ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UNA APLICACIÓN MÓVIL SOBRE IOS/ANDROID PARA REPRESENTAR EN UN MAPA EL ÍNDICE DE RADIACIÓN ULTRAVIOLETA PARA UN SECTOR DE QUITO

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

IAN ARIEL BAQUERO DUQUE

DIRECTOR: M.Sc. SANCHEZ ALMEIDA TARQUINO FABIAN

Quito, noviembre 2020

AVAL

Certifico que el presente trabajo fue desarrollado por Ian Ariel Baquero Duque, bajo mi supervisión.

MSc. SANCHEZ ALMEIDA TARQUINO FABIAN

DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Ian Ariel Baquero Duque, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

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

_____

Ian Ariel Baquero Duque

AGRADECIMIENTO

Agradezco a Dios. A mi madre Alexandra por su amor y apoyo incondicional a lo largo de toda mi carrera para cumplir mis metas. A mi padre Pablo por siempre ayudarme a mejorar como profesional. A mi hermano Yabel por su compañía, admiración y apoyo a seguir. A mi abuela María que siempre velar por mi salud y bienestar, además de su apoyo incondicional y su amor. A mis compañeros de carrera por motivarme a competir entre ellos para obtener las mejores calificaciones y mejorar como profesional. A mi director de carrera por darme la oportunidad de desarrollar este Trabajo de Titulación. A todos mis familiares y amigos que conocen mi situación y siempre me han recibido con los brazos abiertos y su apoyo por culminar mi etapa universitaria.

ÍNDICE DE CONTENIDO

RESUMEN

El presente Trabajo de Titulación se centra en el desarrollo de una aplicación móvil híbrida sobre IOS/Android utilizando un servicio web, una base de datos y sensores prototipos ultravioleta (UV); para representar en un mapa el índice de radiación UV para un sector de Quito.

La aplicación móvil es desarrollada con el lenguaje de programación JavaScript, utilizando el framework Ionic para aplicaciones híbridas.

La arquitectura del sistema se divide en una Aplicación Móvil, disponible en las tiendas virtuales para su descarga gratuita, tanto para el sistema operativo IOS, como para Android; Servicios Web y Base de Datos (MongoDB) hospedados en servidores en la nube, y al menos tres sensores prototipos UV diseñados por una empresa local. Los Servicios Web gestionan la capa de datos y lógica de todas las transacciones que se realicen desde la Aplicación Móvil. Por otro lado, la Base de Datos se encarga de proveer la información de las muestras recibidas por los sensores hacia los servicios web. Estos servicios se encuentran desarrollados en el lenguaje de programación JavaScript, específicamente en el entorno de ejecución para JavaScript NodeJS. Para el consumo de los servicios y transacciones con la base de datos, se utiliza peticiones de tipo REST.

Capítulo 1: En este capítulo se revisarán los fundamentos teóricos del proyecto para tener una visión más clara de los temas involucrados.

Capítulo 2: En este capítulo se explicará la metodología ágil a implementar y como esta divide el desarrollo del proyecto en tres etapas: Análisis, Diseño e Implementación.

Capítulo 3: En este capítulo se realizarán las pruebas de funcionamiento, las correcciones de errores y la implementación del test MOS para medir la calidad de experiencia del usuario con este proyecto.

Capítulo 4: En este capítulo se escribirán las conclusiones y recomendaciones una vez finalizado el proyecto.

PALABRAS CLAVE: Radiación UV, Servicio Web, REST, Aplicaciones Híbridas, JavaScript, Ionic, MongoDB.

ABSTRACT

This Degree Project focuses on the development of a hybrid mobile application on IOS / Android using a web service, a database and prototype ultraviolet (UV) sensors; to represent the UV radiation index for a sector of Quito on a map.

The mobile application is developed with the JavaScript programming language, using the Ionic framework for hybrid applications.

The architecture of the system is divided into a Mobile Application, available in virtual stores for free download, both for the IOS operating system and for Android; Web and Database Services (MongoDB) hosted on cloud servers, and at least three prototype UV sensors designed by a local company. The Web Services manage the data and logic layer of all the transactions that are carried out from the Mobile Application. On the other hand, the Database is in charge of providing the information from the samples received by the sensors to the web services. These services are developed in the JavaScript programming language, specifically in the JavaScript runtime built NodeJS. For the consumption of services and transactions with the database, REST requests are used.

Chapter 1: In this chapter, the theoretical foundations of the project will be reviewed to have a clearer vision of the issues involved.

Chapter 2: This chapter will explain the agile development methodology to be implemented and how to divide the project development into three stages: Analysis, Design and Implementation.

Chapter 3: This chapter performed the functional tests, the bug fixes and the implementation of the MOS test to measure the quality of the user experience with this project.

Chapter 4: In this chapter the conclusions and recommendations will be written once the project is finished.

KEY WORDS: UV Radiation, Web Service, REST, Hybrid Applications, JavaScript, Ionic, MongoDB.

1. INTRODUCCIÓN

La radiación ultravioleta puede producir quemaduras en la piel y otros efectos adversos para la salud humana y el medio ambiente. Afortunadamente la capa de ozono filtra la mayor parte de la radiación ultravioleta proveniente del sol, especialmente los rayos ultravioletas B, dejando pasar los rayos ultravioletas A, necesarios para la vida en la tierra. Lamentablemente, el debilitamiento de la capa de ozono generada por productos químicos a base de carbono creados por el hombre permite que los rayos ultravioleta B entren con mayor intensidad a la tierra y afecten la vida en el planeta. [1]

El Instituto Nacional de Meteorología e Hidrología (INAMHI) es un organismo técnico que proporciona información de la radiación ultravioleta, para que los ecuatorianos tomen las medidas de precaución necesarias. No obstante, la insuficiente información proporcionada por dicho organismo a través de su página web (http://www.serviciometeorologico.gob.ec/), provoca un desconocimiento de las medidas reales de protección que los ecuatorianos deben tomar. El sector de estudio elegido para este Trabajo de Titulación se encuentra comprendido entre: el sector Norte Mariscal Sucre, al Sur la Tola, al Este la Vicentina y al Oeste San Blas.

Del análisis inicial se establece que, el problema a resolver es la falta de información del índice de radiación UV en el sector de estudio, ya que el INAMHI no cuenta con una estación meteorológica fija cercana de menos diez kilómetros a la redonda para el sector de estudio seleccionado. Por lo tanto, la información proporcionada por el INAMHI no cubre las demandas de información que los ciudadanos requieren para precautelar su salud y bienestar.

En el caso de que persista el problema, los ciudadanos de Quito en particular del sector, objeto de estudio seguirán desinformados. Los efectos que esto tendrá en la población en general y sobre todo en los grupos vulnerables como niños y ancianos, provocará que no se pueda tomar medidas de mitigación frente a posibles daños en la salud.

Actualmente en el mercado existen aplicaciones móviles como: Sensor UV, Ultravioleta, UV Índex Widget, así como páginas web entre las que destacan: sunburnmap.com, uv.exa.ec, serviciometeorologico.gob.ec (INAMHI), entre otras. Algunas aplicaciones obligan al consumidor a comprar dispositivos UV inteligentes que se conectan a teléfonos móviles para visualizar el índice de radiación UV geo posicionada y otras aplicaciones como Sensor UV el cual unicamente muestra información de un punto geográfico como es el caso para el sector Colegio Isaac Newton. Todos estos antecedentes hacen que los ciudadanos no puedan en un caso adquirir equipos adicionales a su teléfono celular por motivos económicos y en otros

las aplicaciones existentes tienen costo o son limitadas a sectores geográficas específicas, y por lo tanto no satisfacen las necesidades de información para nuestro sector de interés.

Por ello, la propuesta del Trabajo de Titulación es una aplicación móvil gratuita que muestre el índice de radiación UV de tres sectores, para nuestro caso, el alcance será para la zona delimitada ya mencionada conocida como sector de estudio que se encuentra dentro de la ciudad de Quito.

1.1. OBJETIVOS

El objetivo general de este Proyecto es desarrollar una aplicación móvil sobre IOS/Android para representar en un mapa el índice de radiación ultravioleta para un sector de Quito.

Los objetivos específicos son:

- Analizar la radiación ultravioleta y sus efectos sobre los seres humanos, así como los fundamentos teóricos de los servicios web, bases de datos y las aplicaciones móviles híbridas.
- Diseñar una aplicación móvil para representar en un mapa el índice de radiación ultravioleta para el sector de estudio.
- Implementar la aplicación móvil utilizando servicios web.
- Evaluar los resultados obtenidos de los datos mostrados en las pruebas de funcionamiento.

1.2. ALCANCE

El presente Trabajo de Titulación plantea el desarrollo de una aplicación móvil utilizando servicios web, una base de datos y una aplicación sobre Android/IOS.

El área de estudio se encuentra limitada al Norte por la intersección entre las avenidas Colón y 10 de agosto (sector Mariscal Sucre); al Sur por el sector de Nueva Tola Bella (sector La Tola); al Este por el barrio San Vicente (sector La Vicentina) y al Oeste por el sector de la Basílica Católica Nuestra Señora de la Merced (sector San Blas), como se indica en la Figura 1.1.

Para este estudio se ha representado en un mapa donde se visualice la información del índice de radiación ultravioleta acompañado de una representación de colores como un semáforo donde se indica la intensidad del índice de radiación UV, como se muestra en la Figura 1.1.

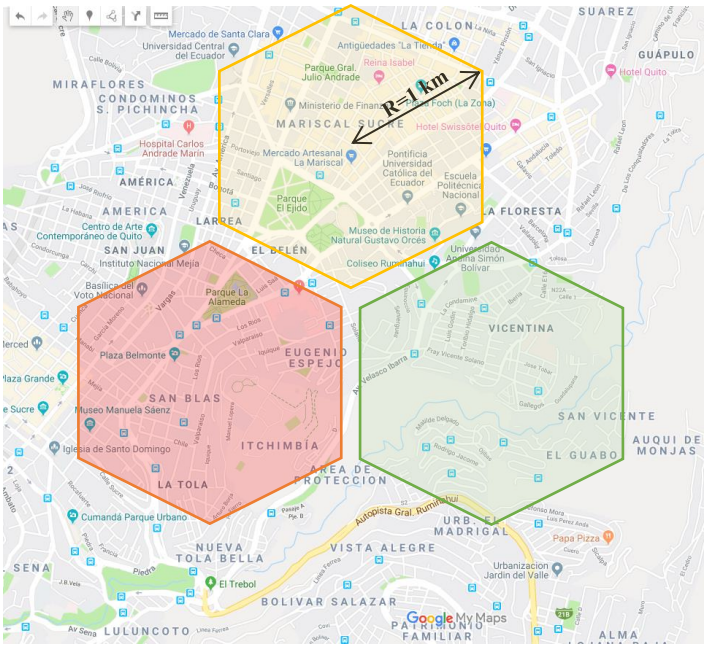


Figura 1.1. Sector de estudio. Fuente: Ian Baquero.

Los valores del índice UV serán representados en el mapa según el estándar mundial de códigos de colores asociados a la medición [2]. La medición del índice de radiación UV será obtenida en tres subáreas, cada una conformada por un sensor UV en puntos geográficos determinados a medida que se desarrolle el proyecto. Cada subárea tendrá una forma hexagonal de al menos un kilómetro de radio aproximadamente (esta medida de longitud será evaluada en las pruebas de funcionamiento), además estas subáreas serán pintadas cada una según el color asociado al índice UV obtenido en ese instante de tiempo dado por cada sensor UV, como se puede ver en la Figura 1.1.

El esquema del proyecto se puede observar en la Figura 1.2, en el cual las mediciones del índice de radiación UV serán obtenidas por tres sensores, los mismos que se fabricarán bajo pedido a una empresa local. Cada sensor contiene un Arduino, un módulo GSM (Global System for Mobile), un sensor UVM30a y un módulo reloj, los cuales se conectan a un servidor web que almacenan los datos de sensores, para la transmisión de los datos obtenidos a través de la tecnología GSM/GPRS (General Packet Radio Service) que utiliza la red telefónica; utilizando el servicio REST con protocolo HTTP (Hypertext Transfer Protocol). Posteriormente, el servidor web de datos de sensores enviará la información empleando el lenguaje PHP utilizando la extensión PDO[1] a la base de datos en la nube. El servidor web de la aplicación móvil utilizará el servicio REST (Representational State Transfer) con protocolo HTTP, el mismo que recibe la información solicitada de la base de datos para mostrarla en la aplicación móvil. Se realizarán al menos tres mediciones de datos con los sensores UV por día (una en la mañana, otra a medio día, y otra en el atardecer), las horas serán definidas a medida que se desarrolle el proyecto.

[1]**PDO (PHP Data Object)**: Es una extensión de PHP que define una interfaz ligera para poder acceder a bases de datos en PHP.

Las muestras almacenadas en la base de datos consistirán en objetos, los mismos que tendrán el campo “hora”, el cual almacenará la fecha y hora de la muestra tomada por al menos tres sensores; permitiendo de esta manera realizar estadísticas básicas para representar en un gráfico dinámico un resumen de la variación del índice UV por día, semana y por mes. Además, la aplicación móvil mostrará recomendaciones de las medidas de seguridad que deberá tomar el usuario para todos los índices de radiación presentes.

La información que se le presentará al usuario no depende del GPS de su teléfono móvil, se mostrará un mapa con la información pertinente y en base al interés del usuario, este podrá ver el índice de radiación UV en cualquiera de las subáreas definidas para el sector de estudio.

En caso de incremento o eliminación de sensores UV el procedimiento que seguirá la aplicación consistirá en un barrido total de las muestras almacenadas para un día determinado y se escogerá la más reciente por cada sensor, y este será el desplegado en el mapa según sus coordenadas geográficas. En el caso que un sensor UV deje de funcionar en un día determinado la información mostrada en el mapa será la última muestra válida.

“Por Calidad de Experiencia entendemos la calidad de un proceso según es percibido por el usuario. Se trata de aquello que hay que maximizar en la prestación de un servicio.”[3]. De esta manera se diseñará una prueba de calidad de experiencia de usuario bajo parámetros de usabilidad de software, para valorar la experiencia del usuario final, se utilizará el instrumento de evaluación Mean Opinión Score (MOS) que generalmente nos permite medir que tan agradable es para el usuario final un sistema o aplicación utilizando encuestas como medios de verificación.

En base a lo descrito anteriormente se definen los roles para la aplicación móvil.

Cliente: Es el usuario que utiliza la aplicación para visualizar el mapa con toda la información referente al índice UV, sus medidas de protección y el gráfico dinámico de al menos tres subáreas. Se prevé realizar pruebas con al menos cinco clientes. Todos los usuarios serán anónimos, por lo tanto, no se necesitará de una interfaz de acceso.

Administrador: Es el usuario que realiza las operaciones CRUD relacionadas a la información presentada en la aplicación móvil como las subáreas en el mapa, el gráfico dinámico y las medidas de protección. Se prevé tener un administrador, el mismo que tendrá acceso al sistema en la nube, a través, de las credenciales proporcionadas por el servicio en la nube contratado.

La aplicación de sistema distribuido se desarrollará basándose en la metodología ágil Kanban. Este sistema distribuido estará conformado por servicios web conectados a una base de datos MongoDB y un *framework* que permita crear aplicaciones móviles híbridas.

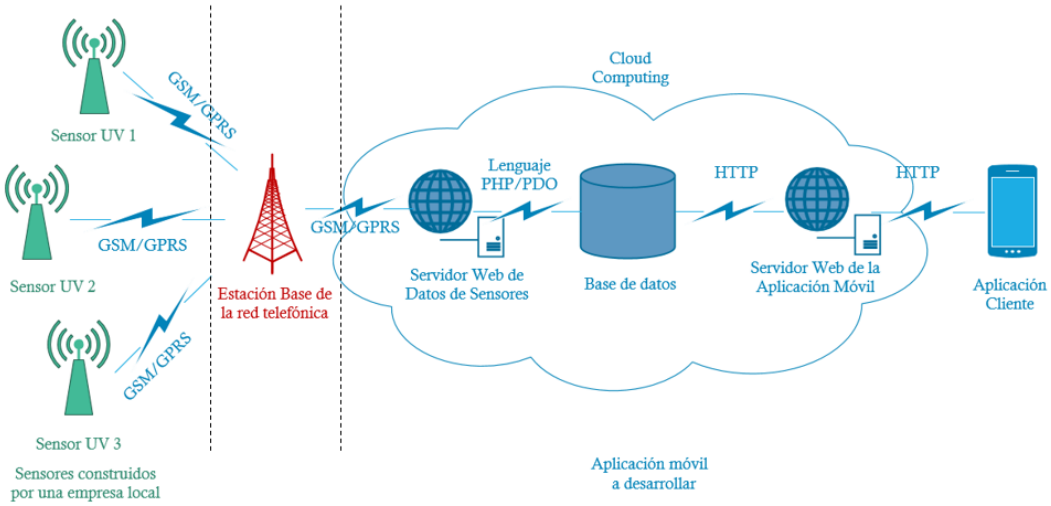


Figura 1.2. Esquema del proyecto. Fuente: Ian Baquero.

El presente Trabajo de Titulación tendrá un producto final demostrable que estará disponible para su descarga gratuita.

1.3. MARCO TEÓRICO

Se explicarán los conceptos de bases de datos, servicio REST, aplicaciones móviles híbridas, así como los fundamentos teóricos relevantes de la radiación UV, sus efectos nocivos en la salud humana y las medidas de mitigación. En el campo de los servicios web se tratarán conceptos entre otros la arquitectura de capas y Application Programming Interface (API).

Además, se detallarán conceptos respecto a la metodología KANBAN que serán usadas para el desarrollo de la aplicación móvil.

1.3.1. RADIACIÓN UV

La luz solar es la fuente más habitual de energía radiante electromagnética, conformada por el espectro infrarrojo, visible y ultravioleta. La radiación UV se encuentra comprendido entre los 100 y 400 nanómetros, la misma que se divide en tres rangos: UVA, UVB y UVC.

- **UVA**: Compone al menos el 90% de la radiación que llega a la superficie terrestre, esta se encuentra en el rango de los 315 y 400 nm.

- **UVB**: Compone al menos el 10% de las radiaciones que llegan a la superficie terrestre, se encuentra en el rango de los 280 y 315 nm.
- **UVC**: Es absorbida en su totalidad por la capa de ozono, esta se encuentra en el rango de los 100 y 280 nm. [4]

1.3.1.1. Datos y riesgos de los rayos UVA

Los rayos UVA, son menos intensos que los UVB, ya que penetran la piel más profundamente. La larga exposición causa daño genético en las células en la parte más interna de la capa superior de la piel (dermis), donde ocurre la mayoría de los canceres de piel. La piel intenta evitar más daños al oscurecerse, lo que conocemos como un bronceado. Con el tiempo, los rayos UVA también conducen al envejecimiento prematuro y al cáncer de piel.

1.3.1.2. Datos y riesgos de los rayos UVB

Los rayos UVB, son los responsables de los efectos biológicos más importantes de dichas radiaciones sobre el ser humano, tienen un poco más de energía que los rayos UVA. Sobre la piel, tiene efectos nocivos a corto y mediano plazo. Desde enrojecimiento de la piel, quemaduras leves hasta casos graves como ampollas. Estos rayos pueden dañar directamente el ADN de las células de la piel. [5]

El riesgo ante la radiación UV disminuye a medida que aumenta el grado de pigmentación natural de la piel del ser humano, siendo máximos en pieles blancas y mínimo en personas de piel negra.

Respecto a los efectos sobre la vista, las afecciones más frecuentes, son las apariciones de cataratas que en casos extremos puede ser capaz de producir ceguera. Además, uno de los canceres oculares más frecuentes, el melanoma de úvea, la cual está en aumento. [6]

La potencia de los rayos UV que llega al suelo depende de un numero de factores, tales como:

- **Hora del día:** Los rayos UV son más potentes entre 10 a.m. y 4 p.m.
- **Temporada del año:** Los rayos UV son más potentes durante los meses de la primavera y el verano. Este es un factor menos importante cerca del ecuador.
- **Distancia desde el ecuador (latitud):** La exposición a UV disminuye a medida que se aleja de la línea ecuatorial.
- **Altitud:** Más rayos UV llegan al suelo en elevaciones más altas.

