tipo de información.
- `DELETE`: petición para solicitar al servidor que elimine algún archivo o información en una ruta específica.

La Figura 3.38 muestra tres microservicios del servidor:

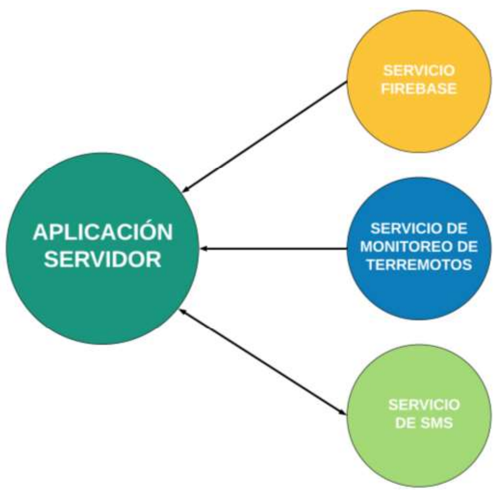

Figura 3.38 Diagrama de la aplicación servidor

- Servicio de Firebase: para el acceso a los servicios de base de datos.
- Servicio de monitoreo de terremotos: para consultar los terremotos a la entidad USGS.
- Servicio SMS: envía y lee SMS en el módem GSM del prototipo.

La inicialización del servicio de Firebase y sus métodos, se implementaron de manera similar a *Sprints* anteriores. El uso de estos métodos se mostrará en la aplicación servidor.

3.3.8.1.1 Servicio de monitoreo de terremotos

Para el servicio de monitoreo de terremotos se emplea el servicio de USGS, el cual es un servicio de consulta de información de terremotos en tiempo real, a través de una API RESTful. El servicio utiliza el formato GeoJSON para su respuesta, además, la API presenta una serie de parámetros que permiten personalizar una petición de consulta [28]. Los parámetros definidos por la API se muestran en la Tabla 3.7.

Tabla 3.7 Parámetros de consulta

Campo	Tipo de dato	Parámetros	Descripción
Formato	`String`	`format`	Formato de la respuesta.
Hora	`String`	`endtime`	Límite a eventos en o antes de la hora de finalización especificada.
		`starttime`	Límite a eventos en o después de la hora de inicio especificada.
		`updatedafter`	Límite a eventos actualizados después del tiempo especificado en formato UTC[47].
Ubicación	`Integer`	`minlatitude`	Límite a eventos con una latitud mayor que el mínimo especificado.

[47] UTC (*Coordinated Universal Time*): es el estándar de tiempo que regula los relojes y el tiempo en el mundo.

Campo	Tipo de dato	Parámetros	Descripción
		`minlongitude`	Límite a eventos con una longitud mayor que el mínimo especificado.
		`maxlatitude`	Límite a eventos con una latitud menor que el máximo especificado.
		`maxlongitude`	Límite a eventos con una longitud menor que el máximo especificado.
Extensiones	`Integer`	`nodata`	Define el código de error que se devolverá cuando no se encuentren datos.
Otros	`Integer`	`limit`	Límite del número de eventos en el resultado.

La Figura 3.39 presenta una captura de pantalla de herramienta la Postman[48], la cual muestra la petición `GET` de los terremotos que hayan ocurrido luego de las `15H00` del `12/07/2019` (`updateAfter`) entre las coordenadas (`-5.400, -93.600`) (`minlat,minlon`) y (`1.770, -74.800`) (`maxlat,maxlon`) al servicio USGS. Si han ocurrido terremotos se especifica que el límite de terremotos en la respuesta sea `1` (`limit`). En caso de que no haya ocurrido ningún terremoto se especifica que la respuesta tenga un estatus `404` (`nodata`).

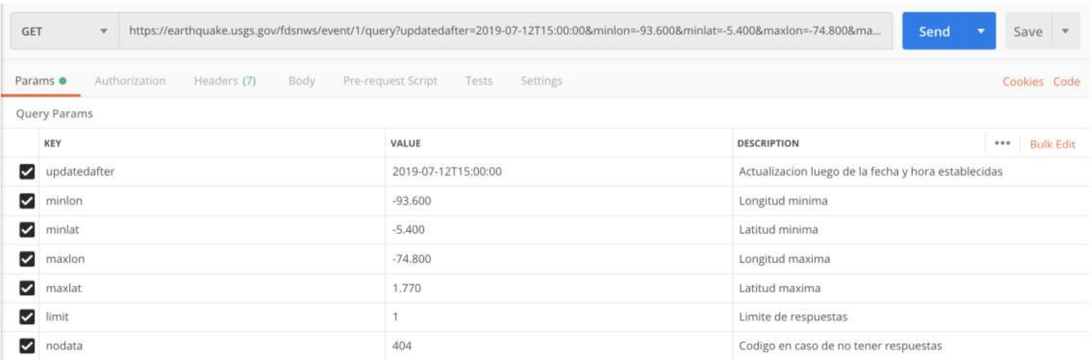

Figura 3.39 Petición al servicio USGS

[48] Postman: es una herramienta que se utiliza para el *testing* de API REST

En la Figura 3.40 se muestra la respuesta obtenida desde USGS.


```
"features": [
    {
        "type": "Feature",
        "properties": {
            "mag": 4.7999999999999998,
            "place": "205km NNW of Puerto Ayora, Ecuador",
            "time": 1563019509000,
            "updated": 1563020587040,
            "tz": -360,
            "url": "https://earthquake.usgs.gov/earthquakes/eventpage/us70004jjy",
            "detail": "https://earthquake.usgs.gov/fdsnws/event/1/query?eventid=us70004jjy&format=geojson",
            "felt": null,
            "cdi": null,
            "mmi": null,
            "alert": null,
            "status": "reviewed",
            "tsunami": 0,
            "sig": 354,
            "net": "us",
            "code": "70004jjy",
            "ids": ",us70004jjy,",
            "sources": ",us,",
            "types": ",geoserve,origin,phase-data,",
            "nst": null,
            "dmin": 1.8029999999999999,
            "rms": 0.70999999999999996,
            "gap": 165,
            "magType": "mb",
            "type": "earthquake",
            "title": "M 4.8 - 205km NNW of Puerto Ayora, Ecuador"
        },
        "geometry": {
            "type": "Point",
            "coordinates": [
                -90.793499999999995,
                1.0683,
                10
            ]
        },
        "id": "us70004jjy"
    }
]
```


Figura 3.40 Respuesta desde USGS

3.3.8.1.2 Servicio de SMS

Para el envío de SMS se han creado servicios web para leer, enviar y eliminar SMS en el módem.

Para este requerimiento se utiliza la librería `serialport-gsm` la cual cuenta con métodos y eventos para el manejo de un módem GSM. La Tabla 3.8 muestra los métodos a usarse para este *Sprint*.

Tabla 3.8 Métodos de la librería `serialport-gsm`

Método	Acción
`initializeModem`	Inicia el módem
`setModemMode`	Establece el modo de conexión con el módem

Método	Acción
`checkSimMemory`	Configura el almacenamiento de la memoria del SIM[49].
`getSimInbox`	Devuelve los SMS de la bandeja de entrada.
`sendSMS`	Envía un SMS.
`deleteMessage`	Elimina un SMS.
`open`	Abre la comunicación serial.
`close`	Cierra la comunicación serial.
`on`	Permite acceder a los métodos de la librería con su nombre y una función.

El Código 3.49 presenta la configuración de la comunicación serial con el módem GSM.

```
3   const serialportgsm = require('serialport-gsm');
4   const gsmModem = serialportgsm.Modem();
5   let options = {
6     baudRate: 115200,
7     dataBits: 8,
8     parity: 'none',
9     stopBits: 1,
10    xon: false,
11    rtscts: false,
12    xoff: false,
13    xany: false,
14  }
```

Código 3.49 Configuración de la comunicación serial con el módem GSM

En la línea 3 la librería `serialpor-gsm`. En la línea 4 se inicializa una instancia del objeto `Modem` de la librería `serialportgsm`. De en las líneas 5 a 14 se crea el objeto `options`, el cual contiene los siguientes parámetros para la comunicación con el módem GSM:

- `bautRate`: velocidad en baudios del puerto
- `dataBits`: bits de datos en la PDU[50] de la comunicación serial
- `parity`: paridad de la comunicación serial
- `stopBits`: bits de parada de la comunicación serial
- `xon`, `rtscts`, `xoff` y `xany`: ajustes de control de comunicación serial

---

[49] SIM (*Subscriber Identity Module*): es una tarjeta que almacena la información del suscriptor usada para identificarse ante la red.
[50] PDU (*Protocol Data Unit*): se utilizan para el intercambio de datos entre protocolos.

El Código 3.50 muestra el método `open` que abre la comunicación especificando la ruta del puerto serial donde fue montado el módulo GSM y las opciones de comunicación definidas en el objeto `options`. Una vez inicializada la comunicación se pueden utilizar los métodos especificados en la Tabla 3.8 para acceder a las funcionalidades del módem GSM.

```
gsmModem.open('/dev/ttyS0', options);
```

Código 3.50 Método `open`

Para hacer disponible las funcionalidades de envió y lectura de SMS del módem GSM, a través de HTTP, se implementó un servicio con `express` [27].

El Código 3.51 presenta un fragmento del archivo `index.js` que implementa el servicio de SMS.

```
17    const express = require('express');
18    const bodyParser = require('body-parser');
19    const app = express();
20    app.use(bodyParser.urlencoded({ extended: false }));
21    app.use(bodyParser.json());
```

Código 3.51 Fragmento del archivo `index.js` del servicio de SMS

En las líneas 17 y 18 se importan las librerías `express` y `body-parser`. En la línea 4 que se crea una aplicación servidor con `express`, a la cual se le denomina `app`. En las líneas 10 y 11 se le agrega el *middleware* `bodyParse` a `app`, para manejar los datos del *body* en formato JSON.

El Código 3.52 muestra el método `listen`, el cual inicia el servidor en el puerto `3000` y muestra un mensaje por consola cuando se ha iniciado la escucha.

```
app.listen(3000, function () {
  console.log('Server sms listening on port 3000!');
});
```

Código 3.52 Fragmento del archivo `index.js` del servicio de SMS

El servicio cuenta con los siguientes métodos en la ruta `/sms`:

- `GET`: obtiene los SMS de la bandeja de entrada del módem GSM
- `POST`: envía un SMS desde el módem GSM
- `DELETE`: elimina un SMS de la bandeja de entrada del módem GSM

El Código 3.53 presenta la implementación del método `POST`. En la línea 86 se especifica `/sms` como ruta para acceder al servicio y una función la cual se ejecutará cuando se realice la petición. Esta función tiene como parámetros los objetos `req` y `res` que

representan la petición y la respuesta. En las líneas 87 y 88 se extrae el destinatario (recipient) y el contenido (message) desde el body de req. En la línea 90 se muestra por consola que se ha realizado una petición POST al servicio. En la línea 91 se utiliza el método sendSMS de la librería serialport-gsm. Este método recibe como parámetros recipient, message, una variable del tipo boolean que indica si el mensaje es multimedia o no, y una función a ejecutarse cuando se haya enviado o no un SMS. Finalmente, en la línea 92 se revisa si existe algún error. Si existe, se envía como respuesta el error (línea 93) caso contrario se envía el resultado (result) del método (línea 94).

```
86  app.post('/sms', function (req, res) {
87    const recipient = req.body.recipient;
88    const message = req.body.message;
89
90    console.log(`post/sms ${recipient} ${message}`);
91    gsmModem.sendSMS(recipient, message, false, (result, err) => {
92      err
93        ? res.status(500).send(`Failed to send sms ${err}`)
94        : res.send(result)
95    });
96  });
```

Código 3.53 Método POST del servicio de SMS

Los métodos GET y DELETE del servicio de SMS se implementaron de manera similar.

3.3.8.1.3 Aplicación servidor

La aplicación servidor de Redux y Firebase. De Firebase obtienen las configuraciones del servidor y los usuarios del prototipo.

En el Código 3.54 se presenta un ejemplo del método on que obtiene las configuraciones del servidor. En la línea 9 se extrae los valores y se lo asigna en el objeto settings. En la línea 10 se realiza un dispatch de los datos obtenidos al store de la aplicación.

```
8   firebase.refSettings.on("value", snapshot => {
9       const settings = snapshot.val();
10      store.dispatch({ type: 'SET_SETTINGS', payload: settings });
11  });
```

Código 3.54 Método on que obtiene las configuraciones del servidor

De manera similar se implementaron la obtención de los usuarios desde Firebase.