- **Formación nubosa:** El efecto de las nubes puede variar, ya que a veces la formación nubosa bloquea a algunos rayos UV del sol y reduce la exposición a rayos UV, mientras que algunos tipos de nubes pueden reflejar los rayos UV y pueden aumentar la exposición a los rayos UV. Lo que es importante saber es que los rayos UV pueden atravésar las nubes, incluso en un día nublado.
- **Reflejo de las superficies:** Los rayos UV pueden rebotar en superficies como el agua, la arena, la nieve, el pavimento, o la hierba, lo que lleva a un aumento en la exposición a los rayos UV.[7]

1.3.1.3. Medidas de protección frente a la radiación UV

- Evitar la exposición solar en las horas centrales del día. Los rayos UV solares son más fuertes entre las 10 de la mañana y las 4 de la tarde. Tener especial cuidado con la exposición al sol durante esas horas.
- Tener en cuenta el índice UV. Este importante dato le ayuda a planificar las actividades al aire libre para evitar una exposición excesiva a los rayos del sol. Es necesaria protección solar siempre que el índice UV prevea niveles de exposición de moderados a altos, por ejemplo, un índice UV de 3 o superior.
- Aprovechar las sombras. Póngase a la sombra cuando los rayos UV sean más intensos, pero no olvide que los árboles, las sombrillas o los toldos no protegen totalmente contra la radiación solar.
- Usar ropa que proteja. Un sombrero de ala ancha protege debidamente los ojos, las orejas, la cara y la parte posterior del cuello. Las gafas de sol con un índice de protección del 99%-100% frente a los rayos UVA y UVB reducen considerablemente los daños oculares debidos a la radiación solar. Las prendas de vestir holgadas y de tejido tupido cubren la mayor superficie corporal posible, las mismas que protegen contra el sol.
- Utilizar cremas con filtro solar. Aplicar una crema protectora de amplio espectro, con factor de protección igual o superior a 30. Extender generosamente sobre la piel expuesta y repita la aplicación cada dos horas, o después de trabajar, nadar, jugar o hacer ejercicio al aire libre.
- Evitar las lámparas y las camas bronceadoras. Las lámparas y las camas bronceadoras aumentan el riesgo de cáncer de la piel y pueden dañar los ojos si no se usa protección. Debe evitarse completamente su uso.

- Proteger a los niños. Los niños suelen ser más vulnerables a los riesgos ambientales que los adultos. Cuando estén al aire libre, hay que protegerlos de la exposición a los rayos UV como ya se ha explicado. Los bebés deben permanecer siempre en la sombra. [8]

1.3.2. BASES DE DATOS

Se conoce a una base de datos como un conjunto de datos o información agrupada perteneciente a un mismo contexto, almacenadas en un espacio de memoria para su posterior recuperación o modificación.

Para el manejo de las bases de datos, es decir su almacenamiento ordenado y rápida recuperación de la información, se necesita de un Sistema de Administración de Base de Datos (SABD), que son un tipo de software dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que utilizan. Se compone entre otras cosas, de un lenguaje de consulta, el cual permite extraer información de la base como respuesta a consultas. NoSQL es una amplia clase de sistemas de gestión de datos que difieren, en aspectos importantes, del modelo clásico de relaciones entre entidades (o tablas) existente en los sistemas de gestión de bases de datos relacionales. Un tipo de base NoSQL muy utilizada actualmente es MongoDB del tipo orientada a documentos. [9]

1.3.2.1. Bases de Datos NoSQL Documentales

El concepto principal en estas bases de datos es el de “documento”. Un documento es la unidad principal de almacenamiento de este tipo de base de datos, y toda la información que aquí se almacene, se hace en formato de documento, generalmente utilizando una estructura simple como JSON o XML, en el cual se utiliza una clave única para cada registro, y que también son utilizados para ser recuperados posteriormente. [10]

1.3.2.2. MongoDB

Es una base de datos NoSQL Open Source orientada a documentos. Creada por la empresa 10gen con la finalidad de que fuera fácil tanto de desarrollar como de ser administrada. El denominado documento está estructurado de datos compuesta de pares de campos y valores. Estos objetos son muy similares a los objetos en notación JSON. Una propiedad de este tipo de base de datos es que los valores de los campos del documento pueden ser otros documentos, *arrays* o incluso *arrays* de documentos. [11]

1.3.3. ARQUITECTURA EN CAPAS

La programación por capas es un estilo de programación que consiste en la separación de la lógica de negociación de la lógica de diseño. La arquitectura de tres capas es diferente de la tradicional arquitectura cliente/servidor de forma que la aplicación en vez de residir en el equipo del cliente, reside en la capa-media, es decir en el Servidor de Aplicación. Esta arquitectura de tres capas se divide en una capa de presentación, capa de negocio y capa de datos.

1.3.3.1. Capa de Presentación

Conocida como capa de interfaz de usuario, es la capa que el sistema presenta su diseño de interfaz al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso.

1.3.3.2. Capa de Negocio

Esta capa se encarga de la gestión de peticiones realizadas por el usuario y el envío de respuestas tras el proceso. Esta capa es la responsable de cumplir con todas las reglas de la lógica del sistema, validar entradas y salidas, llamar servicios externos, entre otras responsabilidades de la capa de negocio.

1.3.3.3. Capa de Datos

Esta capa consiste en acceder a los datos de la base de datos, conformada por uno o más gestores de bases de datos. Además, esta capa interactúa con la capa de negocio. [12]

Para el presente Trabajo de Titulación la capa de negocio y capa de datos físicamente se encuentran en servidores en la nube referidos como “Servidor”. Estos servicios son accesibles por la capa de presentación, referida como “Aplicación Móvil”, la cual se encarga de interactuar con los requerimientos del usuario para enviarlos al servidor y que responda con los resultados de las operaciones realizadas.

1.3.4. SERVICIOS EN LA NUBE

Los proveedores de servicios en la nube permiten reducir las inversiones por hardware y software de los usuarios, sustituyéndolas por gastos en servicios en un atractivo esquema de pago por uso, a la vez ofrecen otras ventajas tales como acceso a los servicios contratados desde cualquier lugar, flexibilidad, escalabilidad, *backup*, etc. [13]

Dentro de los servicios en la nube se puede identificar 3 modelos de servicio principales:

1.3.4.1. Infraestructura como servicio (IaaS)

Un proveedor proporciona a los clientes acceso directo a recursos informáticos, tales como: servidores, almacenamiento y redes. En vez de comprar directamente el hardware, los usuarios pagan por el IaaS según la demanda. La infraestructura es escalable según las necesidades de procesamiento y de almacenamiento.

Dentro de los proveedores IaaS que ofrecen este servicio se encuentran Microsoft Azure y Amazon Web Services (AWS).

1.3.4.2. Plataforma como servicio (PaaS)

Consiste de un proveedor que ofrece acceso a un entorno basado en la nube, en el cual los clientes pueden desarrollar, gestionar y ofrecer aplicaciones. Además del almacenamiento, se puede utilizar conjuntos de servicios desarrollados por el proveedor para realizar pruebas y crear prototipos.

1.3.4.3. Software como servicio (SaaS)

Es una oferta de computación en la nube que consiste en ofrecer acceso a un software alojado en la nube de un proveedor. Esto quiere decir, que los usuarios del software no necesitaran instalar las aplicaciones en sus dispositivos locales, en vez de eso, las aplicaciones residen en una red de nube remota a la que se accede por medio de la web o de una API. [14]

Para el presente Trabajo de Titulación se utilizó los servicios de infraestructura como servicio del proveedor Amazon Web Services, para alojar los servicios web que utiliza la Aplicación Móvil.

1.3.5. SERVICIOS WEB

Es el método de comunicación entre dos dispositivos electrónicos dentro de una red. Contiene un conjunto de protocolos abiertos y estándares utilizados para el intercambio de datos. Debido a que las aplicaciones están escritas en varios lenguajes de programación que funcionan en plataformas diferentes, estas pueden utilizar servicios web para intercambiar información sobre una red.

La interacción se basa en el envió de solicitudes y respuestas, entre un cliente y un servidor. Por lo tanto, podemos decir que un servicio web es como un tráfico de mensajes entre dos dispositivos sobre la red.

El primer servicio web en el mundo del internet fue SOAP, el cual es un protocolo que define como debe realizarse las comunicaciones entre dispositivos informáticos. SOAP utiliza XML como lenguaje de intercambio de datos. [15]

1.3.5.1. REST (REpresentational State Transfer)

Es un tipo de arquitectura de desarrollo web que utiliza las URLs (Localizador Uniforme de Recursos) a través del protocolo HTTP para realizar consultas por la red. Este tipo de servicio web consiste en un conjunto de reglas que se debe cumplir para el diseño de una API (Application Programming Interface), entendido de mejor manera como un estilo de arquitectura software. Este conjunto de reglas que debe seguir son las siguientes: [16]

- Ser un sistema cliente-servidor.
- Ser una interfaz uniforme.
- Soportar un sistema de caches.
- Ser sin estado.
- Ser un sistema por capas.
- Utilizar mensajes auto-descriptivos.

Una API consiste en un conjunto de servicios web que permiten comunicarse con la capa de negocio. No obstante, si esta API es diseñada por este conjunto de reglas del tipo REST, se la conoce como una API REST. Hoy en día es muy utilizada por la mayoría de las empresas para crear servicios, debido a que es un estándar eficiente y lógico para la creación de servicios web.

Una de las características más importantes de las API REST son que las respuestas a las peticiones realizadas por el cliente son en formato XML o JSON, los formatos de intercambio de información más utilizados en el internet. [17]

1.3.5.2. JSON (JavaScript Object Notation)

REST utiliza un formato de representación de datos conocido como JSON. Este formato consiste de ficheros de texto plano que son utilizados por el lenguaje de programación JavaScript. Un objeto JSON se basa en dos estructuras: una colección de pares “nombre”: “valor” y una lista desordenada de valores como una matriz, vector o lista de objetos.

Un objeto JSON es un conjunto desordenado de pares nombre/valor, un objeto está dentro de llaves {y}; los diferentes campos del objeto se organizan en pares “nombre”: valor y están separados por comas; finalmente, las secuencias de elementos están dentro de corchetes [y]. Tal como se puede observar en la Figura 1.3 [16]

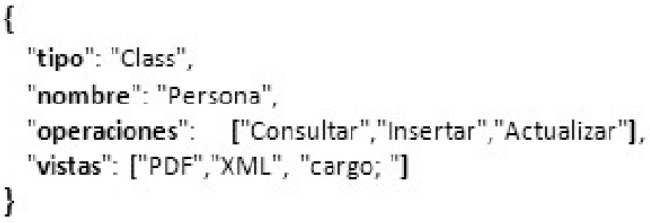


Figura 1.3. Ejemplo de notación de un objeto JSON [18]

1.3.6. API (APPLICATION PROGRAMMING INTERFACE)

Después que un usuario instale una aplicación móvil en su dispositivo y la utilice, el usuario interactúa con el sistema operativo del móvil por medio de las llamadas API propietarias del sistema operativo. Estas se dividen en dos grupos: APIs de bajo nivel y APIs de alto nivel.

1.3.6.1. APIs de bajo nivel

Permite que las aplicaciones puedan acceder directamente al hardware del móvil, para interactuar con dispositivos propios del dispositivo tales como pantalla táctil, teclado, GPS, micrófono, altavoz, acelerómetro, etc.

1.3.6.2. APIs de alto nivel

Incluye los servicios de bajo nivel mencionados, además de otros servicios como navegador web, contactos, acceder a llamadas telefónicas, enviar y recibir mensajes de texto. A pesar de que la mayoría de los sistemas operativos móviles incluyen aplicaciones incorporadas que permitan acceder a este tipo de servicios, existe APIs de alto nivel asequibles a través de aplicaciones móviles descargables que permiten acceder a la mayoría de los servicios de alto nivel mencionados.

1.3.7. APLICACIONES MÓVILES HÍBRIDAS

Una aplicación móvil híbrida es similar a cualquier aplicación, con la particularidad de combinar el enfoque de desarrollo nativo con la tecnología Web como HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) y JavaScript. Este enfoque utiliza la tecnología Web para hacer de una aplicación en multiplataforma, y también utiliza el acceso directo a APIs nativas cuando lo necesita.

La parte nativa de las aplicaciones híbridas utiliza APIs de sistemas operativos para crear un motor de búsqueda HTML incorporado que funcione como puente entre el navegador y las APIs del dispositivo, con el objetivo de procesar partes de la interfaz de usuario de la

aplicación móvil que están escritas en formatos estándar como HTML, CSS y JavaScript, y ejecutarlas localmente dentro del contenedor nativo (Web View). Las funcionalidades nativas son accesibles mediante el uso de un puente JavaScript abstracto como se puede ver en la Figura 1.4. [19]

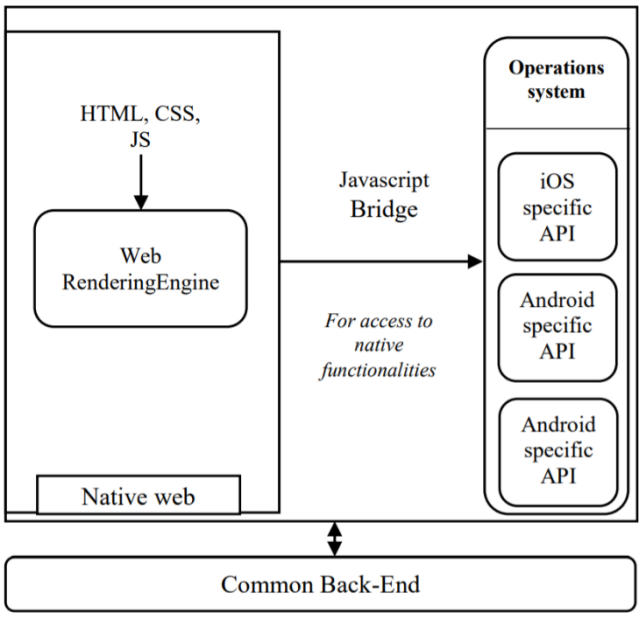


Figura 1.4. Enfoque Híbrido [20]

El *framework* de Ionic es una solución híbrida para construir aplicaciones móviles similares a las aplicaciones nativas, basadas en HTML5, la cual, utiliza *plugins* de Cordova para obtener acceso a características móviles nativas como notificaciones, almacenamiento de archivos, etc.

El enfoque de interpretación de las aplicaciones híbridas consiste en utilizar un lenguaje común como JavaScript para escribir el código de la interfaz de usuario, para así, generar un equivalente a un componente nativo que puede ser compilado en diferentes plataformas.

El enfoque de compilación cruzada es fundamental, ya que, mediante el código del usuario desarrollado en un lenguaje común, se permite transformar este código por compilación cruzada en un código especifico nativo para su plataforma respectiva, como se puede ver en la Figura 1.5 [20]

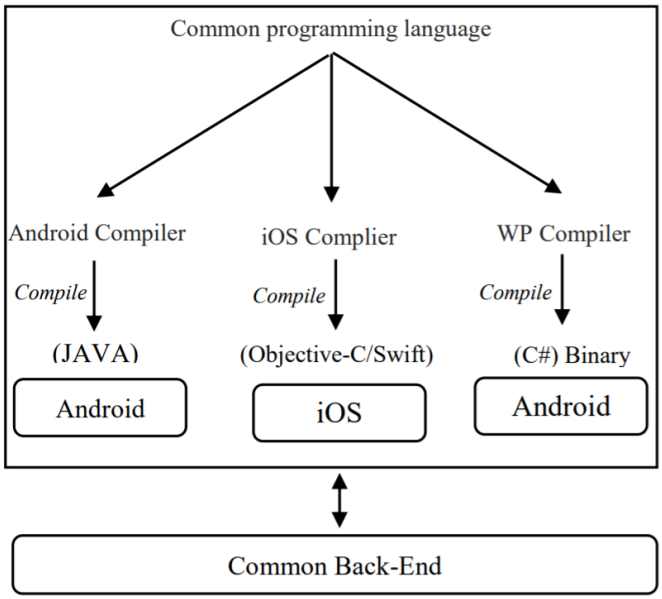


Figura 1.5. Enfoque de compilación cruzada [20]

La principal ventaja de este enfoque es permitir tener aplicaciones que pueden lograr un rendimiento casi nativo. La desventaja es que algunas características no pueden ser usadas como por ejemplo el servicio de geolocalización y de cámara, ya que estas funciones son específicas para cada plataforma y su acceso difiere de una plataforma a otra. [20]

1.3.7.1. Apache Cordova

Es un marco de desarrollo móvil de código abierto. Permite utilizar tecnologías web estándar: HTML5, CSS3 y JavaScript para el desarrollo multiplataforma. Las aplicaciones se ejecutan dentro de los envoltorios (*wrappers*) específicos para cada plataforma y se aprovechan de enlaces con APIs compatibles para acceder a las capacidades y recursos de cada dispositivo móvil.

Grandes empresas avalan esta tecnología ayudando a su desarrollo con productos orientados a este *framework*.

En la Figura 1.6, se puede observar cómo está conformada la arquitectura de una plataforma híbrida en un dispositivo de ejemplo, y como gracias a las APIs de Cordova hacen de puente de comunicación entre la parte del Web App donde se encuentra desarrollada la aplicación híbrida y la parte nativa del dispositivo móvil para hacer uso de los recursos del dispositivo donde se lance la aplicación híbrida.

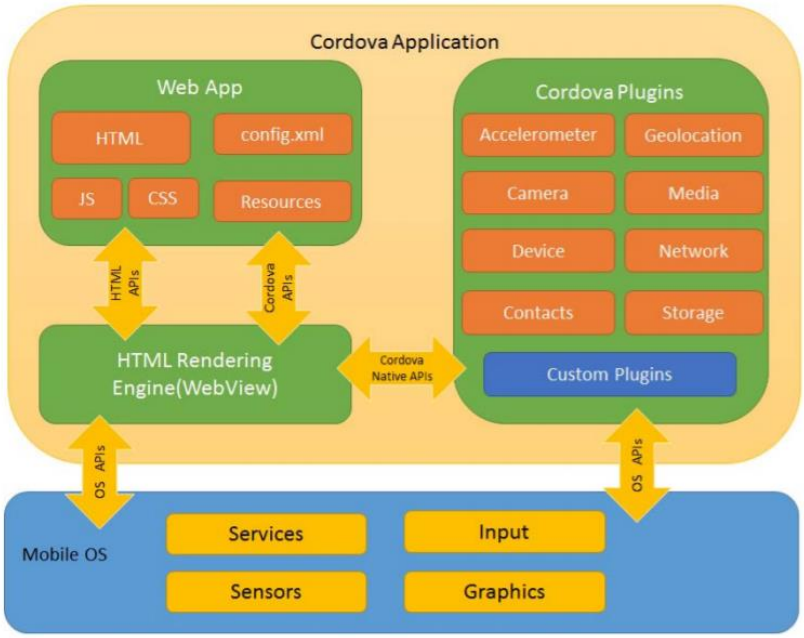


Figura 1.6. Arquitectura de aplicación Híbrida (Apache Cordova) [21]

1.3.8. FRAMEWORK IONIC

Ionic es un *framework* de código abierto para crear aplicaciones móviles y de escritorio de alta calidad construido sobre Angular y Apache Cordova, además, Ionic proporciona herramientas y servicios para desarrollar aplicaciones móviles nativas y de tipo híbrido, esto quiere decir que su codificación es utilizada en aplicaciones iOS como para Android. [22]

Ionic se basa en Angular, el cual fue desarrollado por Google. Angular es un marco estructural para aplicaciones web dinámicas de código abierto de Modelo Vista-Controlador similar al marco estructural de JavaScript. Angular se utiliza para administrar la lógica y los datos de la aplicación web dentro de nuestra aplicación Ionic. Las aplicaciones web de Angular se ejecutan dentro de un contenedor conocido como WebView, además, es el que realiza la lógica de validación entre pantallas y manipulación del HTML para expresar los componentes de su aplicación de manera clara. [22]

Ionic está construido encima de Angular como se muestra en la Figura 1.7. para proporcionar una interfaz interactiva al usuario, este usuario se encargará de diseñar su interfaz con elementos visuales como botones, cabeceras de navegación, pestañas y otros componentes más. Estos componentes son el corazón de Ionic y permite a la interfaz de usuario tener un acercamiento nativo, a pesar de ser una aplicación híbrida. [22]

1.3.9. ARQUITECTURA DEL FRAMEWORK IONIC

Al abrir una aplicación móvil en los dispositivos, el contenedor de aplicación Cordova (*wrapper*) es cargado basándose en las características del dispositivo. El *wrapper* contiene una WebView llamada index.html que gracias a la ayuda de la API CordovaJS, llama al motor de Front-End de nuestra app que es Angular permitiendo desplegar las diferentes pantallas del usuario. Además, gracias a los componentes de Ionic los cuales son parte de la aplicación web del Front-End, estos ayudaran a tener una interfaz de usuario más concisa y amigable. [22]

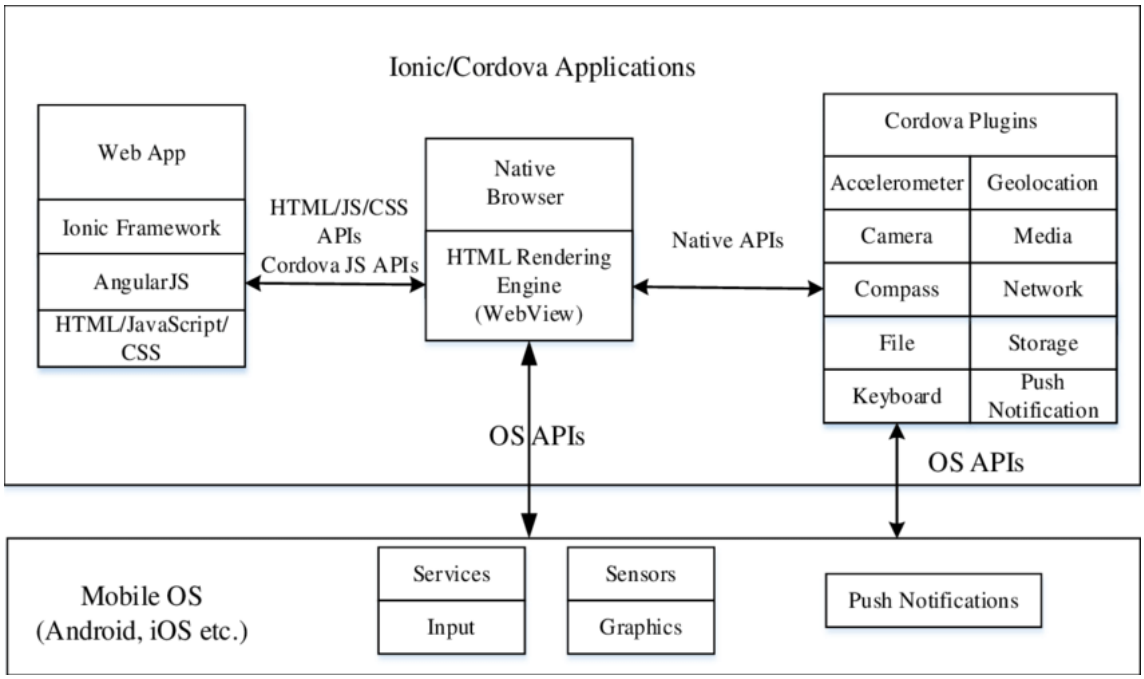


Figura 1.7. Arquitectura de Ionic [23]

El diagrama de la Figura 1.7 permite tener una mejora apreciación de cómo están conformados los componentes técnicos de la arquitectura Ionic y como estos se comunican. Dado que Cordova JavaScript API es un componente importante que permite correr HTML, CSS y JavaScript como substituto de las APIs específicas de la plataforma, se describe a Cordova como un puente de comunicación entre la aplicación web y el dispositivo, el *wrapper* tiene acceso a ambos, tanto a la aplicación web como a la plataforma nativa mediante el API de JavaScript. [22]

1.3.10. METODOLOGÍA DE DESARROLLO KANBAN

Kanban es una metodología ágil para administrar la creación de productos, enfocada en la entrega continua, sin sobrecargar al equipo de desarrollo. La palabra Kanban proviene del japonés “kan” que significa visual y “ban” tarjeta, el cual unidos significan “tarjetas visuales”. Kanban se basa en los cuatro siguientes principios básicos: [24]

- Visualización del trabajo: Kanban se basa en el incremento de tareas, dividiendo todo el desarrollo en partes, con la intensión de separar el producto en tareas para así, distribuir la carga de trabajo. Una de sus principales características es que utiliza técnicas visuales para ver el estado de las tareas, las mismas que se representan en un tablero llamado tablero Kanban.
  El objetivo de este tablero es visualizar el trabajo a realizar para cada miembro del equipo de desarrollo y tener clara la prioridad de las tareas.
- Limitar el trabajo en procesos: Se define la cantidad máxima de tareas a realizar por ciclo de trabajo, a este número de tareas se las conoce como límite del trabajo en progreso. El objetivo es enfocarse en cerrar tareas y no en empezar nuevas.
- Administrar el flujo: Se mide el tiempo en que termina cada tarea. Cuando una tarea se termina, la tarea con mayor peso de importancia se pone a continuación, de esta manera se tiene un control y mejora del flujo de trabajo.
- Mejoramiento continuo: Kanban fomenta cambios continuos, incrementales y evolutivos para mejorar, y hasta incluso acelerar el procedimiento de las tareas. Kaizen es una palabra clave que significa mejora continua. Kaizen es fundamental para el uso efectivo de Kanban, para implementarlo se sugiere construir una comprensión compartida del problema y sugerir acciones que se puedan acordar por consenso. [25]

Kanban fomenta el trabajo en equipo para la colaboración y aprendizaje continuo, con el objetivo de mejorar el flujo de trabajo del equipo. [24]

1.3.10.1. Tarjetas Kanban

Una tarjeta Kanban representa una tarea o actividad que se debe realizar ubicado dentro del tablero Kanban, el mismo que contiene información concisa sobre la tarea a realizar, miembro responsable de la tarea y el tiempo estimado de culminación de la tarea. Son utilizadas como indicadores de progreso para todos los miembros del grupo. [26]

1.3.10.2. Tablero Kanban

El tablero Kanban es una herramienta que se utiliza para visualizar el flujo de trabajo y las tarjetas Kanban. Usualmente el tablero se encuentra dividido por columnas y filas. Cada columna simboliza una fase o estado de su proceso y las filas representan los diferentes tipos de actividades específicas como diseño, pruebas, análisis, implementación, etc.

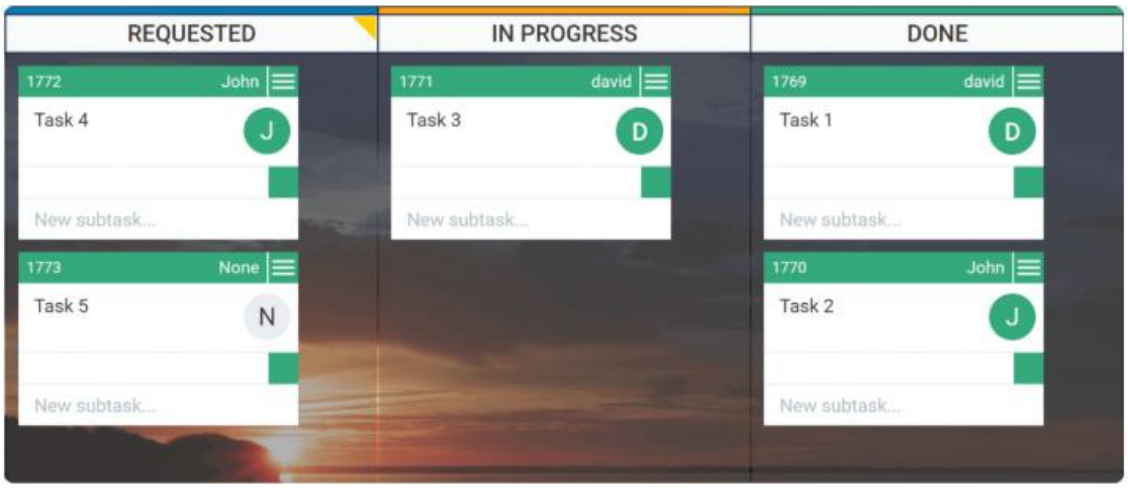


Figura 1.8. Tablero Kanban [27]

Las columnas del tablero Kanban se dividen en tres secciones que representan el estado básico de sus tareas:

- Por hacer.
- En proceso.
- Hecho.

Como se observa en la Figura 1.8 el tablero tiene los tres estados básicos. Dos tarjetas en el estado por hacer, una tarjeta en el estado en proceso y dos tarjetas en el estado hecho. Cada tarjeta representa una tarea por realizar con su encargado respectivo y su tiempo estimado. [27]

1.3.11. HERRAMIENTAS DE DESARROLLO

1.3.11.1. Visual Studio Code

Es un editor de texto desarrollado por Microsoft compatible con MacOS, Linux y Windows. Visual Studio Code se caracteriza de un entorno de desarrollo simplificado, que permite editar código sin compilaciones continuas para su funcionamiento.

Visual Studio Code permite crear aplicaciones eficaces y de alto rendimiento, permitiendo desarrollar servicios web, sitios y aplicación web, entre otros en cualquier entorno que soporte la plataforma. Es importante saber que esta herramienta combina tecnologías web con la velocidad y flexibilidad de las aplicaciones nativas; además, incluye un modelo de extensibilidad publica para los desarrolladores con el objetivo de crear y usar extensiones para mejorar la experiencia del usuario. [28]

1.3.11.2. Robomongo

Es una herramienta multiplataforma que consiste en un administrador gráfico para nuestras bases de datos, el mismo que se ha convertido en un aliado importante para los usuarios de MongoDB.

Robomongo toma el Shell e integra en un entorno grafico toda su funcionalidad, además de añadir nuevas funcionalidades como:

- Múltiples conexiones a las bases de datos, separadas por pestañas.
- Resaltado de sintaxis y autocompletado del código.
- Distintos modos de visualización como en modo árbol, tabla y texto plano.

A continuación, se muestra la interfaz de Robomongo en la Figura 1.9, donde una de sus características es la de identificar visualmente los objetos de la base de datos y sus variables con sus respectivos valores. [29]

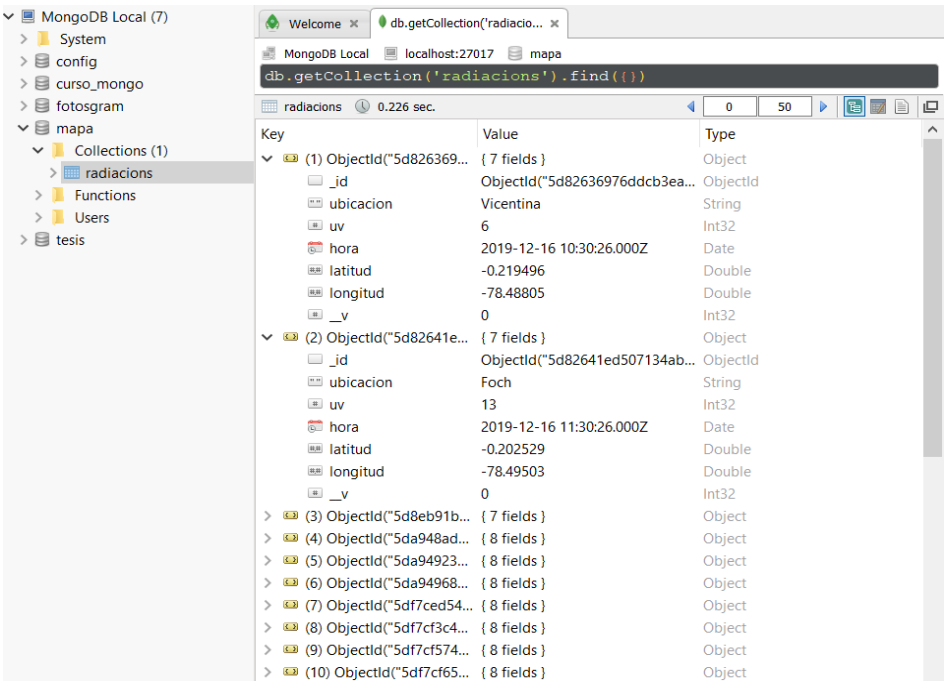


Figura 1.9. Interfaz de Robomongo. Fuente: Ian Baquero.

1.3.11.3. Trello

Es una aplicación basada en la metodología Kanban para gestionar tareas, utilizada para organizar el trabajo en grupo de forma colaborativa mediante tableros virtuales compuestos de listas de tareas en forma de columnas. Esta herramienta es muy útil para desarrollar proyectos con metodología Kanban ya que su tablero virtual tiene la facilidad de representar distintos estados y compartirlas con diferentes personas para que formen un equipo de

trabajo. El objetivo de esta herramienta es mejorar el flujo de trabajo generando tiempos, prioridades, alertas y otros avisos perfectos para organizar un proyecto. [30]

A continuación, en la Figura 1.10 se muestra la interfaz de la aplicación Trello con un ejemplo de tablero virtual con sus respectivas tareas y sus descripciones.

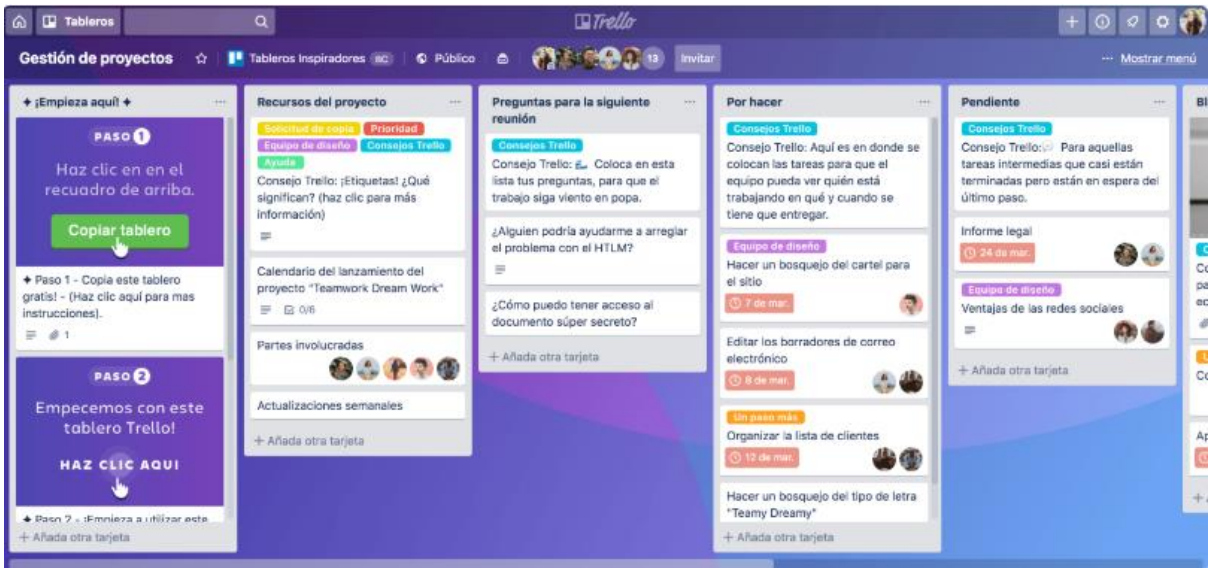


Figura 1.10. Tablero virtual de Trello [31]

1.3.11.4. Postman

Postman surgió como una extensión del navegador Google Chrome. Es una herramienta que se utiliza para el *testing* de API REST, esto quiere decir que permite realizar peticiones a APIs internas o de terceros, con el objetivo de validar el comportamiento de APIs. El interés de esta herramienta consiste en hacer peticiones a APIs y generar colecciones de peticiones que permitan probar de una manera rápida y sencilla.

Dentro del tipo de peticiones que se puede realizar tenemos (GET, POST, etc.), también se puede definir *tokens* de autenticación, cabeceras asociadas a la petición, etc. [32]

A continuación, en la Figura 1.11 se muestra la interfaz de Postman con un ejemplo de una petición POST a una API externa con notación JSON.

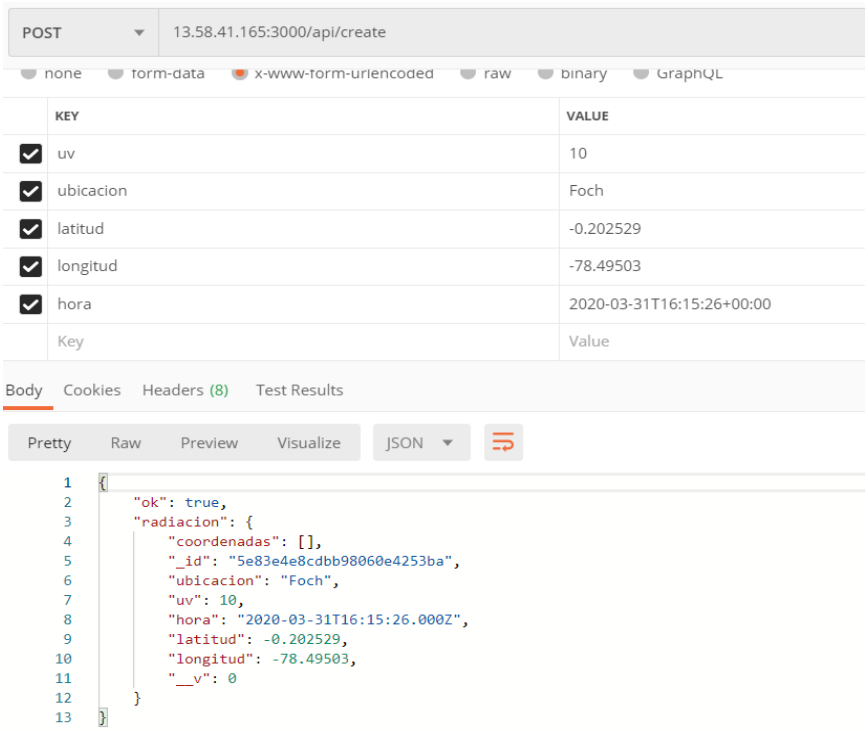


Figura 1.11 Interfaz de la herramienta Postman. Fuente: Ian Baquero.

2. METODOLOGÍA

El presente Trabajo de Titulación propone desarrollar una aplicación móvil, la cual, se basará en la metodología ágil Kanban para gestionar todo el desarrollo de esta. Kanban plantea la división del trabajo en bloques, los cuales tendrán el nombre de ítems y se representarán cada uno en una tarjeta Kanban; estas tarjetas Kanban podrán ser visualizadas en el tablero Kanban, con el objetivo de representar todo el flujo de trabajo.

Se procederá a implementar la metodología ágil Kanban definiendo tres etapas: análisis, diseño e implementación. Se especificará cada etapa dentro de este capítulo, además, cada etapa contará con un conjunto de ítems, los mismos que serán representados en el tablero Kanban. El tablero estará dividido por tres columnas que representaran sus estados: por hacer, en progreso y listo. Por esta razón, al inicio de cada etapa se visualizará todas las tarjetas Kanban en el estado “POR HACER” y una vez finalizada la misma, lo que quiere decir el cumplimiento de todas sus tareas, se ubicarán en el estado “LISTO”.

En la etapa de análisis se comenzará llenando los cuestionarios para al menos seis informantes calificados, las mismas que se encuentran en el ANEXO I. El objetivo de adquirir información de los requerimientos del sistema, y a partir de esta generar historias de usuario, las cuales serán utilizadas para especificar los requerimientos funcionales y no funcionales del sistema.

A partir de los requerimientos se procederá a realizar el diseño de la base de datos. Posteriormente, se diseñará el servicio web utilizando una API REST. Se diseñará la aplicación móvil, se realizarán pruebas de funcionamiento del sistema, donde se detallará el procedimiento a seguir del sistema para enviar la información solicitada por parte de los prototipos UV a los servicios en la nube, para posteriormente ser visualizada en los diferentes dispositivos móviles. Los pasos para comprobar el funcionamiento utilizado se especificarán en el capítulo tres llamado “Resultados y Discusión”, con la conformación de las siguientes etapas: Adquisición de datos, Envío de datos y Presentación de datos. Finalmente, se publicará la aplicación móvil en las tiendas virtuales Play Store y App Store una vez realizada las correcciones necesarias.

2.1. ANÁLISIS

Esta etapa consiste en adquirir información de fuentes externas que ve nuestro producto sin prejuicios, con el fin de identificar problemas percibidos, proponer soluciones y estimar el esfuerzo que requieren implementar las ideas propuestas. Para ello se definen tres ítems en el tablero Kanban. Inicia con el levantamiento de la información para desarrollar las historias usuario e identificar los requerimientos funcionales y no funcionales de la aplicación móvil. Después se procederá a diseñar la aplicación con los diferentes requerimientos definidos.

En la Figura 2.1 se muestra el tablero Kanban de la etapa de análisis con tres ítems en la columna que tiene el estado por hacer.