El Código 3.56 muestra la función getLastEvents que realiza la consulta a la API de USGS. En la línea 1 se importa axios. En la línea 2 se define una variable con la URL de la consulta. En la línea 3 se define la función getLastEvents que recibe como parámetro offsetTime, en milisegundos. La consulta obtendrá los eventos que ocurrieron luego de una fecha y hora especificada en el objeto date (línea 4), la cual se obtiene restando el

offsetTime de la fecha y hora actual del servidor. Esta fecha debe ser transformada al formato UTC con el método ISOString para usarla como parámetro en la consulta.

En la línea 5 se retorna la petición GET realizada con axios al servicio de USGS. Si el status de la respuesta es 200 (línea 8) se revisa si existe data (línea 10) y se retorna los eventos del campo features. Caso contrario se retorna un arregla vacío.

```
1  const axios = require('axios');  15K (gzipped: 5.1K)
2  const url = "https://earthquake.usgs.gov/fdsnws/event/1/query?format=geojson&nodata
3  const getLastEnvents = (offsetTime) => {
4      const date = new Date((Date.now() - offsetTime)).toISOString();
5      return axios.get(url.replace("/:time", date))
6      .then(response => {
7          const { status } = response;
8          if (status === 200) {
9              const { data } = response;
10             if (data.features.length > 0) {
11                 const { features } = data;
12                 return features;
13             } else {
14                 return [];
15             }
16         }
```

Código 3.55 Función getLastEvents

La aplicación debe consultar los terremotos periódicamente de acuerdo con las configuraciones del servidor. El Código 3.56 muestra un fragmento de como se realiza la consulta periódicamente.

```
86      let { settings } = store.getState();
87      const { offsetTime, requestTime } = settings;
88      let requestLastEvents = setInterval(() => getLastEnvents(offsetTime)
89          .then(data => processDataEarthquake(data))
90          .catch(error => {
91              let log = `${new Date()} Error usgs  ${error}`;
92              console.log(log);
93          }),
94          requestTime)
```

Código 3.56 Función initServer

En la línea 84 se extraen las configuraciones (settings) desde el store de la aplicación. En la línea 85 se obtienen los parámetros del servidor offsetTime y requestTime. En la línea 86 mediante el objeto requestLastEvents, que usa el método setInterval[51], se consultan los terremotos en el intervalo de tiempo requestTime (línea 94), a la API de USGS con la función getLastEvents. Al resolverse la petición se ejecuta el método then que ejecuta la función processDataEarthquake que recibe como parámetro la respuesta de la consulta data. En caso de que ocurra un error se imprime un mensaje por consola.

[51] setInterval: es un método que llama a una función o evalúa una expresión a intervalos específicos en milisegundos.

El Código 3.57 muestra la función `processDataEarthquake` que recibe como parámetro el objeto `data`, que representa un arreglo de los terremotos ocurridos. En la línea 120 se revisa si `data` posee elementos, si es así se itera por cada elemento. De la línea 122 a la 123 se extraen la magnitud (`mag`) del terremoto. En la línea 124 se obtienen las `settings` del `store` de la aplicación. En la línea 125 se obtiene la mínima magnitud de un terremoto (`minMagnitud`) de `settings`. En la línea 126 se revisa si la magnitud del terremoto es mayor o igual a la mínima magnitud especificada. Si es así se actualizará con el método `dispatch` el `store` con el terremoto registrado.

```
119  const processDataEarthquake = (data) => {
120      if (data.length > 0) {
121          data.map(featureData => {
122              const { properties } = featureData;
123              const {  mag } = properties;
124              const { settings } = store.getState();
125              const { minMagnitude } = settings;
126              if (mag >= minMagnitude)
127                  store.dispatch({
128                      type: 'SET_EARTHQUAKE',
129                      payload: featureData
130                  })
131          })
132      }
133  }
```

Código 3.57 Función `processDataEarthquake`

En caso de que ocurra un terremoto que supere la magnitud mínima configurada en el servidor, se actualizará el `store` de la aplicación. Esto ejecutará un subproceso con el método `fork`[52] de nodeJS, que notificará a los usuarios que ocurrió un terremoto. El Código 3.58 muestra un ejemplo de la creación de un subproceso. En la línea 117 se crea un subproceso del programa de notificaciones definido en el archivo `notification.js`. En la línea 118 se envían los objetos: `featureData` que representa un terremoto, `users` usuarios a notificar y `waitTime` tiempo de espera por una respuesta de los usuarios.

```
117  const subProcess = fork('./notification.js');
118  subProcess.send({ featureData, users, waitTime })
```

Código 3.58 Subproceso de notificación

Cuando el subproceso recibe un `message` ejecuta una función *callback* asíncrona, la cual se presenta en el Código 3.59.

De la línea 6 a la 9 se obtienen las propiedades del terremoto como: lugar (`place`), latitud (`latitude`) y longitud (`longitude`). La función `geopoint2area` (línea 10) calcula un área cuadrada, similar al de la Figura 3.41, donde $usuario_1$ se encuentra dentro del área y debe ser notificado, mientras que $usuario_2$ se encuentra fuera de la misma y por lo tanto no

[52] Fork: es un módulo, integrado en nodeJS, que proporciona la capacidad de generar procesos secundarios.

será notificado. La función `geopoint2area` internamente hace uso la fórmula de Haversine [29] para calcular el área, presentada en Ecuación 3.1, donde φ y λ repesentan la latitud y longitud del terremoto y `d`, en este caso 20 kilómetros, de la distancia del área afectada. De la línea 11 a 16 define la variable `userInsideArea` que revisa si los usuarios se encuentran dentro del área afectada de acuerdo con su última posición registrada (`lastPosition`) con la función `isIntoArea` (línea 15).

```
5  process.on("message", async ({ featureData, users, waitTime }) => {
6      const { geometry, properties } = featureData;
7      const { place } = properties;
8      const longitude = geometry.coordinates[0];
9      const latitude = geometry.coordinates[1];
10     const area = utils.geopoint2area({ longitude, latitude }, 20);
11     const userInsideArea = users.filter(user => {
12         const { lastPosition } = user;
13         if (lastPosition) {
14             const { latitude, longitude } = lastPosition;
15             return utils.isIntoArea(latitude, longitude, area)
16         } else { return true; }
17     })
```

Código 3.59 Método `process.on`

$$d = 2R * \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Ecuación 3.1 Fórmula de Haversine

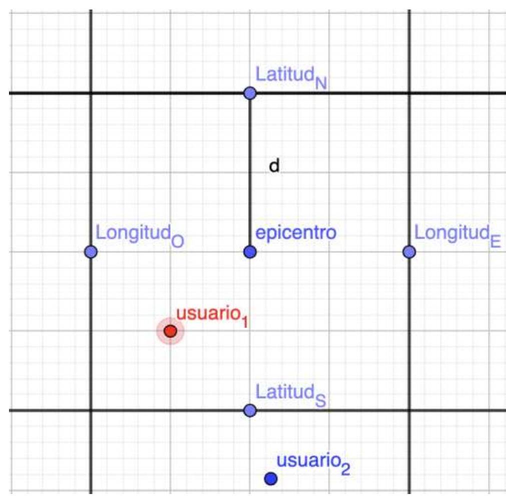

Figura 3.41 Área cuadrada afectada por el terremoto

El Código 3.60 muestra la notificación del terremoto a los usuarios dentro del área afectada. En la línea 18 se obtienen los números de teléfono de estos usuarios. En la línea 19 la función `sendSms` envía una notificación a estos usuarios a través del servicio de SMS. En la línea 20 se esperará el tiempo especificado en `waitTime` para leer y guardar los SMS con la respuesta de los usuarios en el arreglo `smsResponse`. Una vez que ha pasado el intervalo de lectura se crea la variable `recipients_messages` donde se iterarán los usuarios que se encuentran en el área afectada (línea 21) y de cada usuario se extraen los

contactos (`contacts`), nombre (`name`), cédula de identidad (`ci`), tipo de sangre (`blood`) y número celular (`phoneNumber`).

En la línea 23 se revisa si el usuario ha informado su estado y se guarda en el objeto `sms`, si se encuentra su número celular en el arreglo `smsResponse`. Si el usuario ha informado su estado se extrae el contenido del mensaje (línea 26), este contenido se separa por el carácter "_" para generar un arreglo. De la línea 27 a la 30 se extrae su estado (`status`), y su posición.

En la línea 31 se construye el mensaje a enviar a los contactos del usuario. Si el usuario no ha enviado su estado, se extrae su la latitud y longitud de la última posición registrada en Firebase (líneas 34 y 35) y se construye el mensaje a enviar (líneas 36 y 37). En la línea 39 se obtiene los números de teléfono de los contactos del usuario. Finalmente, se retorna los números de teléfono de los contactos y el mensaje a enviar. En la línea 41 se itera `recipients_message` para enviar el estado del usuario a sus contactos con la función `sendSMS` que internamente utiliza el servicio de SMS .

```
18    const phoneNumberUsers = userInsideArea.map(({ phoneNumber }) => phoneNumber);
19    sendSms(phoneNumberUsers, `TERREMOTO ocurrido en ${place}`);
20    const smsResponse = await readSms(waitTime);
21    const recipients_messages = userInsideArea.map(user => {
22        const { contacts, name, ci, blood, phoneNumber } = user;
23        const sms = smsResponse.find(({ sender }) => `+${sender}` === phoneNumber);
24        let message = '';
25        if (sms) {
26            const { message: _message } = sms;
27            const smsFields = _message.split('_');
28            const status = smsFields[0] === 'OK' ? 'esta bien' : 'necesita ayuda';
29            const latitude = smsFields[1];
30            const longitude = smsFields[2];
31            message = `${name} con ci ${ci} y tipo de sangre ${blood}
32                informa que ${status} y se encuentra en google.com/maps/@${latitude},${longitude}`
33        } else {
34            const { lastPosition } = user;
35            const { latitude, longitude } = lastPosition;
36            message = `${name} con ci ${ci} y tipo de sangre ${blood} no informo su estado
37                y su ultima posición registrada fue en google.com/maps/@${latitude},${longitude}`
38        }
39        const phoneNumbers = contacts.filter(({ number }) => number).map(({ number }) => number);
40        return { phoneNumbers, message }
41    })
42    recipients_messages.map(({ phoneNumbers, message }) => sendSms(phoneNumbers, message));
```

Código 3.60 Fragmento del código para la notificación de terremotos a los usuarios

3.3.8.2 Entregables

El presente *Sprint* tiene como entregables un servidor de notificaciones de SMS para terremotos. El ANEXO D contiene el código de la aplicación servidor y sus servicios.

3.3.9 SPRINT 9

El administrador deberá contar con una interfaz web que le permita configurar los parámetros del servidor, crear, editar y eliminar recomendaciones, añadir y eliminar lugares de interés.

Ya que React Native hace uso de React se utilizará esta librería para crear la aplicación web y así reutilizar el código de la aplicación móvil.

3.3.9.1 Diseño

El diseño de la UI de la aplicación web se generó con la herramienta Lunacy con base en los componentes de Material UI.

El administrador tendrá que autenticarse con un correo y contraseña para acceder al panel de administración, el *mockup* de esta pantalla se muestra en la Figura 3.42.

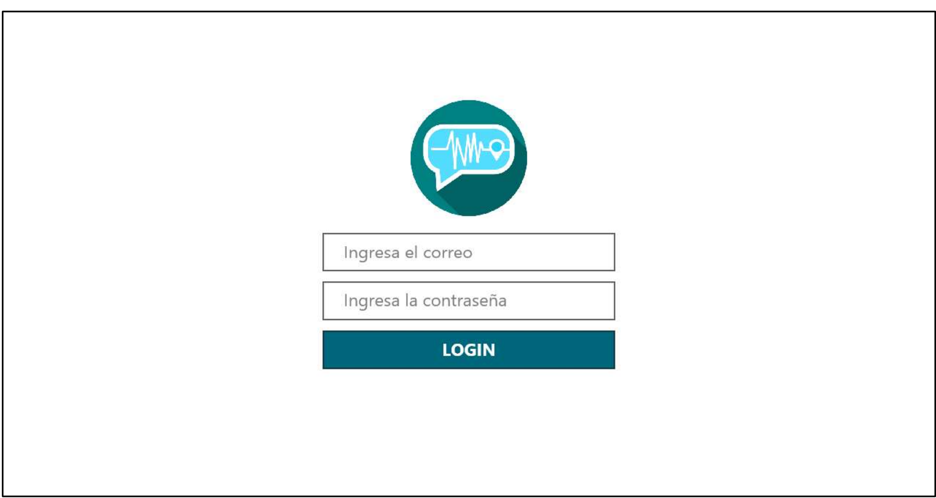

Figura 3.42 *Mockup* de la pantalla de *login*

Una vez autenticado se mostrarán las categorías de las recomendaciones donde podrá para crear, editar, leer o eliminar una recomendación. El *mockup* de esta pantalla se muestra en Figura 3.42.

Figura 3.43 *Mockup* de la pantalla de recomendaciones

El administrador también podrá revisar el mapa colaborativo para ver, agregar o eliminar lugares de interés. La Figura 3.43 muestra el *mockup* generado.

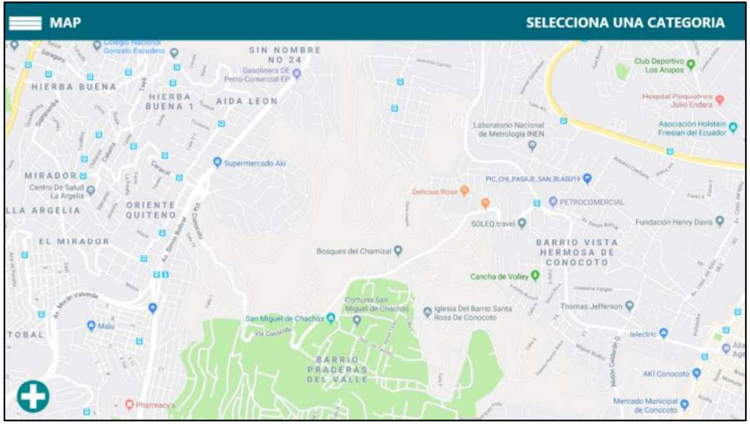

Figura 3.44 *Mockup* de la pantalla del mapa colaborativo

Por último, se contará con la funcionalidad de configurar los parámetros del servidor y la ejecución de un *test* del prototipo. La Figura 3.44 muestra el *mockup* generado.

Figura 3.45 *Mockup* de la pantalla de configuración del servidor

3.3.9.2 Implementación

Para la implementación de este *Sprint* se utilizaron las siguientes librerías:

- `@material-ui`: contiene componentes de UI predefinidos
- `firebase`: contiene el SDK para el uso de Firebase
- `google-maps-react`: librería para el uso de Google Maps
- `react-router-dom`: permite la navegación de pantallas
- `react-redux`: permite hacer uso de Redux dentro de React
- `redux`: contenedor de estado predecible para aplicaciones JavaScript

3.3.9.2.1 Componentes de la UI

Los componentes de UI de la aplicación web se implementaron de manera similar que en el *Sprint* 2 pero con la librería `@material-ui`.

En el Código 3.61 se muestra el código de la UI de la Figura 3.42 y contiene los siguientes componentes:

```
 7    return (
 8        <div style={style.container}>
 9            <Paper style={style.paperPadding}>
10                <Text
11                    text='Hola, es bueno verte de nuevo!'
12                    style={style.helloText} />
13                <Text
14                    text='Inicia sesión'
15                    style={style.loginText} />
16                <GridList style={style.grid} >
17                    <Avatar
18                        stylePhoto={style.avatarStyle}
19                        URLphoto={require('../../Images/logo.png')} />
20                </GridList>
21                <br />
22                <FormGroup>
23                    <TextField
24                        fullWidth={true}
25                        label="Ingresa tu email"
26                        variant="outlined"
27                        type="email"
28                        onChange={onChangeMail}
29                    />
30                    <br />
31
32                    <TextField
33                        fullWidth={true}
34                        label="Password"
35                        variant="outlined"
36                        type="password"
37                        onChange={onChangePassword}
38                    />
39                    <br />
40
41                    <br />
42                    <Button variant="h6" text="Login" onClick={onClickLogin} />
43                </FormGroup>
44            </Paper>
45        </div>
46    )
```

Código 3.61 Implementación de la UI de la pantalla de *login*

- `div`: contenedor genérico dentro de un documento HTML
- `Paper`: es un contenedor con estilos predefinidos
- `Text`: es un componente para mostrar texto con estilos predefinidos
- `GridList`: es un componente que muestra una lista de elemento en una cuadricula
- `Avatar`: es un componente con estilos predefinidos para mostrar una imagen
- `br`: es un salto de línea dentro de un documento HTML
- `FormGroup`: es un contenedor con estilos predefinidos de un formulario
- `TextField`: es un componente `input` para ingresar datos
- `Button`: es un componente botón que cuenta con estilos predefinidos

3.3.9.2.2 Firebase

La conexión de la aplicación web con Firebase se realizó de forma similar que en el *Sprint* 4. Además, el método de autenticación se realiza por medio de un *email* y un *password*, el

Código 3.62 muestra el método `signInWithEmailAndPassword` el cual recibe como parámetros el `mail` y el `password` para validarlo en Firebase. Este método devuelve una promesa para que realice alguna acción luego de autenticar al usuario.

```
auth().signInWithEmailAndPassword(mail, password)
```

Código 3.62 Método `signInWithEmailAndPassword`

Los métodos para crear, eliminar, leer y actualizar los datos de `map`, `feed` y `server` del prototipo en Firebase se lo realiza se manera similar a los implementados en *Sprint* previos.

3.3.9.2.3 Redux

De manera similar a la aplicación móvil se utilizaron las librerías: `redux` y `redux-thunk` para el manejo de los datos dentro de la aplicación web.

La implementación de los *reducers*, *actions* y el *store* se lo realiza de manera similar a los realizados en *Sprints* anteriores, sin embargo, en el archivo `enhancer-redux.js` se deben agregar los cambios mostrados en el Código 3.63 para la persistencia de datos. Para la aplicación web se requiere persistir la sesión del usuario, por lo que se agregó la librería `redux-localstorage` para este propósito.

```
1  import { applyMiddleware, compose } from 'redux';
2  import thunkMiddleware from 'redux-thunk';  1.1K
3  import persistState from 'redux-localstorage';  3
4
5  let middlewares = [thunkMiddleware]
6  export default compose(
7    applyMiddleware(...middlewares),
8    persistState()
9  )
```

Código 3.63 Archivo `enhancer-redux.js`

3.3.9.2.4 Mapa colaborativo

Para acceder al servicio Google Maps se debe agrega el `apiKey` adecuado para la web, en este caso es *Maps JavaScript API,* en el componte `Map` de la librería `google-maps-react`.

El Código 3.64 muestra de las líneas 40 a 46 las propiedades del componente `Map`:

- `style`: estilos para el componente
- `google`: proveedor del mapa
- `zoom`: es un número que indica el enfoque sobre un área en el mapa
- `onClick`: método que se ejecutará cuando se dé un *click* dentro del mapa
- `onRightclick`: método que se ejecutará cuando se dé un *click* derecho dentro del mapa

- `initialCenter`: es la región que será el centro inicial del mapa
- `center`: es la región del centro actual del mapa

```
37          return (
38              <div>
39                  <Map
40                      style={style.map}
41                      google={google}
42                      zoom={12}
43                      onClick={onClickMap}
44                      onRightclick={handleDBClick}
45                      initialCenter={region}
46                      center={center}
47                  >
48                      {this.render_markers(markers)}
49                  </Map>
50              </div>
51          )
52      }
53  }
54  export default GoogleApiWrapper({
55      apiKey: ("AIzaSyB0ONEztpZmzGnpsvCjf2S4hQjjTnyehzA")
56  })(MapGoogle)
```

Código 3.64 Componente `Map` de `google-maps-react`

Para conectar el `apiKey` al componente `Map`, la librería provee el componente `GoogleApiWrapper` (línea 54 a 56) el cual recibe el `apiKey` que se obtiene desde Google *Maps JavaScript API* para asociarla con la clase que contiene al componente `Map`.

3.3.9.2.5 Recomendaciones

El administrador debe ser capaz de crear, editar y eliminar una recomendación. Esta tarea se puede realizar desde la pantalla recomendaciones mostrada en la Figura 3.46.

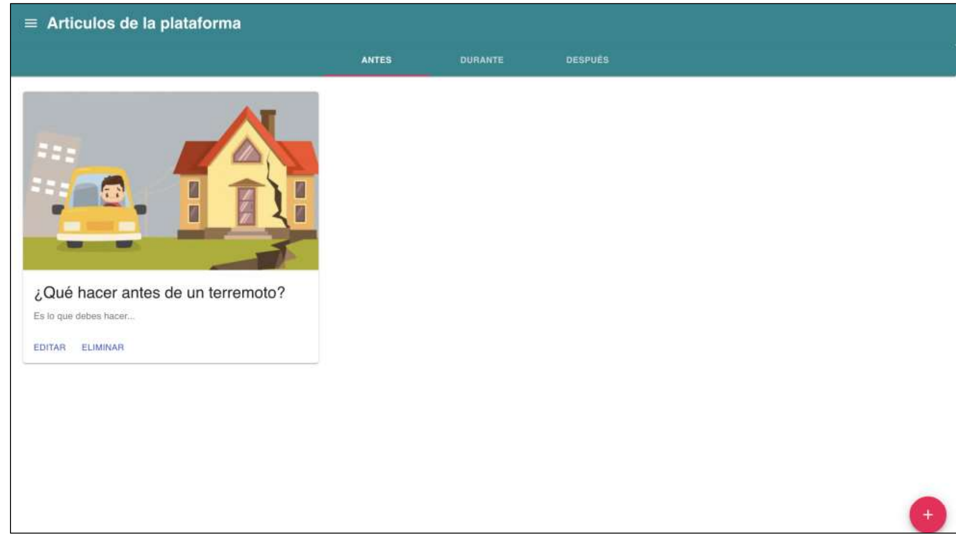

Figura 3.46 Pantalla de administrador artículos de la plataforma

En la Figura 3.47 se muestra la pantalla para la edición y creación de una recomendación, la cual contiene un formulario y la vista previa de esta.

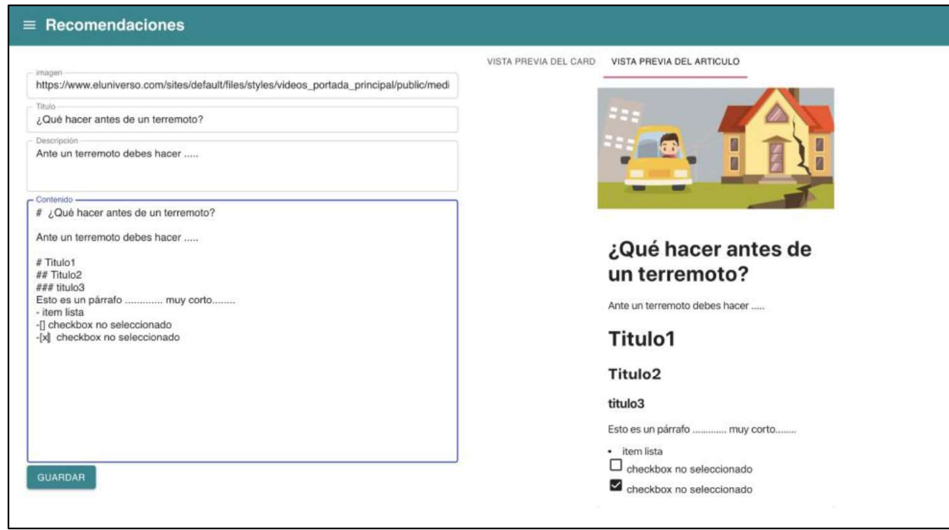

Figura 3.47 Panel de edición y creación de una recomendación

El formulario presenta los siguientes campos:

- Imagen: es la URL de la imagen que se mostrará en el *card* y el encabezado de la recomendación
- Título: es el título de la recomendación que se mostrará en el *card* de la recomendación
- Descripción: es una descripción breve que se mostrará en el *card* de la recomendación
- Contenido: es un texto que define la estructura del contenido de la recomendación

El formato en el que se alberga el contenido de una recomendación en la base de datos es JSON, sin embargo, para evitar que el usuario deba aprender JSON y además cuidar que los campos sean los adecuados se creó un pequeño lenguaje de marcado inspirado en *Markdown*[53]. A continuación, se detallan las etiquetas creadas para el marcado del contenido:

- #: representa un título de primer nivel
- ##: representa un título de segundo nivel
- ###: representa un título de tercer nivel
- -: representa un ítem de una lista no numerada
- -[]: representa un *checkbox*[54] no seleccionado
- -[x]: representa un *checkbox* seleccionado

[53] *Markdown*: es un lenguaje de marcado que facilita la aplicación de formato a un texto.
[54] *Checkbox*: componente de UI con una casilla seleccionable.