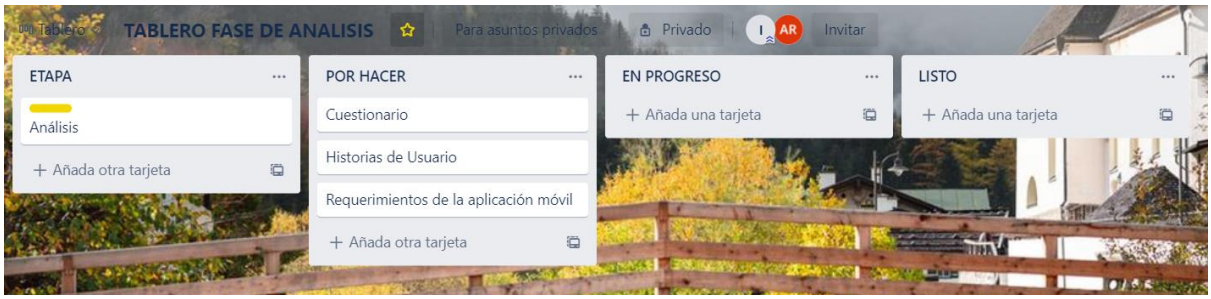


Figura 2.1. Tablero Kanban etapa de análisis estado POR HACER. Fuente: Ian Baquero.

2.1.1. CUESTIONARIO

El objetivo de los cuestionarios es conocer las necesidades de los usuarios frente a las características de las muestras UV tomadas y mostradas en la aplicación móvil; y las preferencias para mejorar la interacción usuario-dispositivo con una aplicación móvil de esta índole.

Se diseñó los cuestionarios con nueve ítems para evaluar al menos a seis informantes calificados. El modelo del cuestionario se encuentra en el ANEXO I. Los cuestionarios se desarrollaron en Google Forms.

El cuestionario se realizó a personas calificadas que cumplieron con el siguiente perfil: Estudiantes egresados o graduados de tercer nivel en las carreras de ingeniería electrónica en redes de la información. Con personas calificadas se referirá a individuos con conocimiento básico en tecnologías de desarrollo de aplicaciones y programación.

A continuación, se mostrará las nueve preguntas y sus resultados respectivos:

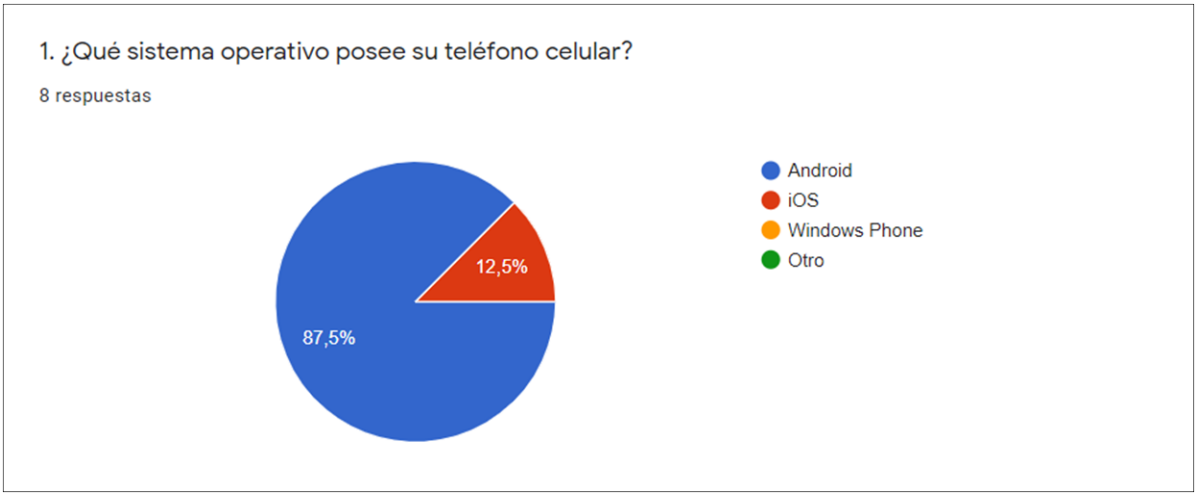


Figura 2.2. Resultados pregunta 1.

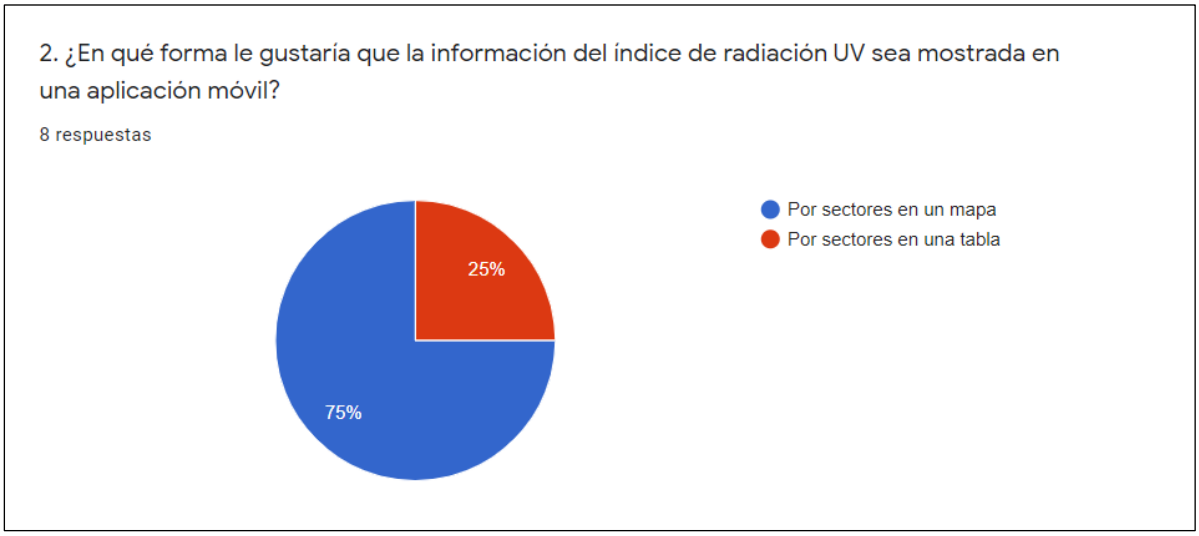


Figura 2.3. Resultados pregunta 2.

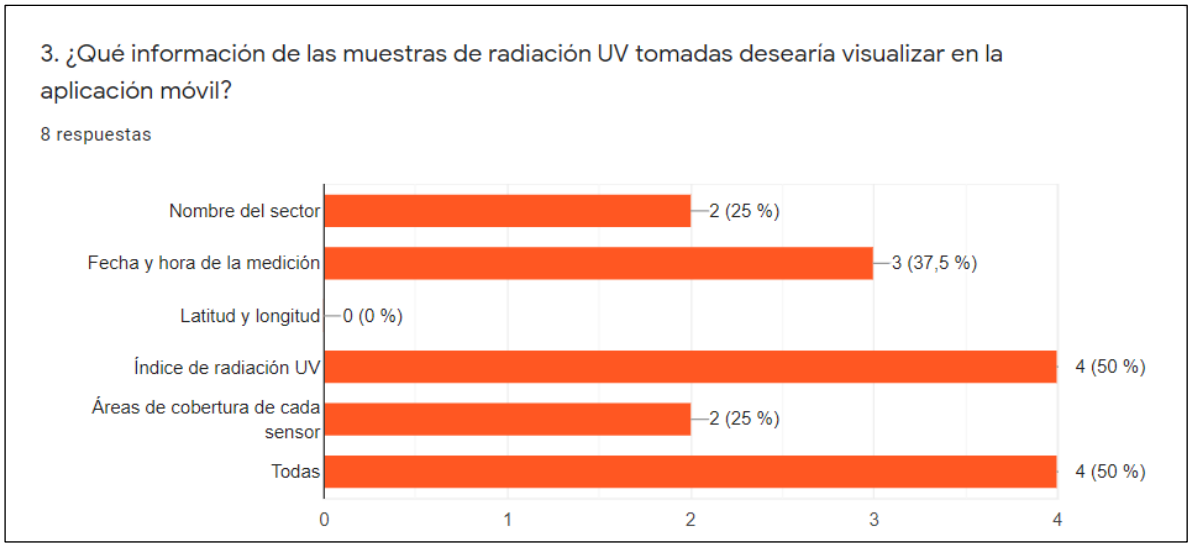


Figura 2.4. Resultados pregunta 3.

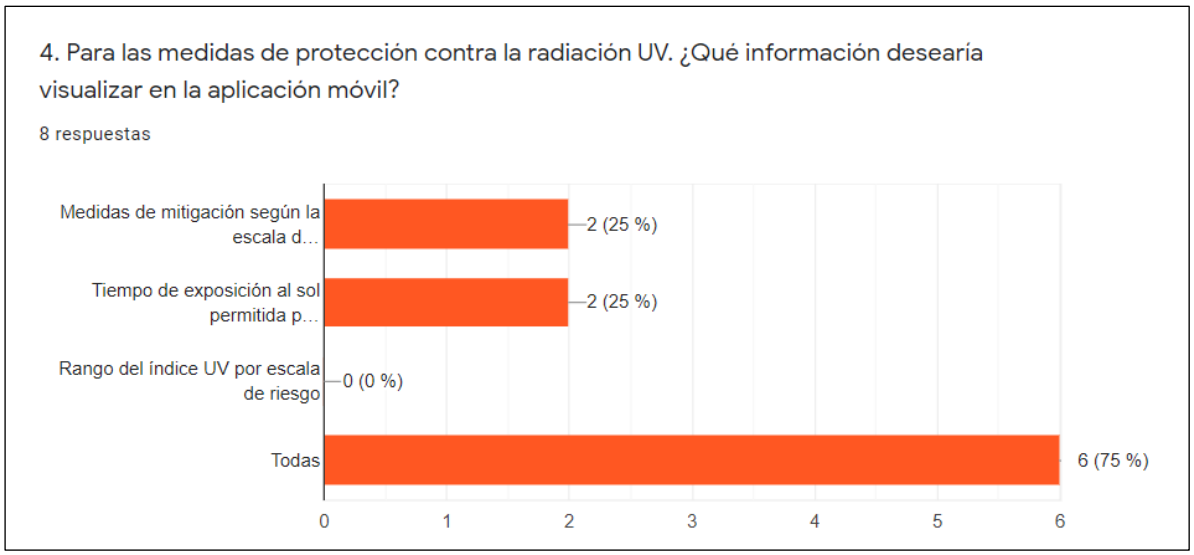


Figura 2.5. Resultados pregunta 4.

Figura 2.6. Resultados pregunta 5.

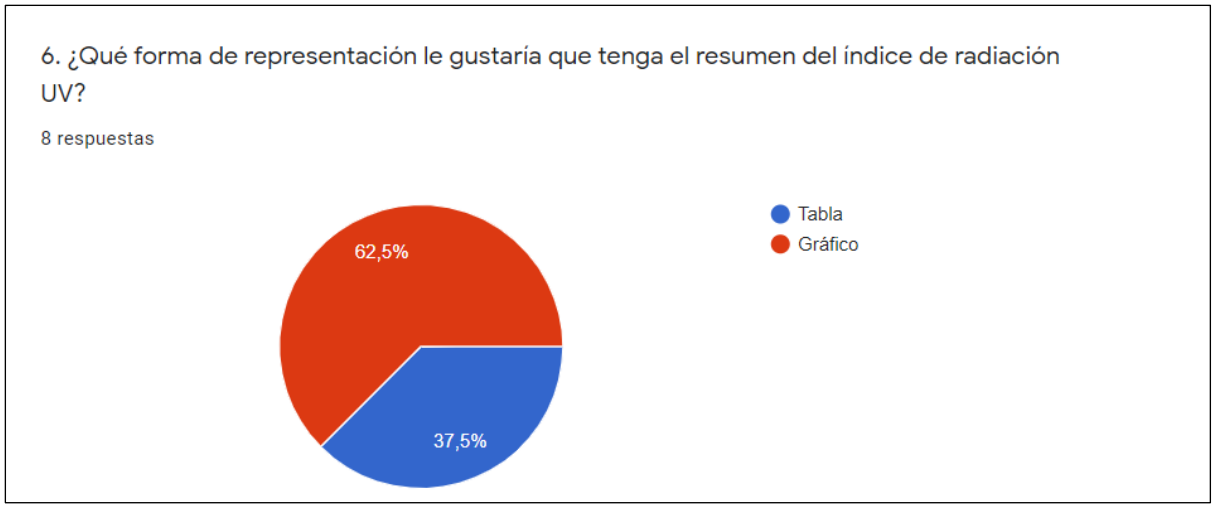


Figura 2.7. Resultados pregunta 6.

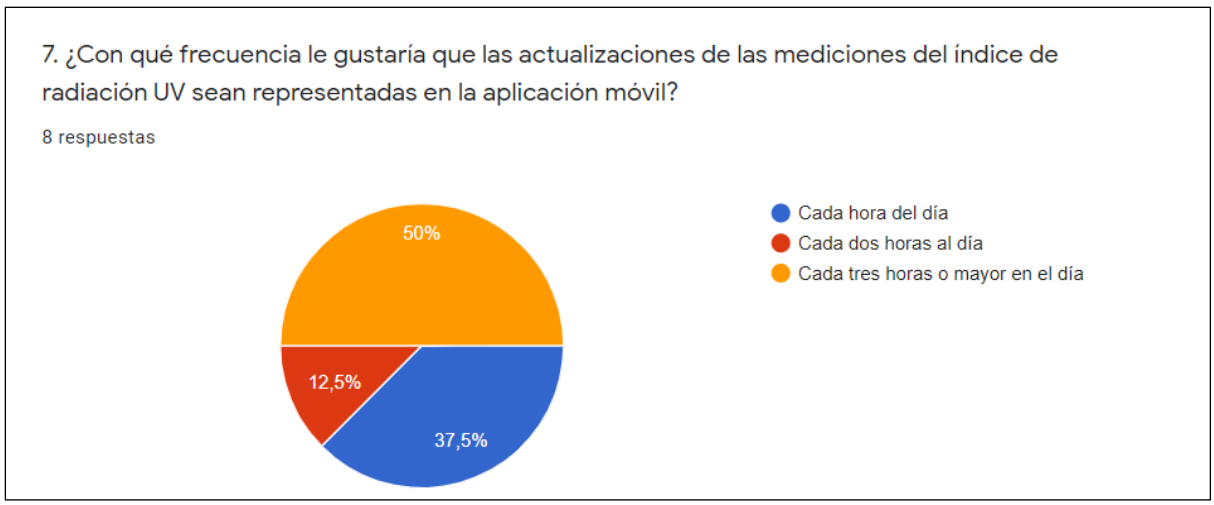


Figura 2.8. Resultados pregunta 7.

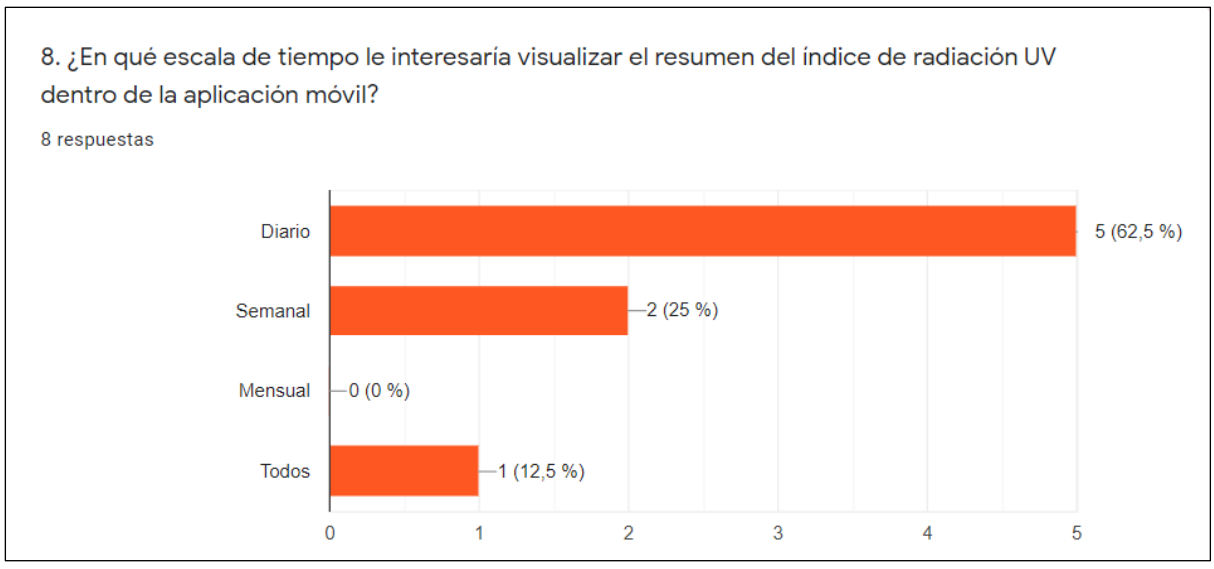


Figura 2.9. Resultados pregunta 8.

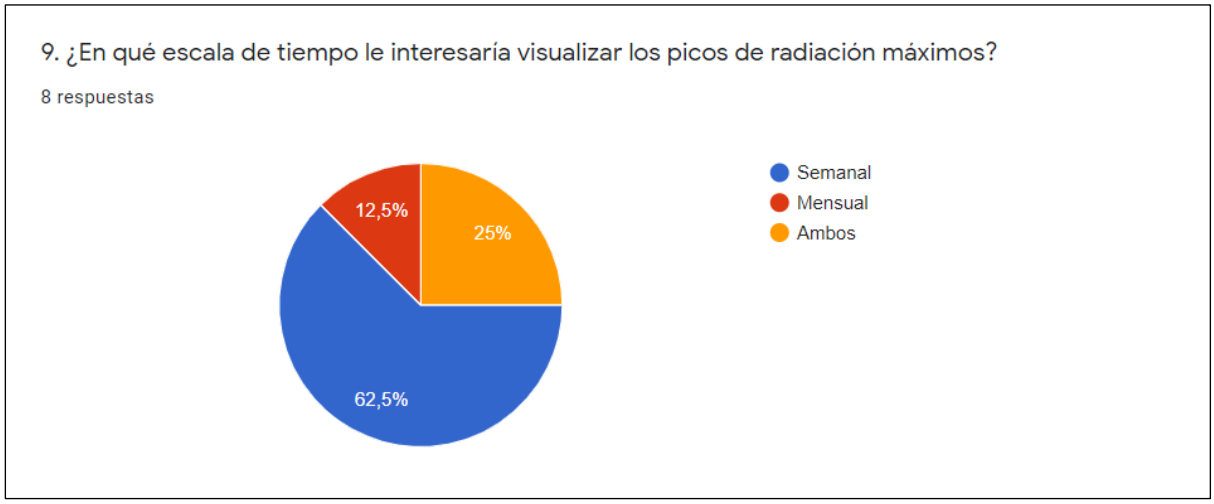


Figura 2.10. Resultados pregunta 9.

Del análisis de los resultados se conoce la cantidad de información que los sensores UV deben enviar y la frecuencia de estas muestras. Además, se plasmarán y desarrollaran los componentes de la aplicación móvil que más valoración tuvieron en las preguntas del cuestionario.

Cabe recalcar que, del análisis del cuestionario realizado, contribuirán en el desarrollo de las historias de usuario, y posteriormente definir los requerimientos funcionales y no funcionales de la aplicación móvil.

2.1.2. HISTORIAS DE USUARIO

La información recolectada de los cuestionarios realizados permite conocer las necesidades y preferencias de los usuarios al interactuar con la aplicación móvil. El análisis de esta información ayuda a desarrollar las historias de usuario, las cuales se convertirán en macro tareas que permitirán definir la etapa de diseño de la aplicación móvil.

Tabla 2.1. Historias de Usuario. Fuente: Ian Baquero.

ID	Título	Descripción	Número de pregunta utilizada del cuestionario
1	Visualización de las muestras	Características de las muestras tomadas por los sensores tales como: nombre del sector, latitud, longitud, índice UV, áreas de	2, 3, 5

	representadas en el mapa	cobertura de los sensores, fecha y hora de la medición. Estas características se mostrarán en un mapa. Además, de una alerta en caso de un índice UV peligroso presente.	
2	Frecuencia de muestras enviadas por los sensores	Las actualizaciones de las muestras tomadas por los sensores deben ser de al menos cada hora, además, la aplicación móvil debe actualizar esta información automáticamente.	7
3	Medidas de protección	Resumen conciso en forma de tabla de las medidas de protección que debe tomar el usuario para protegerse del peligro de la radiación UV, especificado por rango de peligrosidad y tiempo de exposición que puede permanecer el usuario.	4
4	Resumen y picos máximos del índice de radiación UV	Consiste en la representación de la información almacenada en la base de datos en forma de gráfico, con el objetivo de que el usuario vea el resumen del índice de radiación UV diario, además del promedio y los picos máximos del índice de radiación UV en los compendios de tiempo semanal y mensual.	6, 7, 8, 9
5	Configuraciones del rol de administrador	Consiste en tres actividades de uso exclusivo para el administrador de la aplicación, para ello se deberá utilizar un *login* para su acceso y desarrollar las actividades: editar el radio de los hexágonos, descarga de archivo resumen del día seleccionado y visualizar una lista de errores del servicio web.	2

2.1.3. REQUERIMIENTOS DE LA APLICACIÓN

Después de analizar las respuestas de los cuestionarios e historias de usuario, se definen los requerimientos funcionales y no funcionales de la aplicación móvil. No obstante, para los requerimientos no funcionales se involucrará el criterio y experiencia del desarrollador.

Se quiere diseñar un software que tenga los siguientes requisitos:

2.1.3.1. Requerimientos funcionales

- Mostrar un mapa con las mediciones del índice de radiación UV y sus características principales. Estas mediciones se actualizarán en el mapa como subáreas dibujadas en forma de hexágono con su respectivo marcador en el centro del área, haciendo referencia a cada sensor UV que se encuentre operativo en el campo.
- Mostrar una tabla informativa sobre las medidas de protección que debe tomar el usuario y su tiempo estimado de exposición para cada rango de peligrosidad dentro del índice de radiación UV, para que de esta manera se pueda salvaguardar la salud del usuario.
- Mostrar un gráfico dinámico de las mediciones tomadas en el día, así como de los picos máximos y un resumen del índice de radiación UV semanal y mensual para cada sensor UV.
- Se necesita una interfaz exclusiva para el administrador que permita: editar el radio de los hexágonos, descargar un archivo resumen del día especificado y visualizar la lista de errores generados por el servicio web. Esto con la finalidad de tener servicios exclusivos de administrador que justifique su rol en la aplicación móvil.
- Diseñar una aplicación móvil con al menos cuatro interfaces, una para cada módulo.

2.1.3.2. Requerimientos no funcionales

- Desarrollar una aplicación móvil híbrida para los sistemas operativos tanto Android como iOS. Este requerimiento se lo obtuvo con ayuda de la pregunta uno del cuestionario y el criterio personal del desarrollador.
- Utilizar APIs fuera del dispositivo móvil como API REST para disminuir el peso de la aplicación y mejorar la velocidad de procesamiento.
- Implementar una base de datos no relacional MongoDB. Este tipo de base de datos se ajusta a las necesidades del sistema, ya que no existe relaciones entre las muestras almacenadas en la base de datos.

2.1.4. TABLERO KANBAN ETAPA DE ANÁLISIS

Esta etapa consistió en el levantamiento de información a través de los cuestionarios para adquirir información utilizada en el desarrollo de las historias de usuarios, para así, concluir con la elaboración de los requerimientos de la aplicación móvil. Dentro del desarrollo de esta etapa no se presentó tareas adicionales o la adquisición de nuevas tarjetas Kanban.

Todas las tareas han sido completadas de manera exitosa, por lo tanto, las tarjetas Kanban pueden pasar al estado “LISTO” en el tablero como se observa en la Figura 2.11.

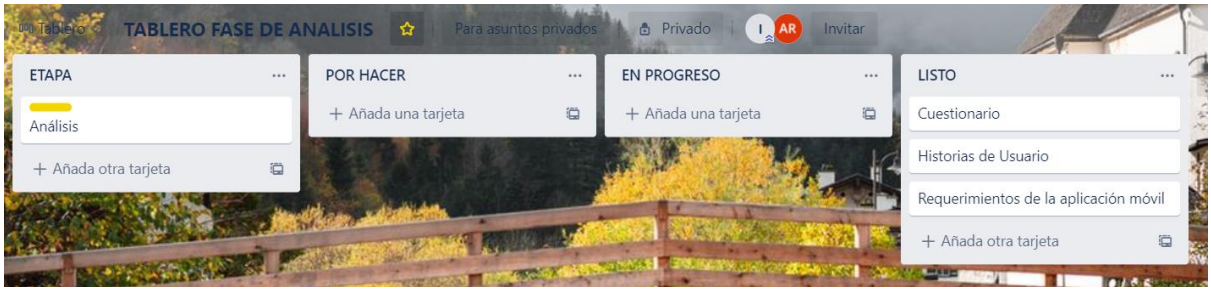


Figura 2.11. Tablero Kanban etapa de análisis estado LISTO. Fuente: Ian Baquero.

2.2. DISEÑO

Se utilizará diagramas de caso de uso para definir las interacciones del usuario con la aplicación móvil y diagramas de clase para representar la estructura de las clases y objetos utilizados. Estos diagramas UML (Unified Modeling Language) se utilizarán para el diseño de la estructura y comportamiento de la aplicación móvil.

Se han desarrollado ocho ítems como se muestran en el tablero Kanban de la Figura 2.12.

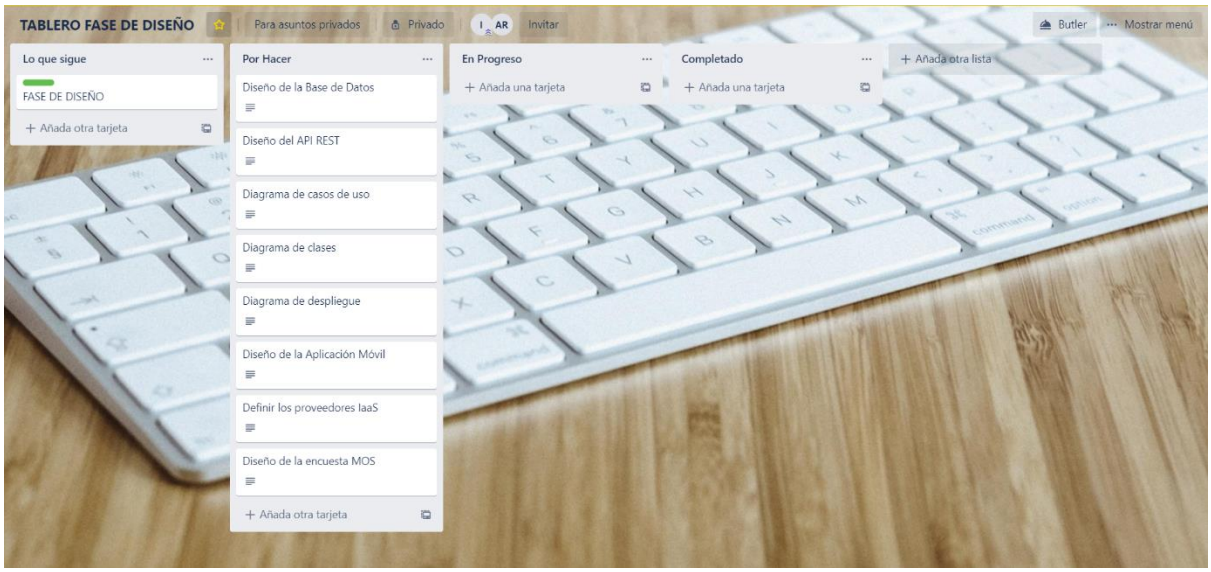


Figura 2.12. Tablero Kanban etapa de diseño estado POR HACER. Fuente: Ian Baquero.

2.2.1. DISEÑO DE LA BASE DE DATOS

La capa de datos se diseñó a través de una base de datos MongoDB llamada “mapa”. Se diseñó dos colecciones en la base de datos. Una para las muestras de la radiación UV tomadas por los sensores conocidas como “radiacions” y otra llamada “logs” para almacenar todos los errores capturados por el servicio web. Los campos de los documentos de la colección “radiacions” son:

- **Ubicación:** Campo del tipo *string* que describe el nombre del sector en donde se encuentra el sensor.
- **UV:** Campo del tipo *number* que describe el índice de radiación UV marcado en ese instante de tiempo.
- **Hora:** Campo del tipo *date* que describe la fecha y hora de la muestra tomada.
- **Latitud:** Campo del tipo *number* que describe la latitud de la posición del sensor.
- **Longitud:** Campo del tipo *number* que describe la longitud de la posición del sensor.

Los campos de los documentos de la colección logs son:

- **Nombre**: Campo del tipo *string* que describe el tipo de error capturado.
- **Mensaje**: Campo del tipo *string* que detalla información del error capturado.
- **Fecha**: Campo del tipo *date* que especifica la fecha y hora del error capturado.

Debido a que los documentos de ambas colecciones no se relacionan entre sí, se optó por implementar una base de datos no relacional.

2.2.2. DISEÑO DEL API REST

La capa lógica se diseñó con una API utilizando un servicio tipo REST el mismo que se alojará en un servicio en la nube de tipo IaaS. Esta proporcionara los servicios disponibles para la aplicación móvil y el servicio de creación de nuevas muestras para los sensores UV.

El diseño de la API cumple con los siguientes servicios o rutas de tipo *get* y *post*, para el rol de usuario y administrador:

- **Marcadores Recientes:** Consiste en un algoritmo que envía un conjunto de objetos conformados por las muestras más recientes de todos los sensores del día actual.
- **Resumen Diario:** Consiste en una consulta que envía un conjunto de objetos conformados por todas las muestras correspondientes a un día especificado por el usuario a través de un calendario, para así, visualizar su comportamiento en un gráfico dinámico.

- **Promedio Semanal:** Consiste en un algoritmo para enviar un conjunto de objetos nuevos por sensor que representarán el promedio UV por día, a partir de un día anterior considerando la fecha actual que se solicite el servicio y de los treinta días anteriores, conformados cada uno por un campo UV que tiene el promedio de los índices de radiación UV de las muestras tomadas dentro de las horas 10 am y 16 pm de todo el día , un campo hora que permite saber el día y el mes que corresponde el promedio realizado y un campo ubicación que especifica el nombre del sector al que hace referencia ese sensor.
- **Promedio Mes:** Consiste en un algoritmo para enviar un conjunto de objetos nuevos por sensor que representarán el promedio UV de los doce meses, conformados cada uno por un campo UV que tiene el promedio de los índices de radiación UV de las muestras tomadas dentro de las horas 10 am y 16 pm de todo el mes, un campo hora que permite saber el mes que corresponde el promedio realizado y un campo ubicación que especifica el nombre del sector al que hace referencia ese sensor.
- **Máximo Semanal:** Consiste en un algoritmo para enviar un conjunto de objetos nuevos por sensor que representarán los picos UV máximos existentes en la base de datos, a partir de un día anterior considerando la fecha actual que se solicite el servicio y de los treinta días anteriores, conformados cada uno por un campo UV que selecciona el máximo índice de radiación UV de las muestras tomadas dentro de las horas 10 am y 16 pm de todo el día, un campo hora que permite saber el día y el mes que corresponde a la selección realizada y un campo ubicación que especifica el nombre del sector al que hace referencia ese sensor.
- **Máximo Mensual:** Consiste en un algoritmo para enviar un conjunto de objetos nuevos por sensor que representarán los picos UV máximos existentes en la base de datos de los doce meses, conformados cada uno por un campo UV que selecciona el máximo índice de radiación UV de las muestras tomadas dentro de las horas 10 am y 16 pm de todo el mes, un campo hora que permite saber el mes que corresponde a la selección realizada y un campo ubicación que especifica el nombre del sector al que hace referencia ese sensor.

El diseño de la API cumple con los siguientes servicios o rutas exclusivas para el rol de administrador:

- **Actualización del radio:** Consiste en una ruta de tipo post para actualizar el radio de los hexágonos automáticamente desde la interfaz de administrador.

- **Mostrar Logs:** Consiste en una ruta de tipo *get* para mostrar todos los errores capturados por todos los servicios web y desplegarlo en una lista dentro de la interfaz de administrador.
- **Limpiar Logs:** Consiste en una ruta de tipo *get* para eliminar todos los documentos de la colección “logs” y recibir como respuesta un JSON vacío.

Cabe mencionar que el servicio Marcadores Recientes utiliza dos métodos adicionales que consisten, el uno en añadir las coordenadas respectivas para que cada área de las muestras pueda ser dibujada en el mapa dependiendo cada una de un radio preestablecido en la API, de la latitud y longitud respectiva por el sensor UV, el otro método consiste en añadir el color respectivo según el índice de radiación UV al área de la muestra dibujada en el mapa.

El diseño de la API también debe cumplir con un servicio o ruta para el sensor UV:

- **Crear**: Consiste en un servicio que permite almacenar nuevas muestras en la colección “radiacions” a través de una ruta de tipo post. Tiene como requisito obligatorio llenar todos los campos de una muestra para su creación.

2.2.3. DIAGRAMA DE CASOS DE USO

Los diagramas de caso de uso sirven para analizar las interacciones de los tipos de usuarios con el sistema, esto permite modelar un comportamiento con el objetivo de comprender el uso de la aplicación móvil. Este tipo de diagrama consiste en tres elementos: actores, casos de uso y relaciones de uso. [33]

Para todos los usuarios o también nombrados con el rol de cliente que hacen uso de la aplicación móvil, pueden visualizar el mapa y las muestras recientes de todos los sensores UV operativos; ver las medidas de protección para los diferentes índices UV y visualizar el gráfico dinámico de todos los sensores UV con sus respectivas variantes de información, como se puede observar en la Figura 2.13.

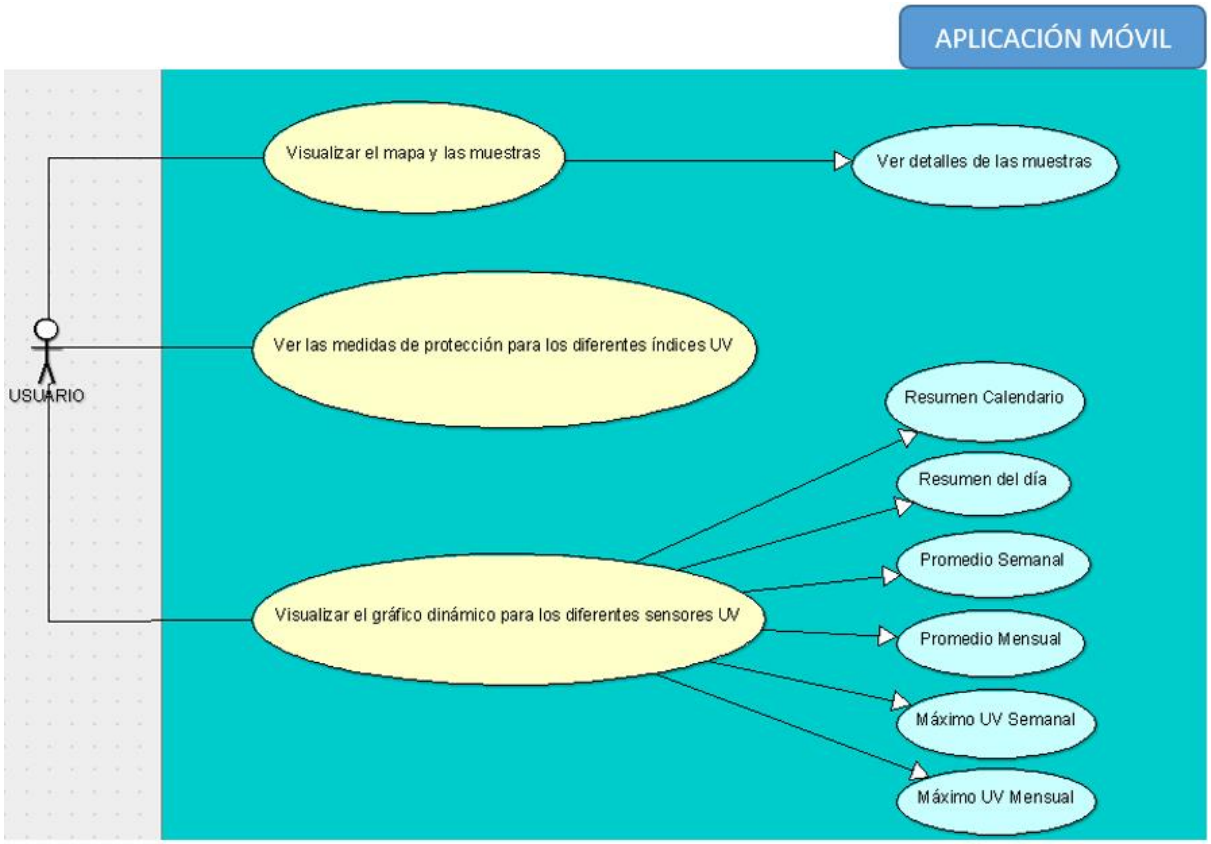


Figura 2.13. Diagrama Caso de Uso: Usuario con rol de cliente. Fuente: Ian Baquero.

Un usuario con el rol de administrador desde el punto de vista de la API puede modificar el radio de las subáreas dibujadas en el mapa de la aplicación móvil, puede descargar en el dispositivo móvil un archivo con el resumen del índice UV de un dia especifico a través del uso de un calendario, ademas de visualizar y limpiar los logs de errores generados en el servicio web como se observa en la Figura 2.14. Existe un usuario del tipo administrador que a través de sus credenciales puede ingresar al rol de administrador en la aplicación móvil.

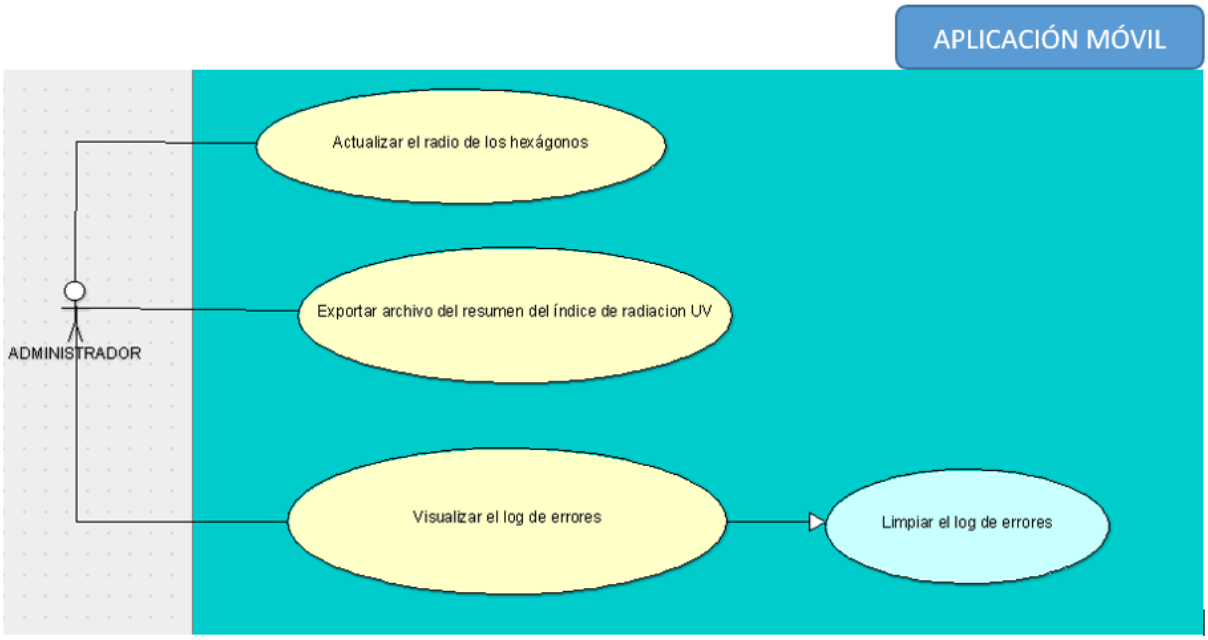


Figura 2.14. Diagrama Caso de uso: Usuario con rol de administrador. Fuente: Ian Baquero.

2.2.4. DIAGRAMA DE CLASES

Los diagramas de clases sirven para presentar la estructura de las clases y sus relaciones en la aplicación. [33] Como se puede ver en la Figura 2.15, se tiene dos clases una llamada Marcador que almacena como atributos todas las características de las muestras para su posterior visualización; y otra clase llamada Polígono que tiene como atributos, características del área del sensor UV dibujada en el mapa y centrada gracias a la clase Marcador, además de incluir un atributo llamado *paths* que almacena un arreglo de coordenadas que permite dibujar un polígono.