Cualquier otro elemento que no cuente con una de las etiquetas definidas previamente se considera como un texto simple o párrafo.

3.3.9.2.6 Configuración de los parámetros del servidor

Para la configuración de los parámetros del servidor se generó la UI mostrada en la Figura 3.44, la cual presenta un formulario con los siguientes parámetros:

- `minMagnitude`: representa la magnitud mínima para notificar a los usuarios de un terremoto
- `offsetTime`: representa el tiempo en milisegundos antes de la fecha actual desde la que se consultan los terremotos
- `requestTime`: representa el intervalo de tiempo de consulta de terremotos
- `waitTime` representa el tiempo de espera de respuesta por parte de los usuarios antes de notificar a sus contactos de su estado

3.3.9.2.7 Navegación

La navegación se realiza por medio de una URL y se implementa con la librería `react-router-dom`. Los componentes de la librería se muestran en el Código 3.65 y se detallan a continuación:

- `BrowserRouter`: es el componente encargado de mantener la UI en sincronía con la URL o `path`, es decir, que cuando se navegue se muestre el recurso adecuado
- `Switch`: presenta el componente `Route` que primero coincida con el `path` ingresado por el usuario
- `Route`: es el componente que representa un recurso de UI dependiendo del `path`

```
15  <BrowserRouter
16    basename='/'
17  >
18    <Switch>
19      <Route exact path="/" component={Login} />
20      <Route exact path="/map" component={Map} />
21      <Route exact path="/articles" component={CategoryArticle} />
22      <Route exact path="/settings" component={Settings} />
23      <Route exact path="/article" component={Article} />
24      <Route exact path="/loading" component={Loading} />
25      <Route exact path="/edit" component={Edit} />
26      <Route component={Layout404} />
27    </Switch>
28
29  </BrowserRouter>
```

Código 3.65 Navegación por pantallas

Un usuario podría ingresar una dirección inexistente por lo que, en la línea 26, se especifica un componente que se mostrará en caso de que el `path` no coincida con ninguna pantalla.

3.3.9.3 Entregable

En este *Sprint* se tiene como entregable una aplicación web la cual permite:

- Autenticar al usuario administrador
- Crear, editar, leer y eliminar recomendaciones;
- Crear, leer y eliminar sitios de interés del mapa colaborativo;
- Configurar los parámetros del servidor.

El ANEXO E contiene el código de la aplicación web y en el ANEXO F se localiza el manual de usuario de las aplicaciones móvil y web.

4 RESULTADOS Y DISCUSIÓN

En esta sección se presentan los resultados de las pruebas realizadas al prototipo, las mismas que ayudaron a validar los requerimientos presentados en el *Product Backlog.*

Primero, se realizan las pruebas de funcionalidad a los servicios implementados, aquí se muestran las peticiones HTTP a los servicios de bases de datos `feed`, `map`, `user` y `server` alojados en Firebase; y las peticiones al servicio de SMS. A continuación, se desarrollaron las pruebas de integración para validar que la aplicación móvil y web funcionen de manera correcta con los servicios del prototipo. Finalmente, se realizó una encuesta a 20 usuarios con el objetivo de verificar con el usuario final la funcionalidad del prototipo, identificar errores no observados en las pruebas previas y recoger recomendaciones.

Al culminar este capítulo, se presenta un resumen de los errores corregidos con base en las recomendaciones dadas por el *Product Owner* durante la fase de implementación y los resultados de las encuestas realizadas a los usuarios.

4.1 PRUEBAS DE FUNCIONALIDAD DE LOS SERVICIOS

Las pruebas de esta sección se realizaron con la herramienta Postman.

4.1.1 PRUEBA DEL SERVICIO DE BASE DE DATOS FEED

En esta prueba se verificó que se puede obtener, a través de HTTP, los datos de las recomendaciones almacenadas en el servicio de base de datos `feed`. La Figura 4.1 muestra la captura de pantalla de la respuesta obtenida en formato JSON de la petición GET, realizada al servicio.

Figura 4.1 Prueba del servicio de base de datos `feed`

4.1.2 PRUEBA DEL SERVICIO DE BASE DE DATOS MAP

Esta prueba tiene como objetivo validar que se puede obtener la información de los lugares de interés guardados en el servicio de base de datos `map`. La prueba se realizó con una

petición GET al servicio. En la Figura 4.2 se puede apreciar la captura de pantalla de los lugares de interés obtenidos con la petición realizada.

Figura 4.2 Prueba del servicio de base de datos `map`

4.1.3 PRUEBA DEL SERVICIO DE BASE DE DATOS USER

La prueba verificó que el servicio de base de datos `user` funciona correctamente. La Figura 4.3 muestra una captura de pantalla donde se aprecia la respuesta de una petición GET, que recupera los usuarios almacenados en la base datos del servicio.

Figura 4.3 Prueba del servicio de base de datos `user`

4.1.4 PRUEBA DEL SERVICIO DE BASE DE DATOS SERVER

En esta prueba se verificó que se pueden obtener los datos almacenados en el servicio de base de datos `server`. En la Figura 4.4 se muestra captura de pantalla de la repuesta obtenida de la petición GET realizada al servicio.

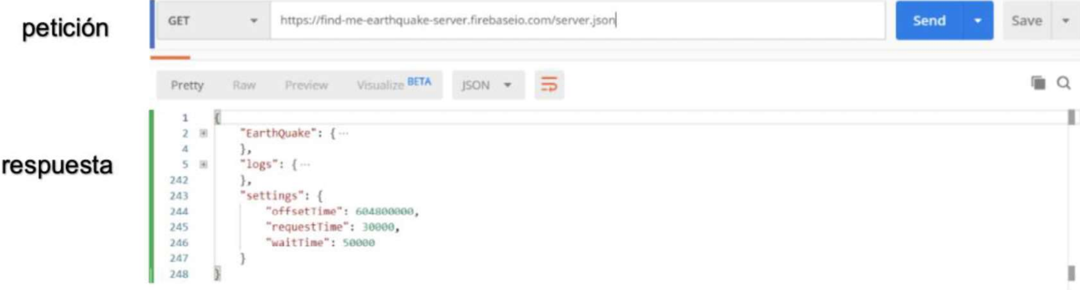

Figura 4.4 Prueba del servicio de base de datos `server`

4.1.5 PRUEBA DEL SERVICIO DE SMS

En esta prueba se verificó que se puede leer, enviar y eliminar un SMS, desde el módem GSM, utilizando peticiones HTTP al servicio implementado.

Para verificar que se puede leer un SMS del módem GSM, primero se envió un SMS desde un celular hacia el módem. La Figura 4.5 muestra la captura de pantalla del SMS enviado en color verde.

Figura 4.5 Captura de pantalla de SMS enviado/recibido en el celular

Luego se realizó una petición GET al servicio para obtener los SMS de la bandeja de entrada del módem. En la Figura 4.6 se puede apreciar que la respuesta obtenida corresponde al SMS recibido desde el celular.

petición

GET 192.168.100.203:3000/sms Send

Pretty Raw Preview Visualize BETA JSON

respuesta


```
{
    "status": "success",
    "request": "getSimInbox",
    "data": [
        {
            "sender": "593987868889",
            "message": "Es un mensaje de prueba de lectura de la bandeja del módem GSM",
            "index": 1,
            "dateTimeSent": "2019-11-04T21:46:16.000Z",
            "header": {
                "encoding": "16bit",
                "smsc": "593995997067",
                "smscType": "INTERNATIONAL",
                "smscPlan": "ISDN"
            }
        }
    ]
}
```


Figura 4.6 Prueba del Servicio de SMS método GET

Para verificar que el servicio puede enviar un SMS, desde el módem GSM, se realizó una petición POST. En la Figura 4.7 se puede apreciar la captura de pantalla de la petición realizada. En el cuerpo de la petición se muestra el destinario y contenido del mensaje, mientras que, la repuesta de la petición indica que el SMS fue enviado con éxito. El SMS recibido en el celular desde el módem se puede ver en la Figura 4.5.

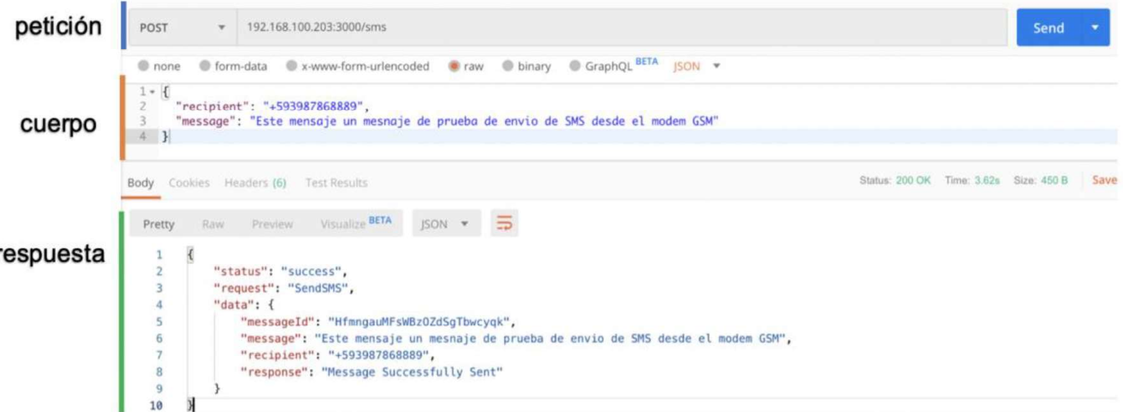

Figura 4.7 Prueba del Servicio de SMS método POST

Por último, para verificar que se puede eliminar un SMS de la bandeja de entrada del módem se realizó una petición DELETE al servicio.

En esta prueba se borró el mensaje que se recibió previamente en la prueba anterior. En la Figura 4.8 se puede observar la repuesta exitosa a la petición realiza.

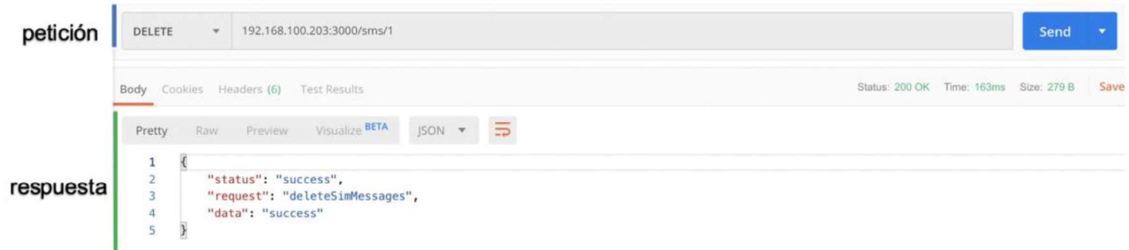

Figura 4.8 Prueba del Servicio de SMS método DELETE

4.2 PRUEBAS DE INTEGRACIÓN

En esta sección se presentan las pruebas de integración de los servicios implementados con la aplicación web, móvil y servidor.

Las funcionalidades de las aplicaciones son verificadas con base en los requerimientos identificados en la sección 2.1.5.

4.2.1 AUTENTICACIÓN DEL USUARIO POR NÚMERO CELULAR

En esta prueba se verificó que un usuario pueda autenticarse con su número de teléfono celular desde la aplicación móvil.

Primero, al usuario se le presenta un formulario, donde deberá seleccionar su país e ingresar su número de celular de acuerdo con el formato del país. Cabe mencionar que para esta prueba solo está disponible Ecuador.

En la Figura 4.9 se puede apreciar el formulario con el número de celular que se autenticará en esta prueba.

Figura 4.9 Pantalla de autenticación

Si el número ingresado no coincide con el formato de país se le mostrará un mensaje similar al de la Figura 4.10.

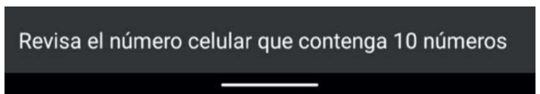

Figura 4.10 Mensaje de numero ingresado incorrectamente

Luego de ingresar un número con el formato correcto, el usuario recibirá un SMS con el código de confirmación, similar al que se observa en la Figura 4.11.