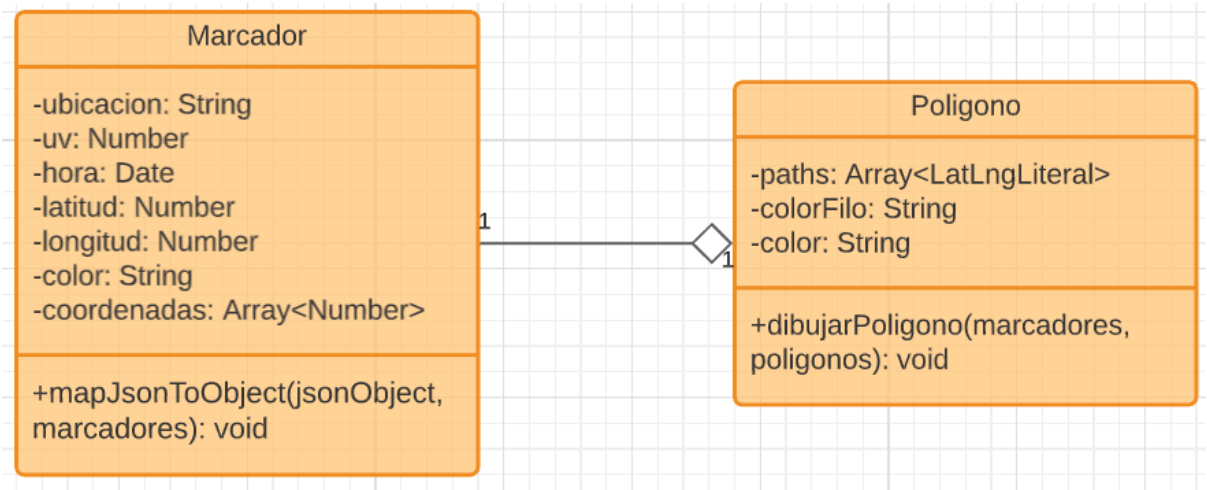


Figura 2.15. Diagrama de clases Fuente: Ian Baquero.

La representación de una relación de agregación en UML es a través de una línea que sale de una clase que representa “una parte de” hacia la clase destino que representa “todo”, con un rombo vacío en la clase por agregación. Como resultado, esta relación de agregación representa que un objeto Marcador es parte de un objeto Polígono. [33]

Para que los objetos de las clases Marcador y Polígono sean visualizados, se necesita crear los arreglos de objetos correspondientes a las clases en el archivo *typescript tab1*, debido a que el archivo *html tab1* encargado de mostrar los cambios en el *front-end* toma los objetos de su correspondiente archivo *typescrit*. Por esta razón, los métodos para la clase Marcador no se los incluyó dentro de la clase.

2.2.5. DIAGRAMA DE DESPLIEGUE

Los diagramas de despliegue sirven para modelar la arquitectura de un sistema. Esto muestra la interacción de los nodos (elementos) y los componentes software de los nodos físicos en el sistema. [33]

Estos tipos de diagrama suelen utilizar el diagrama de componentes para dar una visión más general de cómo está conformado el sistema. [34]

Como se puede observar en la Figura 2.16, se define la arquitectura del sistema utilizando la arquitectura de tres capas. A continuación, se describe los nodos:

- **Aplicación móvil Ionic:** Se integró dentro de este nodo como componentes las interfaces de administrador y usuario. Este nodo utiliza el protocolo de transferencia de información HTTP, para comunicarse y solicitar los servicios alojados en el servidor web.
- **Servidor Web AWS:** Los componentes principales para este nodo son los servicios o rutas, mismas que están conformados en su mayoría por algoritmos y consultas a la base de datos. Utiliza HTTP para comunicarse con la aplicación móvil y TCP/IP como protocolo de comunicación con MongoDB ATLAS.
- **MongoDB ATLAS:** Consiste en una base de datos NoSQL con dos colecciones: radiacions y logs. Este nodo utiliza TCP/IP como protocolo de comunicación con el servidor web, las consultas se solicitan con el lenguaje JavaScript, pasando objetos JSON como parámetros.
- **Prototipos UV:** Los tres prototipos UV están diseñados con un módulo GPRS, el cual se encarga de enviar las muestras con toda su información al servidor web a través del protocolo HTTP.

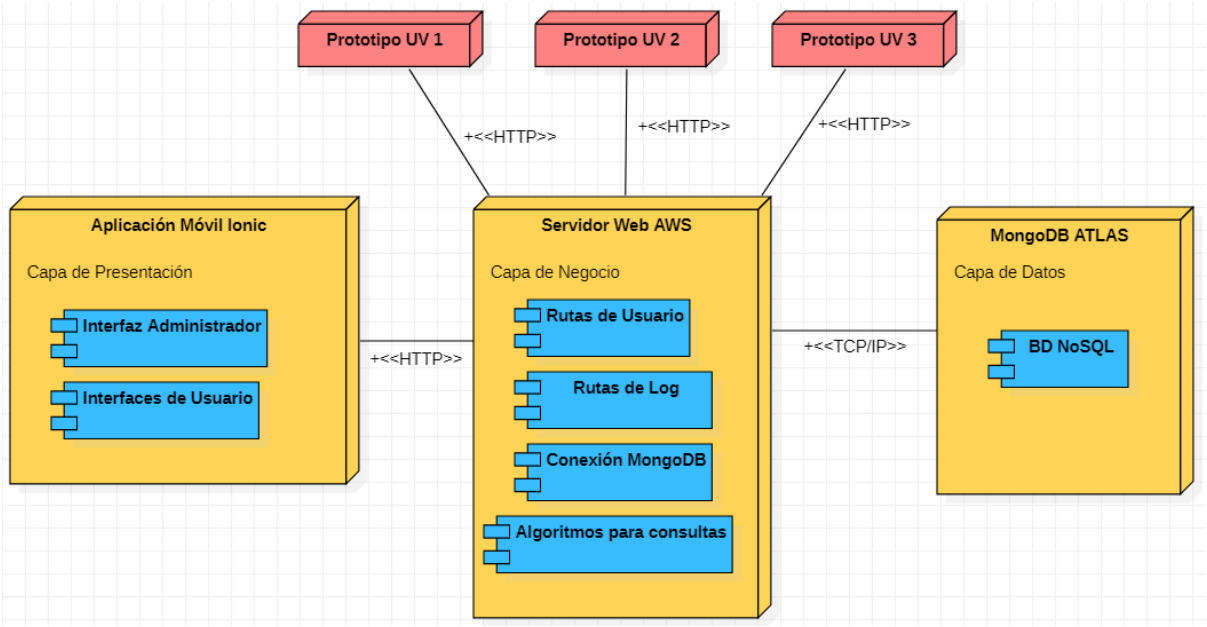


Figura 2.16. Diagrama de despliegue. Fuente: Ian Baquero.

2.2.6. DISEÑO DE LA APLICACIÓN MÓVIL

La capa de presentación es la encargada de mostrar la información al usuario. Después de analizar los requerimientos funcionales de la aplicación se optó por utilizar una plantilla tipo *tabs* como punto de partida, la cual consiste de interfaces navegables ubicadas en la parte inferior de la aplicación, similares a como se muestra en la Figura 2.17

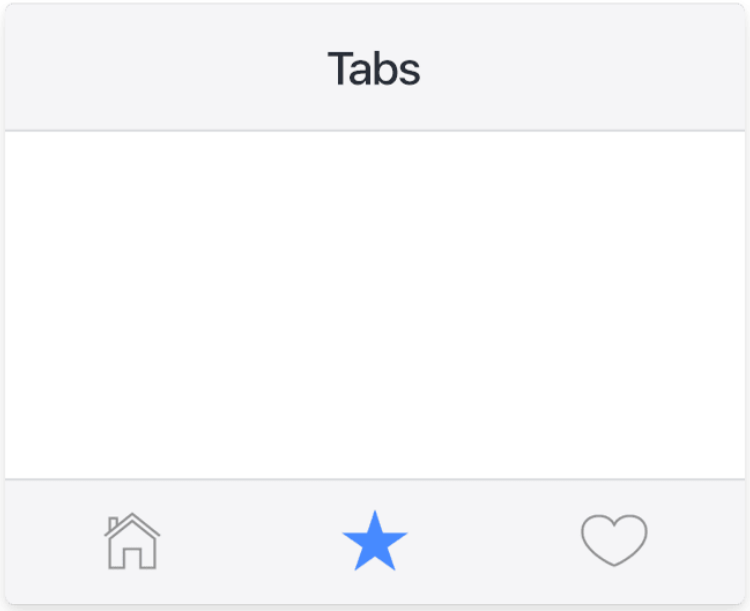


Figura 2.17 Plantilla tipo *tab* [35]

Cada *tab* representa una interfaz nueva suficiente para implementar cada servicio que dispone la aplicación. Los servicios por implementar en cada *tab* serán los siguientes:

- Mapa.
- Recomendaciones de protección UV.
- Gráfico dinámico.
- Administrador.

2.2.7. SELECCIÓN DEL PROVEEDOR DE SERVICIO IAAS

En esta sección se buscará un servicio IaaS en la nube que se ajuste a las demandas de nuestra aplicación, para ello, se ha considerado algunos factores relevantes que se debe considerar al elegir un proveedor IaaS para una aplicación con necesidades similares a las de este Trabajo de Titulación:

- **Ambiente:** Es importante que el proveedor cuente con el sistema operativo y soporte para la base de datos que se tenga pensado implementar.
- **Nivel de servicio:** En este inciso se toma en cuenta la disponibilidad del servicio IaaS y se debe revisar el SLA (Acuerdo de Nivel de Servicio) que se ofrece y conocer las medidas que se llevan a cabo en caso de caídas, incumplimiento de la disponibilidad del servicio, etc.
- **Pago por uso como opción:** Este modelo de pago es ideal para organización que pueden sufrir pico de alta demanda, en nuestro caso se ajusta perfectamente los programas gratuitos.
- **Configuración y tiempo de respuesta:** El cliente debe tener la capacidad de acceder a su servicio en la nube de forma privada; realizar modificaciones en su configuración y contar con el cambio de forma inmediata. [36]

Se buscó información de los proveedores de IaaS en el mercado y los que se ajustaron a las necesidades fueron Amazon Web Services (AWS) y Microsoft Azure. A continuación, se detallarán algunas de las características comunes de interés de ambos proveedores:

- **Ambiente:** Soportan sistemas operativos Linux, Windows y una base de datos en la nube llamada MongoDB Atlas desarrollada para AWS y Azure.
- **Nivel de servicio:** Se comprometen en brindar una disponibilidad del 99,99% en cada región.
- **Pago por uso como opción:** Ambos proveedores cuentan con programas de prueba gratuita durante doce meses con características similares de uso y 750 horas al mes.

- **Configuración y tiempo de respuesta:** Los clientes pueden ingresar de manera privada a la API a través de sus credenciales.

Algunas principales diferencias entre AWS y AZURE se mencionarán a continuación:

- Las máquinas virtuales en Azure están configuradas previamente por un tercer y deben especificar la cantidad de núcleos y memoria necesaria, mientras que las máquinas virtuales en AWS se conocen con el nombre de EC2 y su propio sistema de memoria virtual puede ser configurada por los usuarios.
- En base al almacenamiento temporal AWS crea instancias cuando se inicia y se destruye cuando se finalice. Mientras que Azure ofrece almacenamiento en bloque a través de Blobs de página para máquinas virtuales.
- AWS tiene más configuraciones y características; además, ofrece más flexibilidad y personalización con soporte para la integración de muchas herramientas de terceros. Azure por otro lado será más fácil de usar si se está familiarizado con Windows, ya que su plataforma se basa en Windows. [37]

Ambas soluciones son válidas para el despliegue de los servicios web del Trabajo de Titulación, ambos proveedores cumplen con las consideraciones planteadas por lo que su selección es indistinta.

La flexibilidad de configuración del sistema de memoria virtual, su creación de instancias como almacenamiento temporal son características que se tomaron en cuenta para seleccionar a AWS como el proveedor de servicios IaaS a utilizar en la etapa de implementación.

2.2.8. DISEÑO DEL TEST MOS

Consiste en medir la calidad de experiencia del usuario (QoE), después de utilizar la aplicación móvil. Por lo tanto, esta prueba utilizara un método de respuesta cuantificada para medir la satisfacción del usuario en el test de opinión. Los participantes de este experimento son estudiantes universitarios de entre dieciocho a treinta años.

La siguiente escala se considera en el experimento:

Tabla 2.2. Escala de opinión. Fuente: Ian Baquero.

Tipificación	Puntaje
Excelente	5
Muy Bueno	4

Bueno	3
Mejorable	2
Muy Mejorable	1

Tabla 2.3. Escala de dificultad. Fuente: Ian Baquero.

Tipificación	Puntaje
SI	1
NO	0

2.2.8.1. Preguntas del test

Para la conformación del test se presenta el siguiente formato:

Tabla 2.4. Tabla de opinión. Fuente: Ian Baquero.

Preguntas	Excelente	Muy Bueno	Bueno	Mejorable	Muy mejorable
¿Cómo calificaría los siguientes parámetros de la aplicación?					
Diseño de interfaz					
Facilidad de uso					
Interactividad					

La pregunta diseño de interfaz hace referencia a la presentación física de las interfaces, la facilidad de uso a la intuición del usuario al manejar la aplicación móvil por primera vez y la interactividad a la cantidad de elementos con los que el usuario puede interactuar para una aplicación móvil de este tipo.

Tabla 2.5. Tabla de dificultad. Fuente: Ian Baquero.

Preguntas	Si	No
¿Tuvo problemas en el acceso a la aplicación?		
¿La aplicación tuvo algún error de ejecución?		
¿La aplicación tuvo algún error al mostrar el resumen del gráfico dinámico?		
¿La aplicación mostró las subáreas en el mapa?		

2.2.8.2. Resultados

Para los resultados se tomará en cuenta unicamente la escala de opinión, en la cual se tabularán las respuestas y se aplicará una regla de tres para obtener el resultado sobre cinco puntos. La siguiente tabla detalla la interpretación de este puntaje.

Tabla 2.6. Interpretación de resultados cuantitativos. Fuente: Ian Baquero.

Puntaje	Interpretación
Igual a 5	Es excelente.
Entre 4 y 5	Es muy buena, pero podría ser excelente después de realizar algunas mejoras.
Igual a 4	Es muy buena.
Entre 3 y 4	Es buena, pero podría ser muy buena después de realizar algunas mejoras.
Igual a 3	Es buena.
Entre 2 y 3	Es mejorable, pero podría ser buena después de realizar algunas mejoras.
Igual a 2	Es mejorable.
Entre 1 y 2	Es muy mejorable y requiere ciertas mejoras.
Igual a 1	Es muy mejorable.

2.2.9. TABLERO KANBAN ETAPA DE DISEÑO

Esta etapa consistió en un estudio de manera general de la estructura del sistema a implementar. Se analizaron los proveedores de servicio en la nube; se diseñan los diagramas UML y los componentes que conformarán el sistema. Dentro del desarrollo de esta etapa no se presentó tareas adicionales o la adquisición de nuevas tarjetas Kanban.

Todas las tareas han sido completadas de manera exitosa, por lo tanto, las tarjetas Kanban pueden pasar al estado “LISTO” en el tablero como se observa en la Figura 2.18.

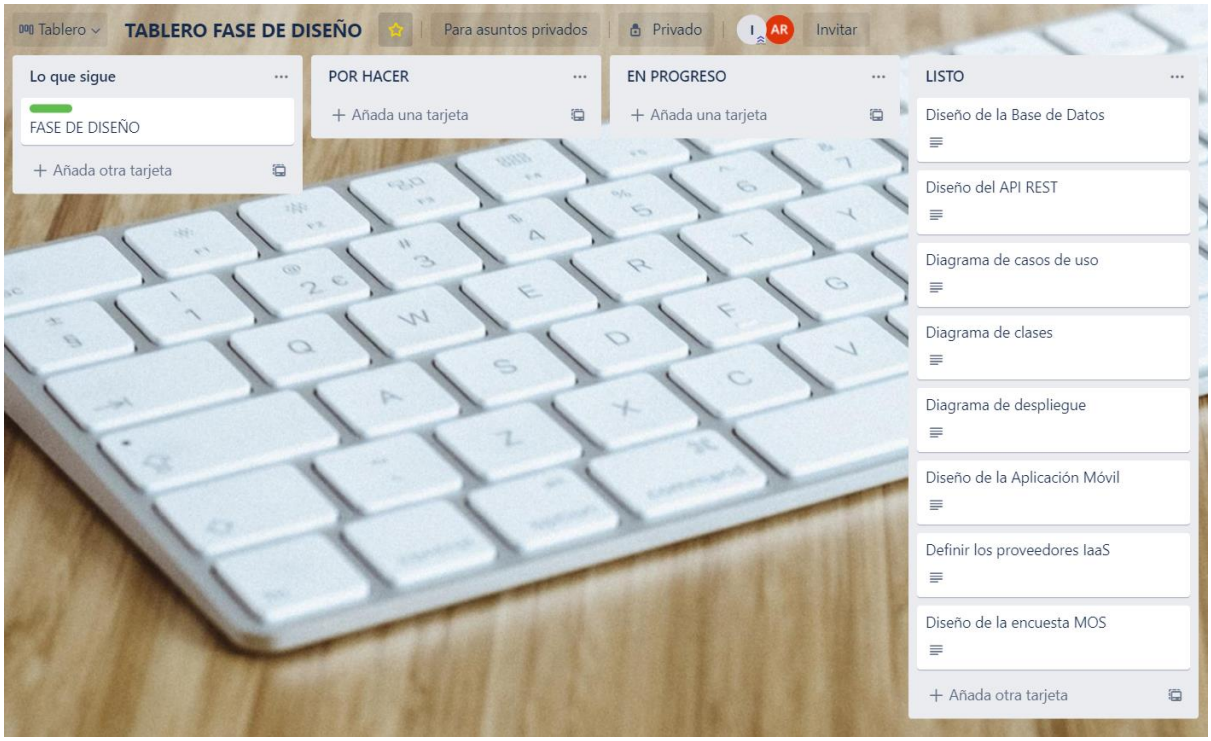


Figura 2.18. Tablero Kanban etapa de diseño estado LISTO. Fuente: Ian Baquero.

2.3. IMPLEMENTACIÓN

En esta etapa se describirá el proceso de implementación de la aplicación móvil. Se detallará los pasos a seguir para crear una base de datos, codificar el servicio web con sus respectivos módulos, implementar las interfaces de la aplicación móvil y sus componentes. Finalmente se contratará el servicio IaaS en la nube y se instalará el servicio web, una base de datos en MongoDB Atlas y se publicará la aplicación móvil en las diferentes tiendas virtuales para los sistemas operativos iOS y Android.

La implementación del servicio web será a través del *framework* Express que utiliza NodeJS como entorno de ejecución para JavaScript, la base de datos será implementada utilizando MongoDB y se utilizará “Ionic” como *framework* para la creación de la aplicación móvil híbrida.

Se definió cinco ítems de trabajo para esta etapa, los mismo que se pueden observar en la columna “Por Hacer” del tablero Kanban en la Figura 2.19.

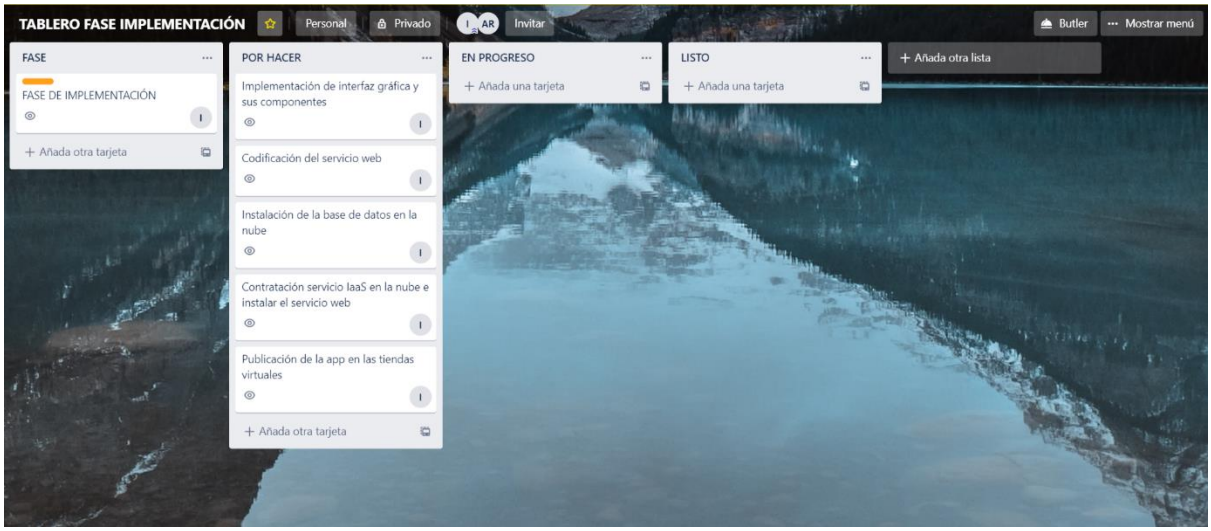


Figura 2.19. Tablero Kanban etapa de implementación. Fuente: Ian Baquero.