Si el usuario no recibe el código, por haber ingresado un número erróneo, tendrá la posibilidad de cambiar su número de celular regresando a la pantalla de la Figura 4.9.

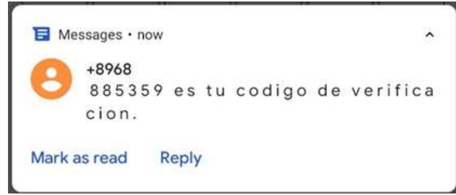

Figura 4.11 Mensaje con el código de verificación

El usuario deberá ingresar el código recibido en la pantalla de verificación. Esta pantalla se puede apreciar en la Figura 4.12

Figura 4.12 Formulario de ingreso de código de verificación

Si el código ingresado ha sido el correcto se guardará el número de celular del usuario en Firebase, en el apartado de autenticación, como se muestra en la Figura 4.13

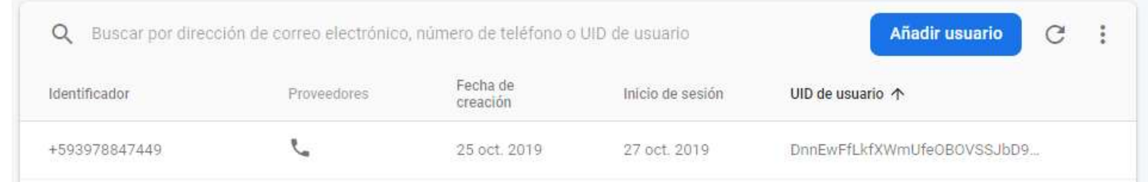

Figura 4.13 Usuario autenticado por número celular en Firebase

En caso de que el código ingresado sea incorrecto se le mostrará al usuario un mensaje, como el que se aprecia en la Figura 4.14.

Figura 4.14 Mensaje de error de código ingresado

4.2.2 REGISTRO DEL USUARIO

Esta prueba tiene como objetivo verificar el registro y actualización de la información del usuario desde la aplicación móvil en el servicio de base de datos `user`.

El usuario podrá ir agregando su información personal y su foto de perfil en la pantalla de registro. La Figura 4.15 muestra la captura de pantalla de la información del usuario agregada para esta prueba.

Figura 4.15 Pantalla de registro de datos

Durante el proceso de registro, la aplicación le solicitará al usuario los permisos correspondientes para acceder a los módulos de cámara, mensajes y GPS. En la Figura 4.16 se puede apreciar una captura de pantalla de la solicitud de los servicios.

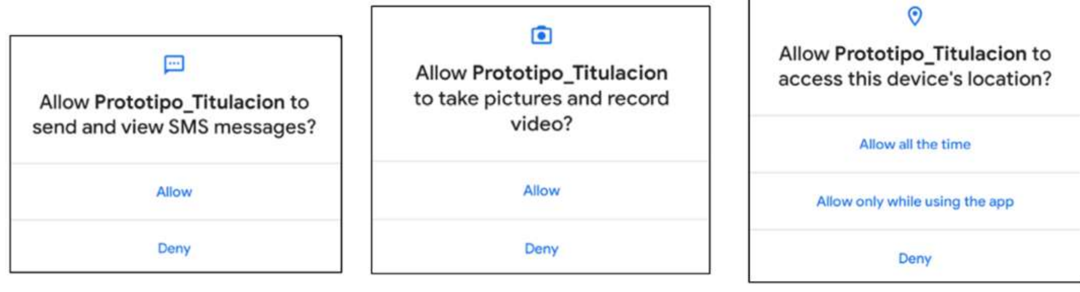

Figura 4.16 Captura de pantalla de permisos solicitados al usuario

El registro continúa con la selección de contactos a los que el usuario quiere que se le notifique su estado en caso de un terremoto. La Figura 4.17 muestra una captura de pantalla de los usuarios agregados para el registro de prueba.

Figura 4.17 Pantalla de selección de contactos

Una vez culminado el registro se almacenará en el servicio de base de datos `user` los datos ingresados por el usuario. En la Figura 4.18 se puede apreciar la captura de pantalla de los datos registrados durante esta prueba

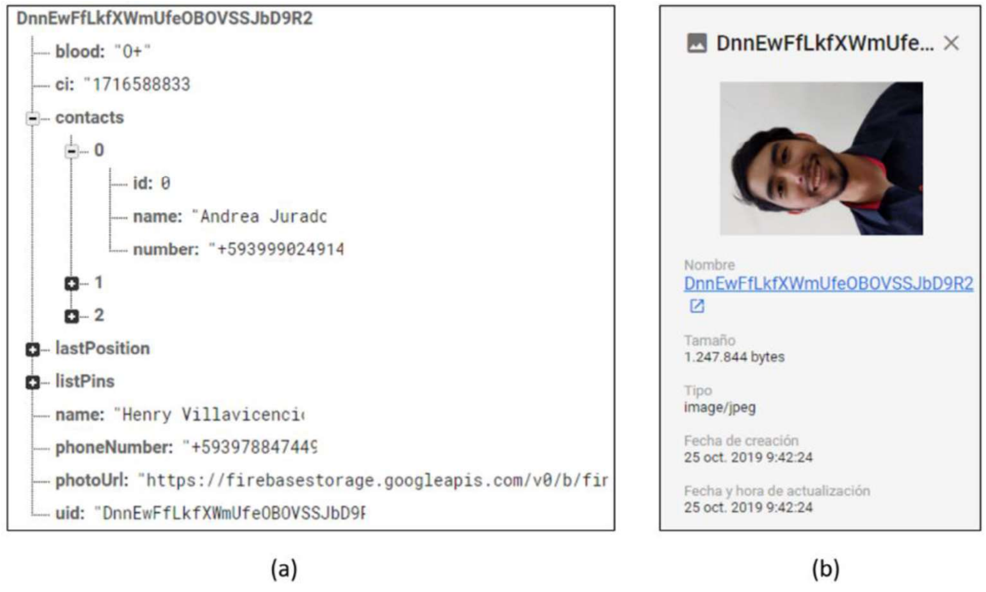

Figura 4.18 Captura de pantalla del registro de prueba

4.2.2.1 Actualización de la información de usuario

El usuario podrá visualizar su información personal en su perfil. Desde este apartado el usuario podrá actualizar su foto de perfil, datos personales, cambiar su número de teléfono y configurar el tiempo para registrar su posición.

En la Figura 4.19 se muestra la captura de pantalla de la actualización del nombre de usuario.

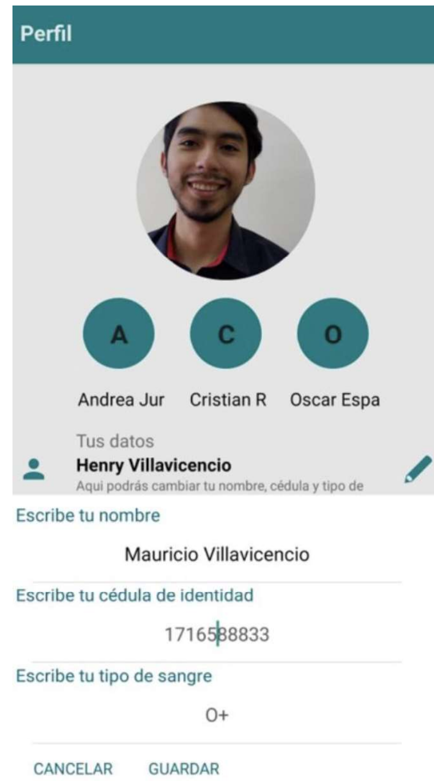

Figura 4.19 Pantalla de perfil de usuario

Cuando los datos se hayan actualizados al usuario se le presentara un mensaje similar el de la Figura 4.20.

Figura 4.20 Mensaje de actualización de información de usuario

En la Figura 4.21 se muestran los campos de datos actualizados en el servicio de base de datos `user`.

Figura 4.21 Actualización de nombre de usuario

4.2.3 OBTENCIÓN DE LA POSICIÓN DEL USUARIO

La aplicación móvil recoge la posición del usuario cada 15 minutos por defecto y la envía al servicio de base de datos `user`.

Para esta prueba se desactivo el envío de la posición y se verificó que el servicio no recibiera la ubicación. Luego se configuró un intervalo de 30 minutos y se comprobó que el servicio recibiera la ubicación correcta del usuario.

En la Figura 4.22 se aprecia la captura de pantalla de la última posición recibida en el servicio durante la prueba.

```
lastPosition
    lastconnection: "09-11-2019 10:08:44"
    latitude: "-0.25639935"
    longitude: "-78.50397412"
```

Figura 4.22 Última posición del usuario

4.2.4 LECTURA DE RECOMENDACIONES *OFFLINE*

Esta prueba tiene como objetivo verificar que las recomendaciones de la aplicación móvil se muestran a pesar de no tener Internet.

Durante esta prueba lo que se hizo es poner el dispositivo móvil en modo avión o sin conexión (*offline*), cerrar la aplicación y luego de unos minutos volver a ingresar a una recomendación. La Figura 4.23 muestra la captura de pantalla de la lectura de una recomendación con el dispositivo en modo avión.

Figura 4.23 Recomendaciones en modo *offline*

4.2.5 MAPA COLABORATIVO

Durante esta prueba se verificaron las funcionalidades del mapa colaborativo con el que cuenta la aplicación móvil. El flujo de las acciones realizadas se detalla a continuación.

Con conexión a Internet, un usuario, puede ver el mapa colaborativo como se aprecia en la captura de pantalla de la Figura 4.24 (a), donde podrá seleccionar un *pin* para ver más información del sitio, como muestra la Figura 4.24 (b).

De forma adicional si se selecciona un *pin* se tendrán las opciones de:

- Ver el sitio en la aplicación de Google Maps del dispositivo, como se observa en la Figura 4.24 (c)
- Ver la ruta que puede seguir desde su posición actual hasta el sitio de interés, en Google Maps, como se aprecia en Figura 4.24 (d)

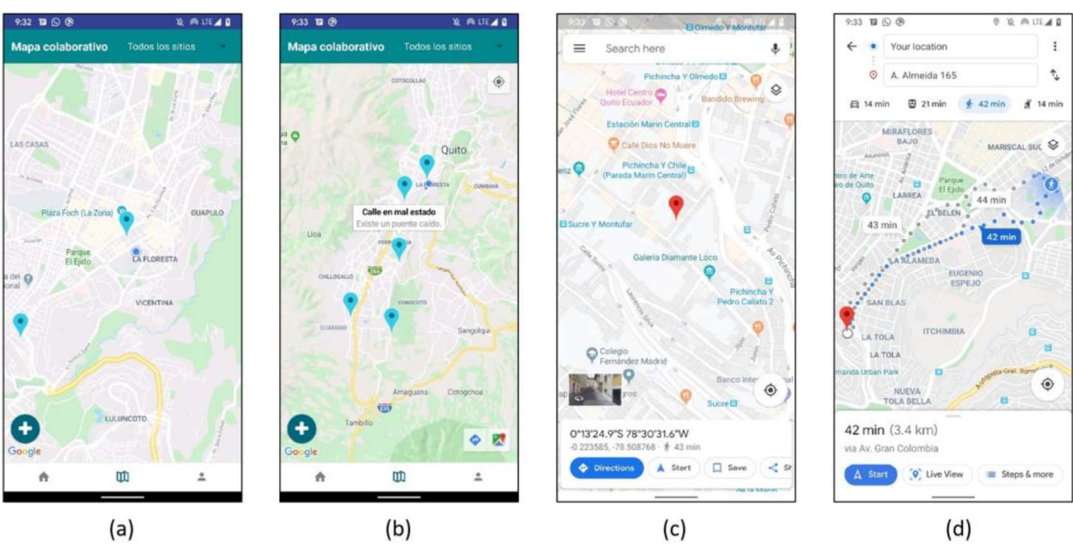

Figura 4.24 Pantallas del mapa colaborativo

4.2.5.1 Filtro de lugares por categoría

El usuario podrá filtrar los pines, de los lugares de interés, por categoría y mostrarlos en el mapa colaborativo. La Figura 4.25 muestra la captura de los pines de la categoría albergue filtrados.

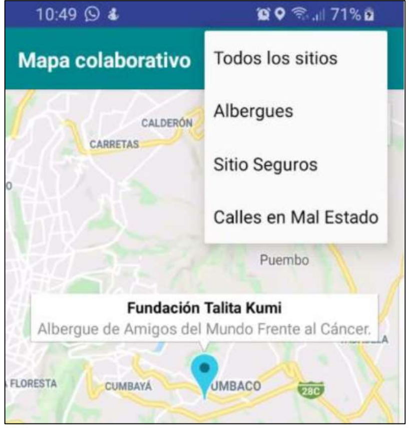

Figura 4.25 Filtro de sitios seguros

4.2.5.2 Agregar un lugar de interés

Para añadir un nuevo *pin* al mapa se deberá presionar sobre el botón “+” y completar un formulario con sus datos. La Figura 4.26 muestra la captura de pantalla del formulario del *pin* que se agregó durante la prueba.

Figura 4.26 Formulario para añadir información de la ubicación seleccionada

Todos los campos del formulario se deben llenar, sino se lo hace, se mostrará un mensaje de alerta similar al de la Figura 4.27.

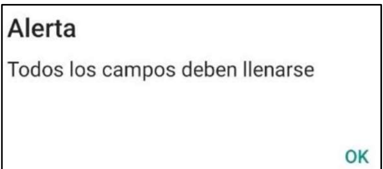

Figura 4.27 Mensaje de alerta de campos sin llenar

Una vez se ha terminado el proceso, se agregarán los datos del *pin* al servicio de base de datos `map`. La Figura 4.28 muestra la captura de pantalla del sitio agregado en el servicio.

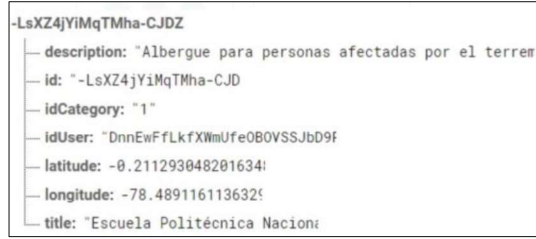

Figura 4.28 Sitio agregado en el servicio de base de datos `map`

4.2.5.3 Eliminar un sitio de interés

La acción de eliminar un *pin* se realiza tocando su descripción. Un usuario puede eliminar un *pin* siempre y cuando el que seleccione haya sido agregado por el. Si el *pin* seleccionado fue agregado por el usuario se mostrará un mensaje de confirmación para eliminar el *pin* similar al de la Figura 4.29.

Figura 4.29 Mensaje de confirmación para eliminar el *pin*

En caso de que el *pin* no haya sido agregado por el usuario se le mostrará un mensaje como el de del Figura 4.30 indicando que no posee los permisos para eliminar el *pin.*

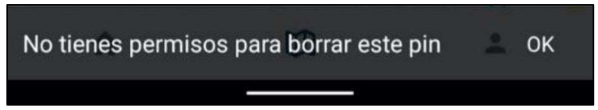

Figura 4.30 Mensaje sin permiso para eliminar el *pin*

4.2.6 NOTIFICACIÓNDE TERREMOTOS

Durante esta prueba se comprobó que el prototipo es capaz de notificar al usuario en caso de que detecte que ocurrió un terremoto.

Para que los usuarios reciban una notificación de que ocurrió un terremoto, se simuló un terremoto falso desde el servidor. En la Figura 4.31 se puede apreciar una captura de pantalla de la alerta que se mostró al usuario durante la prueba.

Figura 4.31 Captura de pantalla de la notificación de terremoto

El usuario tiene dos opciones para informar su estado: `"estoy bien"` y `"necesito ayuda"`. Esta respuesta enviará el estado del usuario en un SMS al servidor, el cual reenviará un SMS a sus contactos registrados de acuerdo con su respuesta.

Cuando un usuario informe que está bien sus contactos recibirán un SMS similar al mostrado en la Figura 4.32.

Figura 4.32 Captura de pantalla del estado del usuario cuando informó

Si el usuario informa que necesita ayuda sus contactos recibirán un SMS similar al de la Figura 4.33

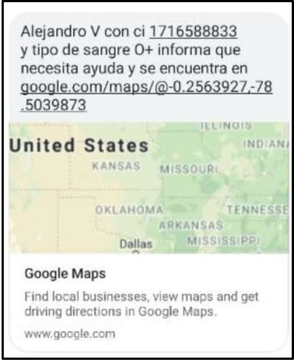

Figura 4.33 Captura de pantalla del estado del usuario cuando informó su estado

En caso de que un usuario no pueda notificar de su estado. El prototipo enviará automáticamente un SMS similar al presentado la Figura 4.34.

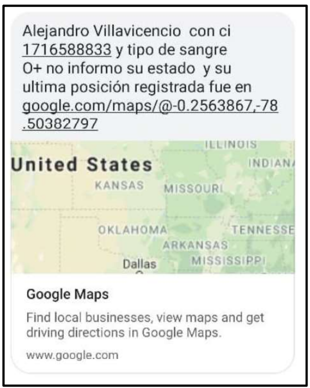

Figura 4.34 Captura de pantalla del estado del usuario cuando no informó su estado

4.2.7 ADMINISTRACIÓN DE CONTENIDOS Y CONFIGURACIÓN DEL PROTOTIPO

4.2.7.1 AUTENTICACIÓN

En esta prueba verificó que el usuario administrador pueda ingresar al panel de administración del prototipo.

El administrador debe ingresar sus credenciales en la página de *login* en la URL https://find-me-earthquake.firebaseapp.com/. Una captura de pantalla de esta página se muestra en la Figura 4.35.

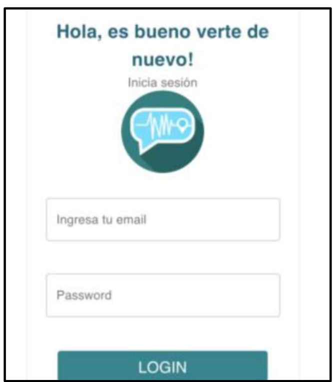

Figura 4.35 Pantalla de *login*

Si el administrador se equivoca en la información ingresada se le informará con un mensaje similar al de la Figura 4.36.

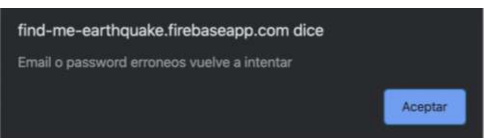

Figura 4.36 Mensaje de error en *email* o *password*

Si el *login* es correcto se le redireccionará al panel de administración. La Figura 4.37 muestra una captura de pantalla de este panel.

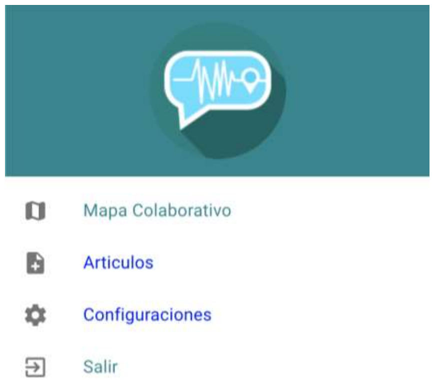

Figura 4.37 Panel de administración

4.2.7.2 ADMINISTRACIÓN DEL MAPA COLABORATIVO

Durante esta prueba se verificó si el usuario administrador pueda agregar o eliminar un *pin* sin ninguna restricción.

El mapa colaborativo se presenta en la Figura 4.38 y es similar al que se presenta en la aplicación móvil.

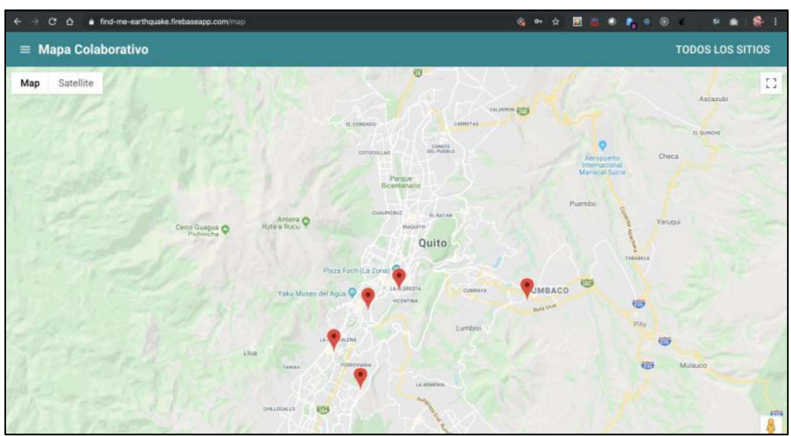

Figura 4.38 Pantalla del mapa colaborativo

El usuario deberá llenar un formulario similar al de la Figura 4.39, para agregar un nuevo *pin* al mapa.

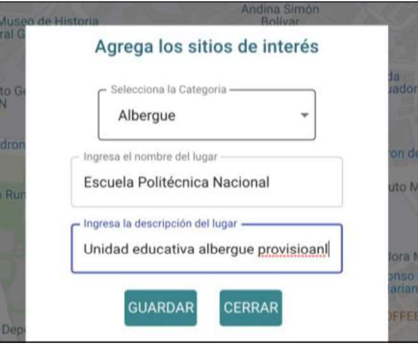

Figura 4.39 Formulario para agregar un sitio de interés

En la Figura 4.40 se muestra la captura de pantalla del *pin* de prueba agregado en el servicio de base de datos `map`.

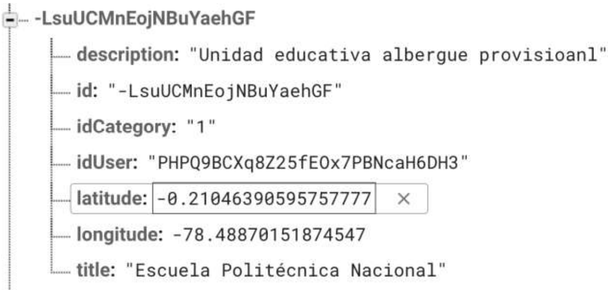

Figura 4.40 *Pin* agregado por el administrador en el servicio de base de datos `map`

El administrador tiene el permiso para borrar cualquier *pin* dentro del prototipo. Deberá seleccionar el sitio y eliminarlo presionando en el botón `"Borrar"` que se muestra en la Figura 4.41.

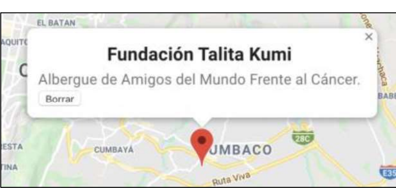

Figura 4.41 Detalles del pin seleccionado

4.2.7.3 ADMINISTRACIÓN DE RECOMENDACIONES

En esta prueba se verificó si el administrador pueda crear, actualizar y eliminar las recomendaciones del prototipo.

Para crear o actualizar una recomendación al usuario se le presentará un formulario similar al de la Figura 4.42.

Figura 4.42 Captura de pantalla de la redacción de una recomendación

Las recomendaciones se actualizaron de forma automática a los usuarios de la aplicación móvil. En la Figura 4.43 se muestra la captura de pantalla de la aplicación móvil, en la que se observa la recomendación creada en la prueba.

Figura 4.43 Nueva recomendación en la aplicación móvil

El administrador también puede eliminar una recomendación, cuando se realice esta acción se le mostró un mensaje similar al de la Figura 4.44 informando que se ha eliminado la recomendación.

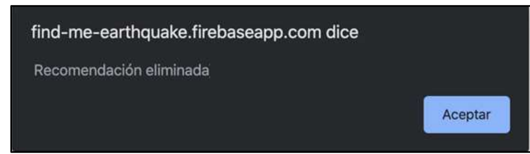

Figura 4.44 Mensaje de eliminación de la recomendación

4.2.7.4 CONFIGURACIÓN DEL SERVIDOR

Para esta prueba se configuraron los parámetros que usa el servidor para enviar las notificaciones con los siguientes valores: tiempo de espera de las respuestas del usuario en 20 minutos, tiempo de previo para la consulta de terremotos en 15 minutos, intervalo de tiempo para la consulta de terremotos en 10 minutos y magnitud mínima de un terremoto en 4.5.

La Figura 4.45 muestra la captura de pantalla con los valores de configuración para la prueba.

Figura 4.45 Captura de pantalla de las configuraciones del servidor

La Figura 4.46 muestra la captura de pantalla de la configuración del servidor almacenada en Firebase.