A continuación, se desarrollará todos los ítems definidos en el tablero Kanban.

2.3.1. IMPLEMENTACIÓN DEL BACK-END

2.3.1.1. Configuración del servidor web

Antes de empezar a codificar el servicio web, se debe instalar NodeJS, MongoDB y el gestor de base de datos Robo 3T para manipular la capa de datos de manera local en primera instancia.

Una vez realizadas las instalaciones necesarias, desde la ventana de comandos (CLI) dentro de un nuevo directorio de trabajo ejecutamos el comando: $npm init.

Esto inicializa un proyecto NodeJS creando un archivo package.json que tiene toda la información de cómo trabaja esta aplicación. Dentro del archivo package.json por defecto la etiqueta “main” busca el archivo index.js para iniciar el servidor como se observa en la Figura 2.20.

```
{
  "name": "mapa1-server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

Figura 2.20. Archivo package.json. Fuente: Ian Baquero.

Después se ejecuta el comando: $tsc –init

Esto crea un archivo de inicialización “TypeScript” de ahí viene el nombre del comando *tsc* compilador de “TypeScript”. Este nuevo archivo especifica todas las reglas de validación que “TypeScript” usa al momento de compilar todos los archivos de TypeScript a JavaScript.

Una vez ejecutado el comando ***tsc –init*** dentro del archivo tsconfig.json se busca la etiqueta “outDir” la cual viene comentada por defecto, se descomenta y se agrega un nombre para crear un directorio donde se guarden todos los archivos compilados JavaScript como se observa en la Figura 2.21.

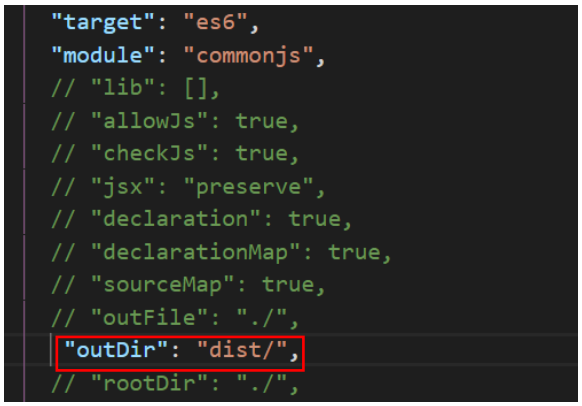


Figura 2.21. Configuración del archivo tsconfig.json. Fuente: Ian Baquero.

A través del comando ***tsc –w*** se compila automáticamente todos los archivos TypeScript a JavaScript en la carpeta especificada de la etiqueta “outDir”. Para ejecutar la codificación configurada en el archivo index.ts se ejecuta el comando:

$node dist/index.js

A continuación, se instalarán algunos paquetes necesarios para configurar nuestro servidor de Express:

- **Express:** Utilizado para crear un servidor web y las configuraciones necesarias para implementar servidor REST.
- **Body-parser:** Permite manejar la información que se recibe de una petición HTTP-POST y transformarla en un objeto JavaScript del lado de Node.
- **Mongoose:** Utilizado para trabajar con el modelado de datos del lado de Node y realizar las interacciones con la base de datos.

2.3.1.2. Codificación del servicio web

2.3.1.2.1. Instancia de Express

Se codifica el archivo de escucha del servidor web para tener una dirección IP y un puerto que este pendiente de todos los cambios que reciba la carpeta “dist”, para realizarlo se importa el modulo Express a un nuevo archivo con el nombre de server y se crea los atributos *app* y *port* como se observa en la Figura 2.22.

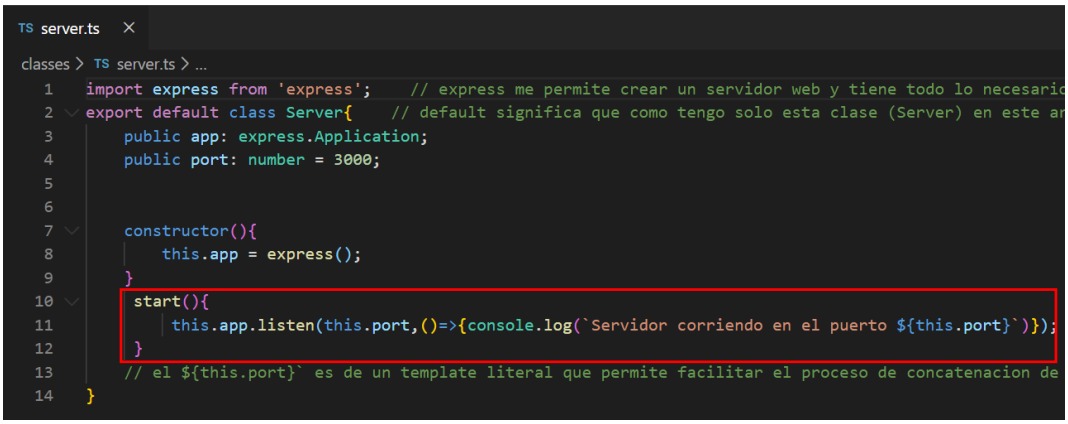


Figura 2.22. Configuración de la instancia Express. Fuente: Ian Baquero.

A través del atributo app se inicializa con la propiedad express.Application para crear la instancia de Express, la cual utilizando su método *listen*, el servidor entra en modo de escucha. El atributo *port* es el puerto por el que nuestro servidor web está atendiendo las peticiones.

2.3.1.2.2. Creación del servicio REST en Express

Como se crea dos colecciones en la base de datos mapa, una para las muestras de los sensores UV llamada radiacions y otra para almacenar los errores capturados por los servicios web llamada logs. Por lo tanto, se crea dos archivos de rutas los cuales utilizan la clase Router como se observa en el código de la Figura 2.23. para radiacions y en el código de la Figura 2.24. para logs; la instancia de esta clase reconoce a Express, el mismo que cuenta con todas las peticiones REST tradicionales para crear los diferentes servicios web para cada colección. Además, las rutas creadas en estos archivos definen la extensión de la dirección URL, para que los servicios web puedan ser diferenciados unos de otros.

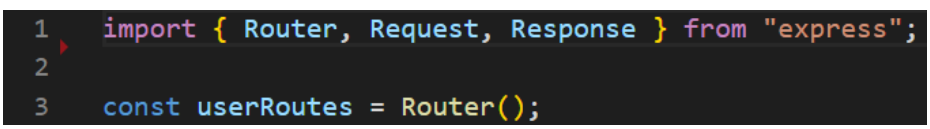


Figura 2.23. Instancia Router para rutas de la colección radiacions. Fuente: Ian Baquero.

```
1   import { Router, Request, Response } from "express";
2
3   const logRoutes = Router();
```

Figura 2.24. Instancia Router para rutas de la colección logs. Fuente: Ian Baquero.

Es importante exportar estas instancias Router al archivo index para que el servidor pueda acceder a todas las rutas creadas como se observa en el código de la Figura 2.25. Cabe recalcar que las rutas creadas son los servicios web que el servidor dispone para la aplicación móvil y el sensor UV. Además, todas las rutas se extienden de la ruta "/api", como ruta inicial.

```
22   // rutas de mi aplicacion
23   server.app.use('/api',userRoutes);
24   server.app.use('/api',logRoutes);
```

Figura 2.25 Configuración del acceso a las rutas. Fuente: Ian Baquero.

Como se observa en el código de la Figura 2.26, se tiene un ejemplo de un servicio REST que utiliza el método *get* y la ruta "/prueba" para recibir el mensaje "Todo funciona bien!". Para comprobar la funcionalidad del servicio REST, se puede utilizar herramientas software como Postman para generar peticiones HTTP como se observa en la Figura 2.27.

```
1   import { Router, Request, Response } from "express";
2
3   const userRoutes = Router();
4
5   userRoutes.get('/prueba',(req:Request, res: Response)=>{
6       res.json({
7           ok:true,
8           mensaje: 'Todo funciona bien!'
9       })
10  });
```

Figura 2.26. Ejemplo de configuración del archivo de rutas. Fuente: Ian Baquero.

Figura 2.27. Respuesta petición a ruta /api/prueba Fuente: Ian Baquero.

2.3.1.2.3. Conexión a la base de datos MongoDB

Se importa el paquete mongoose instalado anteriormente en el archivo index y se crea la conexión de la instancia mongoose con el método *connect* en el cual se define la dirección local, el puerto y la base de datos a la cual se desea conectar. En caso de que no exista la base de datos especificada en el método *connect*, MongoDB se encarga de crearla. En el código de la Figura 2.28. se observa un ejemplo de conexión del objeto mongoose a la base de datos MongoDB a través de su dirección IP, puerto y base de datos llamada mapa.

```
25  // conectar DB mongo con node
26  mongoose.connect('mongodb://localhost:3000/mapa ',
27                   {useNewUrlParser: true, useCreateIndex: true},(err)=>{
28                   if (err) throw err;
29                   console.log('Base de datos OnLine no se cayo..!!!')
30                   })
31
```

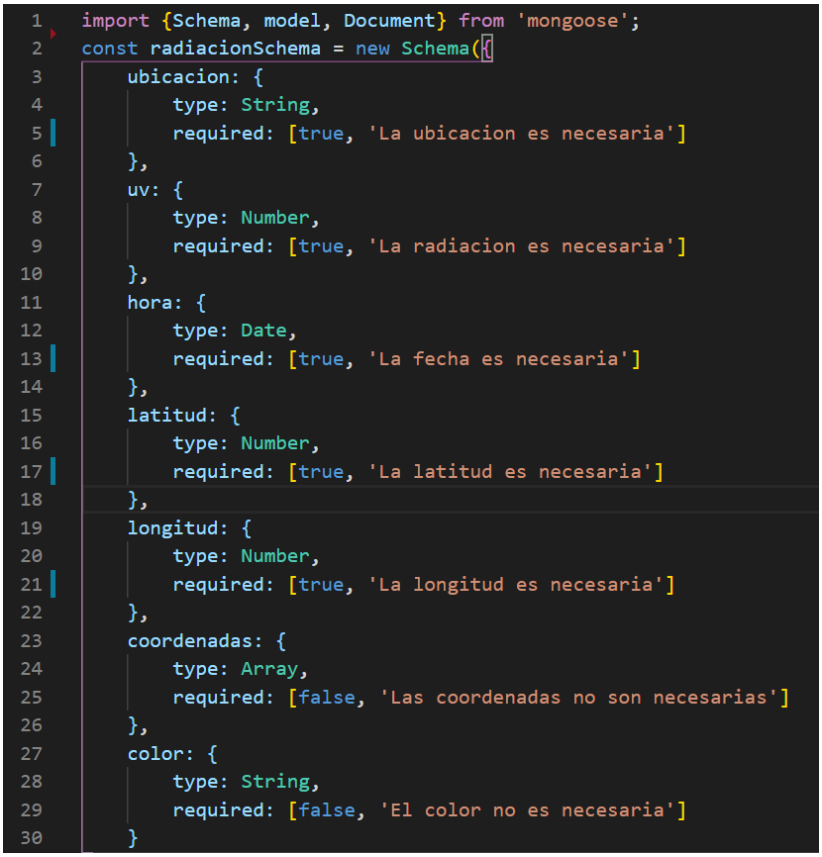
Figura 2.28. Configuración de la conexión a la base de datos MongoDB. Fuente: Ian Baquero.

- **Modelo de Base de Datos**

Un esquema se puede explicar como un registro de bases de datos, el cual se configura en el servidor y en caso de insertar un objeto, se utilizará este esquema para validar la inserción en caso de tener todas las reglas o caso contrario se enviará un mensaje de no se pudo insertar por falta de campos.

2.3.1.2.4. Esquema radiación

Se asigna el esquema radiacionSchema a la colección MongoDB llamada “radiacions” y se define la forma de los objetos o documentos dentro de esta colección. A continuación, en el código de la Figura 2.29. se muestra la configuración del esquema para las muestras de los sensores UV.



```
import {Schema, model, Document} from 'mongoose';
const radiacionSchema = new Schema({
    ubicacion: {
        type: String,
        required: [true, 'La ubicacion es necesaria']
    },
    uv: {
        type: Number,
        required: [true, 'La radiacion es necesaria']
    },
    hora: {
        type: Date,
        required: [true, 'La fecha es necesaria']
    },
    latitud: {
        type: Number,
        required: [true, 'La latitud es necesaria']
    },
    longitud: {
        type: Number,
        required: [true, 'La longitud es necesaria']
    },
    coordenadas: {
        type: Array,
        required: [false, 'Las coordenadas no son necesarias']
    },
    color: {
        type: String,
        required: [false, 'El color no es necesaria']
    }
```

Figura 2.29. Configuración del schema radiación. Fuente: Ian Baquero.

Como se muestra en el *schema* de la Figura 2.29. las variables ubicación, uv, hora, latitud y longitud son obligatorias para validar la inserción de una muestra en la colección, mientras que las variables coordenadas y color son opcionales.

Finalmente se exporta el esquema utilizando la instancia *model* como se observa en el código de la Figura 2.30. Model define un modelo o lo recupera; además, los modelos definidos en la instancia de moongose están disponibles para todas las conexiones creadas por la misma instancia de moongose.

```
interface IRadiacion extends Document{
    ubicacion: string;
    uv: number;
    hora: Date;
    latitud: number;
    longitud: number;
    coordenadas: number [];
    color: string;
}

export const Radiacion = model<IRadiacion>('Radiacion',radiacionSchema);
```

Figura 2.30. Exportación del modelo Radiación. Fuente: Ian Baquero.

De manera opcional se extendió el *model* con una interfaz IRadiacion como se observa en el código de la Figura 2.30. para conocer las variables que tiene el objeto cuando se crea o recupera.

Un modelo al ser definido por un esquema, conoce la estructura de los objetos dentro de una colección en este caso del tipo “Radiacion”, esto permite utilizar las propiedades de una base de datos como búsqueda, inserción, eliminación, etc.

2.3.1.2.5. Esquema log

Se asigna el esquema logSchema a la colección MongoDB llamada “logs” y se define la forma de los objetos o documentos dentro de esta colección. A continuación, en el código de la Figura 2.31. se muestra la configuración y estructura del esquema para los *logs*.

```
import {Schema, model, Document} from 'mongoose';
const logSchema = new Schema({
    nombre: {
        type: String,
        required: [true, 'El nombre del Log es necesario']
    },
    mensaje: {
        type: String,
        required: [true, 'El mensaje del Log es necesario']
    },
    fecha: {
        type: Date,
        required: [true, 'La fechas del Log es necesario']
    }
});
interface ILog extends Document{
    nombre: string;
    mensaje: string;
    fecha: Date;
}
export const Log = model<ILog>('Log',logSchema);
```

Figura 2.31. Configuración y exportación del modelo log. Fuente: Ian Baquero.

De manera opcional se extendió el *model* con una interfaz IRadiacion como se observa en el código de la Figura 2.31. para conocer las variables que tiene el objeto cuando se crea o recupera.

Finalmente se exporta el esquema utilizando la instancia *model* como se observa en el código de la Figura 2.31. Model define un modelo o lo recupera; además, los modelos definidos en la instancia de moongose están disponibles para todas las conexiones creadas por la misma instancia de moongose.

2.3.1.2.6. Crear un documento en la colección radiacions

Se define el middleware a través del paquete bodyParser en el archivo index como se observa en el código de la Figura 2.32, para que de esta manera se pueda manipular y obtener los atributos de una petición POST, con el objetivo de convertir el objeto JSON de la petición en un objeto JavaScript.

```
12  server.app.use(bodyParser.urlencoded({extended: true}));
13  server.app.use(bodyParser.json());
```

Figura 2.32. Configuración del middleware. Fuente: Ian Baquero.

Una vez definido el archivo de rutas Radiacion, se codifica la ruta o servicio web para insertar una muestra en la colección radiacions, para ello se utiliza el método POST disponible en los métodos de las rutas y se define lo que se va a recibir y lo que se responde en caso de una inserción exitosa o fallida, tal como se muestra en el código de la Figura 2.33.

```
179  userRoutes.post('/create', (req: Request,res: Response, next)=>{
180      const radiacion = {
181          ubicacion: req.body.ubicacion,
182          uv: req.body.uv,
183          hora: req.body.hora,
184          latitud: req.body.latitud,
185          longitud: req.body.longitud
186      };
187      Radiacion.create( radiacion).then(radiacionDB=>{ // radiacion
188          res.json({ // esta es la respuesta que le voy a mandar al
189              ok:true,
190              radiacion: radiacionDB // muestro el item radiacion c
191          });
192      }).catch(err => {
193          next(err); // genera el error con un response
194      })
195  });
```

Figura 2.33. Servicio web para inserción de muestras. Fuente: Ian Baquero.

El objeto radiación se creó con la finalidad de tener un objeto JavaScript con todos los campos del objeto JSON recibido en la petición.

Como “Radiacion” es un modelo de mongoose se tiene todas las propiedades de una base de datos. Se utiliza el método *create* para insertar las muestras recibidas por las peticiones POST del sensor UV en la colección radiacions.

2.3.1.3. Servicios web del modelo radiación

Debido a que no existe una consulta para filtrar objetos con todas las especificaciones que necesitamos, se desarrolló algoritmos para satisfacer las necesidades. Se escogerá un algoritmo base “marcadoresRecientes” para explicar el desarrollo de los demás algoritmos.

Se define la ruta “/recientes” con el método *get*, ya que no necesita parámetros de entrada, como resultado se espera recibir la muestra más reciente de todos los sensores operativos en el día. Para ello se instancia una nueva variable tipo *date* llamada hoy y se modifica la hora con el valor de cero con el objetivo de tener una variable de tipo *date* de referencia para filtrar las muestras respecto a la fecha actual con hora cero.

Se define el modelo con el método *find* y dentro de esta un condicional para buscar las muestras con fechas y horas mayores respecto a la que se definió en la variable hoy (variable que define la fecha actual y hora cero), acompañado del método *sort* para ordenar las muestras en este caso por horas, tal como se observa en la Figura 2.34.

```
31  userRoutes.get('/recientes',(req: Request, res:Response, next)=>{ // envia las ultima
32      var hoy = new Date(); // obtenemos el dia, mes y anio del dia de hoy
33      hoy.setHours(0); // seteamos a 0 la hora debido a que nos interesa obtener todas
34      Radiacion.find({hora:{$gt:new Date(hoy)}}).sort({hora:1}).exec((err,radiacion)=>{
35          if(err){
36              err.message = 'Error en el servidor';
37              next(err);
38          }else{
39              if(Object.keys(radiacion).length !== 0){
40                  radiacion = Metodos.marcadoresRecientes(radiacion);
41                  for(var marcador of radiacion){     // añadimos las coordenadas y el
42                      marcador.coordenadas = coordenadas;
43                      marcador.color=Metodos.escogerColor(marcador.uv);
44                  }
45                  res.status(200).send({
46                      radiacion
47                  });
48              }else{
49                  var err = new Error('No hay muestras en la base');
50                  next(err);
51              }
52          }
53      });
54  });
```

Figura 2.34. Configuración de la ruta “/recientes”. Fuente: Ian Baquero.

Dentro del modelo se define condicionales en caso de que se llegase a presentar un error en el servidor o no existan muestras recientes en la base. En caso de que sea exitosa la consulta en la colección, se recibirá todas las muestras del día actual, para escoger la muestra más reciente se llama al método marcadoresRecientes y se le envía como parámetro de entrada el conjunto de objetos recibidos de la consulta realizada para que a través del algoritmo obtener el servicio esperado. Posteriormente se agrega a las muestras recientes obtenidas, las variables de las coordenadas del método calcularCoordenadas inicializadas al inicio del archivo, la cual se explicará posteriormente y se llama al método escogerColor para asignar los colores respectivos al índice UV de los objetos de marcadoresRecientes.

2.3.1.3.1. Método marcadoresRecientes

Todos los algoritmos utilizados en el servicio web, se encuentran almacenados en el archivo métodos el mismo que es exportado para ser utilizado en las rutas respectivas.

El algoritmo de este método se definirá como algoritmo base, el mismo que será replicado con algunas modificaciones en los otros métodos de los servicios web respectivos. Como parámetro de entrada se tiene un *array* de objetos llamada "radiacion" el cual almacena todas las muestras recibidas de la consulta realizada previamente.

Se define las siguientes variables a utilizar en el método, como se observa en el código de la Figura 2.35:

- **ubicaciones**: Variable de tipo *array strings*, utilizada para almacenar los nombres de los sectores de todos los sensores UV recibidos en la consulta.
- **eliminados**: Variable de tipo *array number*, utilizada para almacenar el índice de los objetos que se eliminaran en el parámetro de entrada.
- **i**: Variable de tipo *number*, utilizada para llevar la cuenta del índice de los objetos dentro de las iteraciones realizadas por lazos.
- **x**: Variable de tipo *number*, utilizada para ayudar a eliminar los objetos en el parámetro de entrada según el índice.
- **bandera**: Variable del tipo *boolean*, utilizada para inicializar una variable.
- **reciente**: Variable del tipo *object*, utilizado como la muestra que será actualizado cada vez que se encuentre una muestra con una hora mayor.