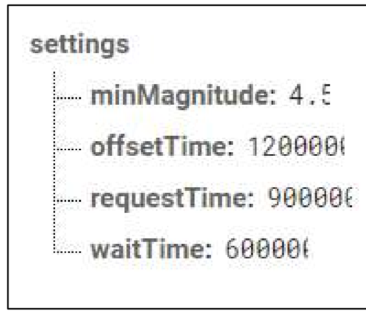

Figura 4.46 Captura de la base de datos del servidor

4.3 PRUEBAS DE VALIDACIÓN

Las pruebas de validación del prototipo se realizaron a través de una encuesta que se realizó con 20 personas que utilizaron la aplicación móvil. La encuesta se incluye en el ANEXO G.

Las preguntas de la encuesta contaban con las opciones: si o no. La Figura 4.47 muestra los resultados obtenidos en la encuesta.

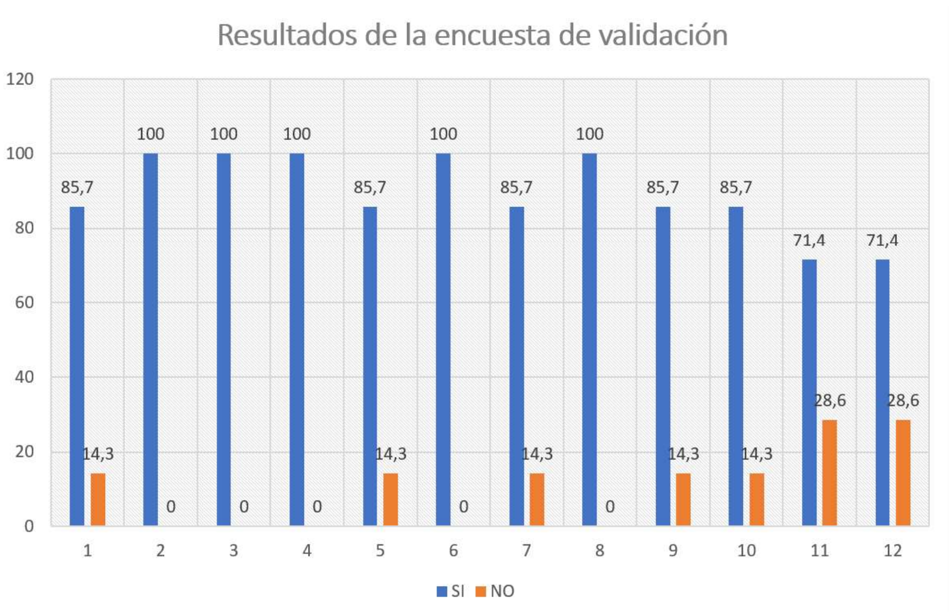

Figura 4.47 Resultados de la encuesta de validación

La pregunta uno: ¿Recibió el código de verificación para acceder al registro? sirve para comprobar si el usuario recibió el código de verificación, donde el 85.7% recibió el código de verificación y un 14.3% no lo recibió.

La pregunta dos: ¿La aplicación te permitió agregar la foto de perfil en el registro? permite comprobar si el usuario pudo agregar su foto de perfil. El resultado muestra que el 100% de los encuestados pudo agregar su foto.

En la pregunta tres ¿La aplicación te permitió agregar, cambiar y eliminar sus contactos? Los resultados muestran que el 100%. de los encuestados pudieron realizar estas acciones.

En la pregunta cuatro ¿La aplicación te pidió los permisos para acceder a la cámara, contactos, mensajes de texto y GPS del dispositivo? El 100% de los encuestados informaron que la aplicación le solicitó los permisos.

En la pregunta cinco ¿La aplicación te permitió revisar las recomendaciones de manera *offline*? El 85.7% de los usuarios encuestados pudieron revisar las recomendaciones de manera *offline*, por otro lado, solo un 14.3% no pudo.

En la pregunta seis ¿La aplicación te permitió agregar sitios de interés en el mapa colaborativo? el 100% de los encuestados pudo agregar sitios de interés.

En la pregunta siete ¿La aplicación te permitió eliminar sitios de interés que tú agregaste en el mapa colaborativo? el 85.7% de los encuestados pudo eliminar sus sitios de interés, mientras que, un 14.3% no lo pudo hacer.

En la pregunta ocho ¿La aplicación te permitió cambiar tu foto de perfil? El 100% de los encuestados pudo cambiar su foto de perfil.

En la pregunta nueve ¿La aplicación te permitió cambiar tus datos personales y número de teléfono? El 85.7% de los encuestados pudo cambiar sus datos, mientras que, un 14.3% no lo pudo hacer.

En la pregunta diez ¿La aplicación te permitió cerrar sesión de manera correcta? El 85.7% de los encuestados pudo cerrar sesión, mientras que, un 14.3% no lo pudo hacer.

En la pregunta once ¿La aplicación te informó de un terremoto ocurrido y te pregunto sobre tu estado? al 71.4% de los encuestados se les informó del terremoto, mientras que, al 28.6% no se informó.

En la pregunta doce ¿Tus contactos recibieron tu estado? El 71.4% de los encuestados manifiesta que sus contactos recibieron su estado mediante un SMS y el 28.6% no.

Finalmente, la pregunta trece ¿Ocurrió algún otro error en la ejecución de la aplicación? se obtuvieron las siguientes respuestas:

- No recibieron todos mis contactos el mensaje con mi estado
- La aplicación pedía registrarse cada vez que se abría, pero luego funcionó
- No recibí la notificación del terremoto
- No se mostró la pantalla de alerta preguntando su estado

Luego de indagar los resultados se determinó que:

- El 14.3% de los encuestados que presentaron problemas con la aplicación móvil, corresponde a usuarios que tenían una versión de Android *Marshmallow* (Android 6.1) o inferiores, debido a que la librería `react-native-firebase` presenta problemas con estas versiones de Android.
- El porcentaje de encuestados a los que no se les informó correspondía a usuarios que estaban fuera de área de notificación o usuarios que no pudieron realizar el registro correctamente debido al problema anterior.

4.4 CORRECIÓN DE ERRORES

En los diferentes *Sprints* se encontraron errores, los cuales se resolviendo a medida que se desarrollaba el prototipo.

En la Tabla 4.1 se encuentran las correcciones del *Sprint* 1.

Tabla 4.1 Revisión del *Sprint* 1

Error	Corrección
Problemas para importar un archivo JSON en Firebase.	Cambiar los archivos que se importaron a Firebase a un formato JSON estricto.
Firebase no permite crear varios servicios de base de datos en su versión gratuita.	Agregar un método de pago en Firebase para habilitar varios servicios de bases de datos.

En la Tabla 4.2 se encuentran las correcciones del *Sprint* 2.

Tabla 4.2 Revisión del *Sprint* 2

Error	Corrección
No se puede realizar la navegación entre pantallas en la aplicación móvil.	Agregar el componte `AppContainer` de `react-navigation` en la aplicación para habilitar la navegación.
React Native no carga la aplicación y muestra el error `Unable to load script from assets'index.android.bundle’`.	Agregar las variables de entorno de desarrollo Android al computador.

En la Tabla 4.3 se encuentran las correcciones del *Sprint* 3.

Tabla 4.3 Revisión del *Sprint* 3

Error	Corrección
La selección de foto de perfil devuelve un objeto `undefined` antes de que el usuario haya seleccionado una foto.	Crear una promesa para manejar la selección de la foto de perfil.

En la Tabla 4.4 se encuentran las correcciones del *Sprint* 4.

Tabla 4.4 Revisión del *Sprint* 4

Error	**Correcciones**
El comando `react-native link react-native-firebase` presenta problemas para enlazar la librería de Firebase con el proyecto Android de React Native.	Agregar las dependencias y paquetes necesarios al proyecto Android de forma manual.

En la Tabla 4.5 se encuentran las correcciones del *Sprint* 5.

Tabla 4.5 Revisión del *Sprint* 5

Error	**Corrección**
La aplicación se cierra porque tiene datos `undefined` al obtenerlos desde el servicio de bases de datos.	Agregar el *middleware* Redux Thunk para que el *store* pueda manejar datos de métodos asíncronos.
La aplicación da un error al persistir los datos ya que las librerías `redux-persistent` e `immutable` no son compatibles.	Incluir la librería `redux-persistent-immutable`.

En la Tabla 4.6 se encuentran las correcciones del *Sprint* 6.

Tabla 4.6 Revisión del *Sprint* 6

Error	**Corrección**
La aplicación no se inicia debido a que los módulos de Firebase y Google Maps no son compatibles.	Excluir los módulos `play-services-base` y `play-services-maps` de Firebase en la librería de mapas en el archivo `build.gradle` a nivel de `app` del proyecto.
La aplicación se cierra cuando el GPS del dispositivo se encuentra apagado.	Agregar una ubicación por defecto en la que se mostrará el mapa.

Error	Corrección
El botón para ubicar al usuario automáticamente dentro del mapa no aparece.	Modificar los estilos del componente `MapView`.

En la Tabla 4.7 se encuentran las correcciones del *Sprint* 7.

Tabla 4.7 Revisión del *Sprint* 7

Error	Corrección
No se ejecuta la escucha de SMS.	Activar los permisos necesarios para el manejo de SMS en el archivo `AndroidManifest.xml`.
	Actualizar el `BroadcastReciver` a la versión sugerida por en el API 28 de Android.
	Realizar un módulo nativo para obtener la posición.
Al invocar la función `setTimeJob` desde el código JavaScript se muestra el error `is not a function`.	Registrar el paquete `SharedPreferencesPackager` en el archivo `MainApplication.java`.

En la Tabla 4.8 se encuentran las correcciones del *Sprint* 8.

Tabla 4.8 Revisión del *Sprint* 8

Error	Corrección
El módem GSM no funciona correctamente.	Agregar una alimentación eléctrica externa para la Raspberry Pi y módem GSM.
La comunicación serial entre la Raspberry Pi y el módem GSM no se establece.	Habilitar la comunicación serial en los GPIO[55] la Raspberry Pi que por defecto viene desactivada.

[55] GPIO (*General Purpose Input/Output)*: pines que se pueden usan para conectar sensores u otros dispostivos de hardware en la Raspberry Pi.

En la Tabla 4.9 se encuentran las correcciones del *Sprint* 9.

Tabla 4.9 Revisión del *Sprint* 9

Error	Correcciones
`InfoWindow` transforma los componentes que muestra en HTML, sin incluir sus métodos o funciones, por lo cual el botón "`borrar`" no borraba los *pins* en el mapa colaborativo.	Agregar un `id` al botón "`borrar`" al cual se referencio la función borrar.

A continuación, se presenta la Tabla 4.10 que muestra los errores detectados por los usuarios en la encuesta de validación.

Tabla 4.10 Errores detectados por los usuarios

Error	Correcciones
No recibieron todos mis contactos el mensaje con mi estado.	No todos los contactos agregados contaban con el código de país, por lo cual se codifico un método que corrige el formato del número celular antes de subirlo a Firebase.
La aplicación pedía registrarse cada vez que se abría, pero luego funcionó.	No se aplicó una corrección debido a que no `react-native-firebase` ya no da soporte a las versiones anteriores de Android donde se reportó el problema.
No se mostró la pantalla de alerta preguntando su estado.	No se aplicó una corrección debido a que el error fue causado por restricciones de en los dispositivos con la versión de Android 10 o de la marca Xiaomi.
La posición del usuario enviada en el SMS, a sus contactos, estaba alejada varios metros de la posición real del	Se aumentó el número de decimales de la posición que se enviaban en el SMS.

Error	Correcciones
usuario debido a que el SMS enviaba solo dos decimales de la posición.	

La Tabla 4.11 muestra los cambios realizados durante el desarrollo del prototipo.

Tabla 4.11 Errores detectados por los usuarios

Cambios
Para que el administrador pueda visualizar las acciones realizadas en el servido se agregó un apartado de *logs* en aplicación web.
Se cambió el componente `input` del tipo de sangre a un componente de selección.
Para el registro del usuario la imagen de perfil no debe ser un campo requerido.
Se ajustaron los estilos en los títulos y textos en las recomendaciones.
Se cambio la configuración por defecto del envió de la posición de desactivada a cada 15 minutos.

5 CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- El objetivo del presente Trabajo de Titulación fue desarrollar un prototipo que vía SMS notifica el estado de las personas después de un terremoto. El prototipo está conformado por: un módulo de gestión de datos y un módulo de notificaciones. El desarrollo se basó en microservicios, por lo que, el módulo de gestión de datos se dividió en: servicios de bases de datos y el módulo de notificaciones en servicio de SMS y servicio de consulta de terremotos.
- Las encuestas iniciales se realizaron a 20 personas para generar historias de usuario y obtener requerimientos funcionales y no funcionales de la aplicación Android. Al tabular la encuesta se identificó que se necesita de un administrador que gestione los contenidos y configuraciones del prototipo.
- Para gestionar los contenidos de recomendaciones y lugares de interés del prototipo se implementó una aplicación web con React. Lo que permitió acelerar el desarrollo, ya que la gran mayoría de código escrito en React Native, para la aplicación móvil, pudo reutilizarse en la versión web.
- Se eligió React para la creación de las aplicaciones móvil y web, porque agiliza el desarrollo de estas, ya que permite reutilizar el código. Además, reduce los costos de desarrollo, pues no es necesario contratar un profesional experto en cada plataforma.
- React Native se eligió porque brinda la flexibilidad de integrar código JavaScript con código nativo, en este caso el lenguaje de programación Java para Android, que fue necesario para integrar los servicios de obtención de la posición y recepción de SMS implementados en la aplicación móvil.
- Se escogió el servicio de Firebase para la autenticación de un usuario con su número de celular debido a que proporciona esta funcionalidad de manera gratuita con hasta 50 autenticaciones diarias.
- Para la implementación de la aplicación servidor se utilizó la plataforma Raspberry Pi, la cual es una alternativa de prototipado rápido, que permite combinar software y módulos de hardware como el módem GSM utilizado en este prototipo.
- Para el intercambio de información entre servicios se utilizó JSON, de manera particular el servicio de consulta de terremotos usó una variante de JSON,

denominada GeoJSON, el cual es un formato estándar para intercambio de datos geográficos.

- El utilizar microservicios para la arquitectura del prototipo permitirá en un futuro reutilizar los componentes desarrollados en otras aplicaciones, por ejemplo, el servicio de SMS implementado podrá ser utilizado por una aplicación externa al prototipo.
- Durante el desarrollo de la aplicación servidor se comprobó que los subprocesos de nodeJS permitieron ejecutar paralelamente los servicios de SMS y consulta de terremotos, ya que nodeJS no cuenta con procesamiento multihilo.
- En el desarrollo de las aplicaciones con React se presentó la dificultad de manejar los estados entre componentes, debido a esto se optó por utilizar la librería Redux, ya que facilita el manejo de los estados de los componentes y brinda una estructura más sencilla y organizada del código.
- Debido a los requerimientos de los usuarios, de mantener los datos de manera *offline,* se utilizaron las librerías `redux-persist-inmutable` y `redux` que permiten tener una copia de los datos sin ninguna conexión en la aplicación móvil.
- Las encuestas finales permitieron detectar los errores de la aplicación Android. Existieron errores en la funcionalidad, además de recomendaciones brindadas de los usuarios con esta información se corrigieron los errores.
- Para tener acceso a los servicios implementados en Firebase, desde las aplicaciones del prototipo, se utilizó el SDK proporcionado por Google. Pues permitió acceder a la API de Firebase mediante métodos simples de utilizar para acceder a los servicios de bases de datos, evitando codificar peticiones HTTP propias.
- Debido a que es impredecible la ocurrencia de un terremoto, para las pruebas se optó por generar un evento falso, el cual puede ser ejecutado desde la aplicación web de administración.
- Para el *hosting* de la aplicación web de administración se eligió Firebase, ya que ofrece un servicio de *hosting* gratuito, evitando costes adicionales en el despliegue de la aplicación.
- Para evitar múltiples consultas a la base de datos información del usuario se cambió la relación entre las entidades `User` y `Contact` por una relación de agregación que permite guardar una entidad dentro de otra.

5.2 RECOMENDACIONES

- El prototipo para la notificación del estado de las personas después de un terremoto implementado en este Trabajo de Titulación solo consulta al servicio de USGS. Pero se recomienda que se agregue o se reemplace con otro servicio de consulta de terremotos.
- El prototipo cuenta con un servidor de SMS que se ejecuta de manera local y utiliza el protocolo HTTP, por lo que se recomienda que se migre a un servidor en Internet y se añada soporte para el protocolo HTTPS, pues el presente Trabajo de Titulación no contemplo el costo de implementación de este protocolo.
- Se recomienda el uso de Cloud Computing de Firebase para alojar la aplicación servidor.
- Este Trabajo de Titulación se desarrolló con la versión Android 9.0 y no consideró la actualización de esta plataforma. Por lo que se recomienda actualizar los métodos y módulos nativos para las versiones futuras.
- El prototipo se diseñó para funcionar en Ecuador, sin embargo, se recomienda extenderlo para más países.
- Para datos que se obtengan mediante peticiones HTTP, de un API o métodos asíncronos, se recomienda usar Redux, combinado con el *middleware* Redux Thunk para el tratamiento de los datos antes de que pasen al *store*.
- Para revisar las estadísticas de terremotos se podría generar un mapa de calor de acuerdo con las respuestas de las personas que han notificado su estado o no dentro de un área afectada.
- Debido a este Trabajo de Titulación se centró en el desarrollo de una aplicación Android no se tiene una versión disponible para iOS. Por lo que se recomienda codificar los módulos nativos en el lenguaje de programación Swift e integrarlos con la aplicación actual.
- En este Trabajo de Titulación se implementó un servicio de SMS. Para futuros Trabajos de Titulación se puede implementar la automatización de algún proceso utilizando este servicio.
- A futuro se podría agregar un mapa sismográfico en tiempo real, dentro de la aplicación web del prototipo.

- Para la depuración durante el desarrollo de aplicaciones que utilizan la librería Redux se recomienda el uso de la extensión para navegadores Redux DevTools, que ayuda a hacer seguimiento de las acciones realizadas en el *store*.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] Gestionderiesgos.gob.ec, «INFORME DE SITUACION N°65 –16/05/2016,» 2016. [En línea]. Available: https://www.gestionderiesgos.gob.ec/wp-content/uploads/downloads/2016/05/Informe-de-situaci%C3%B3n-n%C2%B065-especial-16-05-20161.pdf. [Último acceso: 25 Febrero 2019].

[2] El Universo, «El Universo,» El Universo, 14 04 2019. [En línea]. Available: https://www.eluniverso.com/noticias/2019/04/13/nota/7282394/alta-sismicidad-es-realidad-ecuador. [Último acceso: 2 09 2019].

[3] «Plan de respuesta EC,» Gestionderiesgos.gob.ec, 2018. [En línea]. Available: https://www.gestionderiesgos.gob.ec/wp-content/uploads/downloads/2018/05/Plan-de-Respuesta-EC.pdf.. [Último acceso: 25 Febrero 2019].

[4] J. Matson, «Why It's Better to Text Than Call in a Mass Emergency,» SCIENTIFIC AMERICAN, 17 Abril 2013. [En línea]. Available: https://blogs.scientificamerican.com/observations/why-its-better-to-text-than-call-in-a-mass-emergency/. [Último acceso: 03 Septiembre 2019].

[5] Google, «Firebase,» Google, [En línea]. Available: https://firebase.google.com/docs?hl=es. [Último acceso: 20 Julio 2019].

[6] GSMA, «Towards a Code of Conduct: Guidelines for the Use of SMS in Natural Disasters,» [En línea]. Available: https://www.gsma.com/mobilefordevelopment/wp-content/uploads/2013/02/Towards-a-Code-of-Conduct-SMS-Guidelines.pdf. [Último acceso: 18 Junio 2019].

[7] Doctor Tecno, «Operadoras celulares facilitan SMS tras terremoto en Ecuador,» El Universo, 17 Abril 2016. [En línea]. Available: https://www.eluniverso.com/vida-estilo/2016/04/17/nota/5530938/operadoras-celulares-facilitan-sms-tras-terremoto-ecuador. [Último acceso: 3 Septiembre 2019].

[8] Yate Documentation, «GSM Functionalities,» YateBTS, 2018. [En línea]. Available: https://yatebts.com/documentation/concepts/gsm-functionalities/. [Último acceso: 20 Julio 2019].

[9] S. F. D. S. J. G. Alberto Hernández, «Exploring the Visualization of Schemas for Aggregate-Oriented NoSQL Databases,» Noviembre 2017. [En línea]. Available: http://ceur-ws.org/Vol-1979/paper-11.pdf. [Último acceso: 22 Julio 2019].

[10] B. W. Por Jamie Kurtz, «HTTP VERBS,» de *ASP.NET Web API 2: Building a REST Service from Start to Finish* , APRESS, 2014, pp. 12 - 13.

[11] USGS, «ComCat Documentation - Event Terms,» United States Geological Survey, [En línea]. Available: https://earthquake.usgs.gov/data/comcat/data-eventterms.php. [Último acceso: 18 Julio 2019].

[12] M. Hamedani, «React Virtual DOM Explained in Simple English,» Programming with Mosh , 3 Diciembre 2018. [En línea]. Available: https://programmingwithmosh.com/react/react-virtual-dom-explained/. [Último acceso: 18 Julio 2019].

[13] B. Evkoski, «React Native: What it is and how it works,» Medium, 12 Junio 2017. [En línea]. Available: https://medium.com/we-talk-it/react-native-what-it-is-and-how-it-works-e2182d008f5e. [Último acceso: 18 Julio 2019].

[14] «Understanding Why React Native is the Future of Mobile Apps,» de *React Native - Building Mobile Apps with JavaScript*, Birmingham, Pack Publishing, 2017, pp. 9 - 17.

[15] Flux, «In-Depth Overview,» Flux, [En línea]. Available: https://facebook.github.io/flux/docs/in-depth-overview/. [Último acceso: 18 Julio 2019].

[16] Redux, «Three Principles,» Redux, [En línea]. Available: https://redux.js.org/introduction/three-principles. [Último acceso: 10 Julio 2019].

[17] Redux, «Usage with React,» Redux, [En línea]. Available: https://redux.js.org/basics/usage-with-react. [Último acceso: 10 Julio 2018].

[18] SCRUMstudy, «INTRODUCTION,» de *SCRUM BODY OF KNOWLEDGE (SBOK™ GUIDE)*, Avondale, Arizona , SCRUMstudy™, 2016, pp. 10 - 19.

[19] nodeJS, «ECMAScript 2015 (ES6) and beyond,» nodeJS, [En línea]. Available: https://nodejs.org/es/docs/es6/. [Último acceso: 10 Julio 2019].

[20] Metro, «Concepts,» Metro, [En línea]. Available: https://facebook.github.io/metro/docs/en/concepts. [Último acceso: 10 Julio 2019].

[21] react-native-community, «React Native CLI,» Facebook, [En línea]. Available: https://github.com/react-native-community/cli#documentation. [Último acceso: 10 Julio 2019].

[22] V. S. Code, «Getting Started,» Microsoft, [En línea]. Available: https://code.visualstudio.com/docs. [Último acceso: 10 Julio 2019].

[23] Lunacy, «About Lunacy,» Icons8 Production, [En línea]. Available: https://docs.icons8.com/about/. [Último acceso: 10 Julio 2019].

[24] React, «Native Modules,» React Native, 2019. [En línea]. Available: https://facebook.github.io/react-native/docs/native-modules-ios. [Último acceso: 28 Septiembre 2019].

[25] A. R. Rahul Gaba, «Android custom native module,» [En línea]. Available: https://www.reactnative.guide/16-custom-native-modules/16.1-android-native-module.html. [Último acceso: 29 Julio 2019].

[26] A. Developers, «ComponentName,» Google Developers, [En línea]. Available: https://developer.android.com/reference/android/content/ComponentName. [Último acceso: 30 Septiembre 2019].

[27] MDW web docs, «Introducción a Express/Node,» [En línea]. Available: https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction. [Último acceso: 21 Octubre 2019].

[28] USGS, «API Documentation - Earthquake Catalog,» United States Geological Survey, [En línea]. Available: https://earthquake.usgs.gov/fdsnws/event/1/. [Último acceso: 18 Julio 2019].

[29] D. Neff, «Deriving the Haversine Formula,» The Math Forum, 20 Abril 1999. [En línea]. Available: http://mathforum.org/library/drmath/view/51879.html. [Último acceso: 3 Noviembre 2019].

7 ANEXOS

ANEXO A. Modelo de la encuesta inicial.

ANEXO B. Archivos con los objetos JSON de la base de la base de datos.

ANEXO C. Proyecto de la aplicación móvil.

ANEXO D. Proyecto de la aplicación servidor.

ANEXO E. Proyecto de la aplicación web.

ANEXO F. Manual de usuario de las aplicaciones Android y web.

ANEXO G. Modelo de la encuesta de verificación de funcionalidades.

ORDEN DE EMPASTADO

ORDEN DE EMPASTADO