```
34      static marcadoresRecientes(radiacion:any){
35      var ubicaciones: string[] = []; // guarda las ub
36      var eliminados: number[] = []; // almacenara los
37      let i:number;  // utilizado para llevar la cuent
38      let x:number=0;     // utilizado para ayudar a l
39      let bandera: boolean; // sirve para entrar a un
40      let reciente;     // objeto que almacenara al pr
```

Figura 2.35. Variables del método marcadoresRecientes. Fuente: Ian Baquero.

Se utiliza un lazo *for* con un método *distinct* para guardar todos los nombres sin repetir de la variable ubicacion del *array* radiacion en la variable ubicaciones.

Se utiliza un lazo *for* para realizar tantas iteraciones como ubicaciones existan en la variable radiacion, de esta manera se realizará un filtrado de la muestra más reciente por ubicación.

Se asigna valor a las variables i, bandera y reciente para que en cada iteración se inicialicen con un valor respectivo. Se utiliza un lazo *for* para la variable radiación e iterar todos los

objetos dentro de esa variable. A través de un condicional *if*, se compara la variable bandera y la ubicación del marcador con el fin de inicializar los campos “index” y “date”, para tener una variable date valida que se pueda comparar con los demás objetos del arreglo radiación. Este condicional se utilizará una vez por cada ubicación diferente.

Se utiliza un condicional *if* para buscar el objeto más reciente y almacenar el índice del objeto que no lo es para posteriormente ser borrado.

El algoritmo base se puede observar en el código de la Figura 2.36.

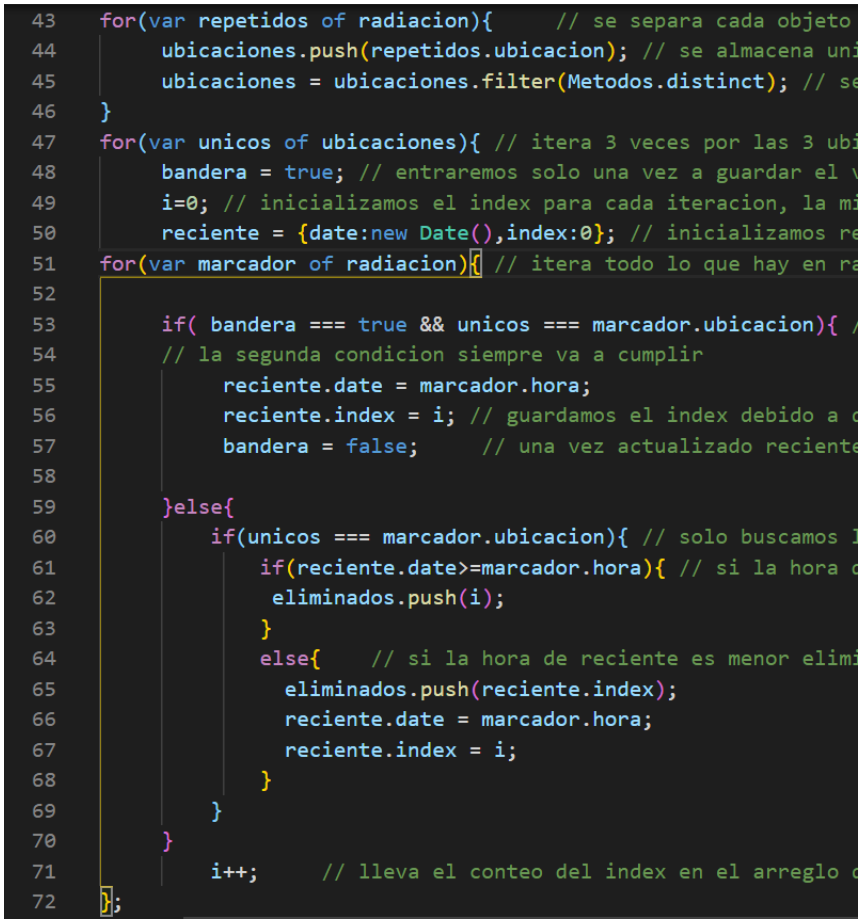


Figura 2.36. Algoritmo base marcadoresRecientes. Fuente: Ian Baquero.

A continuación, se explica las diferencias de los demás algoritmos del archivo métodos respecto al algoritmo base:

- **Método marcadoresMes:** Se realiza el filtrado de los objetos y la comparación en base al mes y las muestras dentro del horario de 10 am a 16 pm. Para posteriormente almacenarlos en una variable del tipo *number* y sacar el promedio de todas las muestras por mes.

- **Método maximoMes:** Se realiza el filtrado de los objetos y la comparación en base al mes y se almacena la variable con el índice UV más alto por mes.
- **Método marcadoresSemanal:** Se realiza el filtrado de los objetos y la comparación en base al día del mes y las muestras dentro del horario de 10 am a 16 pm. Para posteriormente almacenarlos en una variable del tipo *number* y sacar el promedio de todas las muestras por día.
- **Método MaximoSemana:** Se realiza el filtrado de los objetos y la comparación en base al día del mes y se almacena la variable con el índice UV más alto por día.

A continuación, se explica la diferencia de las demás rutas:

La ruta "/recientes" es del tipo post y realiza la consulta a la colección a partir de la fecha especificada por el usuario en la aplicación móvil, alojada en la variable calendar como se observa en el código de la Figura 2.37; se utiliza el método *sort* para ordenar de menor a mayor según la hora los objetos y *find* para filtrar los objetos que tengan la fecha especificada en calendar entre la primera hora del día y finalice la búsqueda en la hora 19pm, ya que dentro de este rango de horas se encuentran las muestras de interés.

```
56  userRoutes.post('/radiacion', (req: Request,res: Response,next)=>{
57      calendar = req.body.calendar;
58      var end = new Date(calendar); // obtenemos el dia, mes y anio del dia de hoy
59      end.setHours(19); // seteamos a 19 la hora debido a que es una hora posterior a las mediciones del
60      Radiacion.find({hora:{$gte:new Date(calendar), $lt: end}}).sort({hora:1}).exec((err,radiacion)=>{
```

Figura 2.37. Métodos de las rutas radiación y recientes. Fuente: Ian Baquero.

Las rutas "/semanal" y "/maxsemanal" realizan la consulta a la colección de la base de datos utilizando el método *sort* para ordenar de menor a mayor los objetos según la fecha y *find* para filtrar los objetos que tengan la fecha entre el día actual y treinta días anteriores como se observa en el código de la Figura 2.38. Este tipo de filtrado se explicará más adelante en la implementación de las muestras en el grafico dinámico.

```
142  userRoutes.get('/maxsemanal',(req: Request, res:Response, next)=>{
143      var hoy = new Date(); // obtenemos el dia, mes y anio del dia de hoy
144      hoy.setHours(0); // seteamos a 0 la hora debido a que nos interesa obtener todas las mu
145      let start= new Date(hoy);
146      let end = new Date(hoy); // no restamos 1 a end debido a que el limite es el dia actual
147      start.setDate(start.getDate()-30); // restamos 30 porque queremos los 30 dias anteriore
148      Radiacion.find({hora: {$gte: start, $lt: end}}).sort({hora:1}).exec((err,radiacion)=>{
```

Figura 2.38. Métodos de las rutas semanal y maxsemanal. Fuente: Ian Baquero.

Las rutas "/mes" y "/maxmes" realizan la consulta a la colección de la base de datos utilizando el método *sort* para ordenar de menor a mayor los objetos según de todos los documentos de la colección, como se observa en el código de la Figura 2.39.

```
80  userRoutes.get('/mes',(req: Request, res:Response, next)=>{
81      Radiacion.find().sort({hora:1}).exec((err,radiacion)=>{
```

Figura 2.39. Métodos de las rutas mes y maxmes. Fuente: Ian Baquero.

2.3.1.3.2. Método calcularCoordenadas

Para implementar el algoritmo de este método se definió el equivalente de un grado en kilómetros:

- Latitud: Un grado de latitud equivale a 111.12 kilómetros. [38]
- Longitud: Un grado de longitud equivale a 111.32 kilómetros. [39]

Un hexágono regular necesita tres distancias básicas para ser dibujado, las cuales serán explicadas posteriormente en la implementación de los hexágonos en el mapa, estas distancias son:

- radio: Valor de la distancia del radio de los hexágonos.
- x: Distancia igual a la mitad de la longitud del radio.
- y: Distancia del centro del hexágono a la mitad de uno de sus lados.

Dado que se define el radio por el administrador, utilizando geometría a través de la distancia entre dos puntos se implementa el algoritmo para encontrar las distancias básicas dado un valor de radio como parámetro de entrada como se observa en el código de la Figura 2.40.

```
3     static calcularCoordenadas(radio: number ) {
4       let puntos: number [] = [];
5       let x = radio / 1.15468;
6       radio = Math.pow(radio, 2);
7       let y = Math.sqrt(radio - Math.pow(x, 2));
8       x = x / 111.32; // 111.32 equivale a un grado de longitud en kilometros
9       y = y / 111.12; // 111.12 equivale a un grado de latitud en kilometros
10      radio = Math.sqrt(radio) / 111.12;
11      puntos = [x, y, radio];
12      console.log('mis coo:', puntos);
13      return puntos;
14    }
```

Figura 2.40. Metodo calcularCoordenadas. Fuente: Ian Baquero.

2.3.1.3.3. Método escogerColor

Dado el parámetro de entrada índice de radiación UV, a través de condicionales se define el color especifico según el rango en el que se encuentre el índice de radiación UV, tal como se observa en el código de la Figura 2.41.

```
static escogerColor(uv: number) { // metodo para
   if (uv <= 2) {
     return '#43FF33';     //color verde
   } else if (uv <= 5 && uv > 2  ) {
     return '#F9F80D';     // color amarillo
    } else if (uv <= 7 && uv  > 5 ) {
        return '#F98204';     // color naranja
    } else if (uv <= 10 && uv > 7  ) {
        return '#FF0000';     // color rojo
    } else {
        return '#BB04F9';     // color violeta
    }
}
```

Figura 2.41. Método escogerColor. Fuente: Ian Baquero.

2.3.1.3.4. Servicio web configuración del radio

Se creó una variable radio del tipo *number*, inicializada cada vez que se encienda el servidor web con el valor de uno, el cual representa por defecto un kilómetro de radio para el tamaño de los hexágonos. A través de una ruta con extensión "/radio" del tipo *post*, se creó un servicio que permite configurar y actualizar el radio de todos los hexágonos utilizando el método calcularCoordenadas, tal como se observa en el código de la Figura 2.42. Esto significa que el cambio realizado en esta variable radio actualizara el tamaño de los hexágonos cada vez que se llame al servicio marcadoresRecientes en todos los dispositivos móviles.

```
8   var radio: number = 1 ;
9   var coordenadas = Metodos.calcularCoordenadas(radio);
10
11  userRoutes.post('/radio', (req: Request,res: Response,next)=>{
12      radio = req.body.dato;
13      coordenadas = Metodos.calcularCoordenadas(radio);
14      try{
15          res.json({ // esta es la respuesta que le voy a mandar al
16              ok:"true",
17              radio
18           });
19      }catch(err){
20          next(err); // genera el error con un response
21      }
22  })
```

Figura 2.42. Servicio web configuración del radio. Fuente: Ian Baquero.

2.3.1.4. Servicios web del modelo log

2.3.1.4.1. Función middleware manejo de errores

Express es una serie de llamadas a funciones *middleware*. Estas funciones *middleware* son funciones que tienen acceso al objeto de solicitud (Request), al objeto de respuesta (Response) y a la función *middleware* en el ciclo solicitud – respuesta de la aplicación,

denotada con una variable denominada *next*. Si la funciona *middleware* no finaliza el ciclo solicitud – respuesta, debe invocar next () para pasar el control a la siguiente función de *middleware*, caso contrario la solicitud quedara colgada. [40]

Para ello, se creó un archivo con el nombre de “errors”, para crear una función *middleware* que capture cualquier error presentado en el servidor web y lo guarde en la colección “logs” de la base de datos mapa. La función se definió como un *middleware* de manejo de errores de la misma forma que otras funciones de *middleware*, con la excepción que las funciones de manejo de errores tienen cuatro argumentos de entrada en lugar de tres: (err, req, res, next). [40]

Como se observa en el código de la Figura 2.43. se define la función *middleware* con el modelo Log para enviar el error capturado a la colección “logs”. Además, se configura la hora del error debido al cambio de horario por el estándar GMT (Greenwich Mean Time) una vez almacenada en la colección.

```
import { Log } from './models/log.model';
var log
var dia = new Date();
dia.setHours(dia.getHours()-5);
function middleware(error:any, req:any, res:any, next:any): void {
    res.json({
        status:'error',
        name: error.name,
        message: error.message,
        date: Date(),
    });

    log = {
        nombre: error.name,
        mensaje: error.message,
        fecha : dia,
    };
    Log.create(log).then(logDB=>{
        console.log('Se guardo con exito el log');
    }).catch(err => {
        next(err);
    })
};


module.exports =  middleware;

```

Figura 2.43. Función middleware para captura de errores. Fuente: Ian Baquero.

Una vez creado el *middleware* para captura de errores, se importa al archivo “index”, al final de todas las rutas y app.use (). Finalmente se agrega la función next() en todas las secciones de código posibles de capturar errores de todas las rutas como se observa en el ejemplo del código de la Figura 2.44.

```
85  userRoutes.get('/mes',(req: Request, res:Response, next)=>{
86      var hoy = new Date(); // obtenemos el dia, mes y anio del dia de hoy
87      hoy.setHours(0); // seteamos a 0 la hora debido a que nos interesa obt
88      Radiacion.find().sort({hora:1}).exec((err,radiacion)=>{
89          if(err){
90              err.message = 'Error en el servidor';
91              next(err);
92          }else{
93              if(Object.keys(radiacion).length !== 0){
94                  radiacion=Metodos.marcadoresMes(radiacion);
95                  res.status(200).send({
96                      radiacion
97                  });
98              }else{
99                  var err = new Error('No hay muestras en la base');
100                 next(err);
101             }
102         }
103     });
104
105 });
```

Figura 2.44. Ejemplo de función next en las rutas del servicio web Fuente: Ian Baquero.

2.3.1.4.2. Servicio mostrar log de errores

Dentro del archivo de rutas para la colección “logs”, se utilizó una ruta del tipo *get* llamada “/errores” para enviar todos los errores almacenados de la colección “logs” a la aplicación móvil, tal como se observa en el código de la Figura 2.45. Utilizando el modelo Log con el método *find* se busca todos los documentos de la colección.

```
6   logRoutes.get('/errores',(req: Request, res:Response, next)=>{
7       Log.find({}).exec((err,log)=>{
8           if(err){
9               err.message = 'Error en el servidor';
10              next(err);
11          }else{
12              if(Object.keys(log).length !== 0){
13                  res.status(200).send({
14                      log
15                  });
16              }else{
17                  res.status(200).send({
18                      log: []
19                  });
20              }
21          }
22      });
23  });
```

Figura 2.45. Servicio web mostrar log de errores. Fuente: Ian Baquero.

2.3.1.4.3. Servicio limpiar log de errores

Similar al servicio de mostrar log de errores, se utilizó una ruta del tipo *get* llamada “/limpiar” para eliminar todos los errores almacenados de la colección “logs” y enviar un archivo vacío a la aplicación móvil, tal como se observa en el código de la Figura 2.46. Utilizando el modelo Log con el método *remove* y el atributo “/$/” se elimina todos los documentos de la colección.

```
logRoutes.get('/limpiar',(req: Request, res:Response, next)=>{
    const todos = {"nombre" : /$/};
    Log.remove(todos).exec((err,log)=>{
        if(err){
            err.message = 'Error en el servidor';
            next(err);
        }else{
            if(log.n !== 0){
                res.status(200).send({
                    log: []
                });
            }else{
                res.status(200).send({
                    log: [],
                    mensaje: 'no existe documentos en la coleccion'
                });
            }
        }
    });
});
```

Figura 2.46. Servicio web limpiar log de errores. Fuente: Ian Baquero.

2.3.2. IMPLEMENTACIÓN DEL FRONT-END

Como se definió en los requerimientos funcionales se decidió implementar cuatro módulos, una interfaz por cada uno. Para la implementación de la aplicación móvil se ha utilizado el *framework* Ionic, el mismo que ha sido detallado en el capítulo uno.

Para empezar a desarrollar en Ionic, es necesario contar con unas instalaciones previas para que nuestra ventana de comandos reconozca los comandos utilizados en la codificación, estos son:

- NodeJS.
- TypeScript.
- AngularCLI.
- Ionic.
- Cordova.

Una vez realizadas las instalaciones necesarias, desde la ventana de comandos (CLI) ejecutamos el comando:

$ionic start myApp tabs

Esto creará la plantilla principal como base de todos los módulos tal y como se observa en la Figura 2.17. Cada módulo tiene su propio *tab* el cual se caracteriza de un archivo html, css, TypeScript (ts) y module.ts para la configuración del Front-End del módulo respectivo.

2.3.3. ANGULAR MATERIAL

Es una librería de componentes prediseñados webs disponibles para angular. Para abarcar todos los componentes de angular material en un solo archivo se creó un módulo personalizado con el comando:

$ng g module material –flat

Dentro del archivo creado material.module.ts se importa y exporta todos los componentes que serán utilizados en los diferentes módulos como se observa en el código de la Figura 2.47.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

// Angular Material
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatCardModule } from '@angular/material/card';
import { MatDialogModule } from '@angular/material/dialog';
import { MatTableModule } from '@angular/material/table';

@NgModule({
  declarations: [],
  imports: [
    CommonModule,
    MatToolbarModule,
    MatCardModule,
    MatDialogModule,
    MatTableModule
  ],
  exports: [
    MatToolbarModule,
    MatCardModule,
    MatDialogModule,
    MatTableModule
  ]
})
export class MaterialModule { }

```

Figura 2.47. Archivo material.module.ts. Fuente: Ian Baquero.

- **Interfaces**

Las interfaces se crearon con la finalidad de tener un modelo de datos para los objetos JSON recibidos del servicio web y un modelo de datos para los elementos de la tabla de medidas de protección, tal como se muestra en el código de la Figura 2.48.

```typescript
export interface RespuestaRadiacion {
  radiacion: Radiacion[];
}

export interface Radiacion {
  _id: string;
  ubicacion: string;
  uv: number;
  hora: Date;
  latitud: number;
  longitud: number;
  coordenadas: number [];
  color: string;
}

export interface Element {
  uv: string;
  riesgo: string;
  exposicion: number;
  description: string;
}

```

Figura 2.48. Archivo interfaces.ts. Fuente: Ian Baquero.

- **Servicio Radiación**

Se generó un servicio de angular con el nombre de radiacion.service.ts, con el objetivo de configurar la librería HttpClient y realizar las peticiones REST a los servicios web.

Se crea una variable constante llamada “URL” para guardar la dirección IP y puerto local del servicio web al que nos queremos comunicar. Debido a que todavía no se configura un servidor en la nube, se conecta localmente al servidor.

Dentro de este servicio se crea una instancia de la librería HttpClient llamada http dentro del constructor y se definen las rutas del servicio web como se observa en el código de la Figura 2.49.

Figura 2.49. Configuración de las peticiones del servicio web. Fuente: Ian Baquero.

De manera opcional se extendió las peticiones *get* y *post* de la interfaz RespuestaRadiacion que se encuentra en el archivo de interfaces, para conocer los campos que tiene el objeto cuando se recupera, esto es unicamente para los servicios web con el rol de usuario.

- **Servicio Event**

Se generó un servicio de angular con el nombre de data.service.ts, con el objetivo de ejecutar métodos de un componente externo, a través de eventos generados desde el componente actual. A través de la librería EventEmitter se crean instancias para enviar eventos entre formularios del mismo nivel, tal como se muestra en el código de la Figura 2.50.

```
import { Injectable, EventEmitter } from '@angular/core';
import { Subscription } from 'rxjs';
@Injectable({
  providedIn: 'root'
})
export class DataService {
  dibujarCalendar = new EventEmitter();
  dibujarHoy = new EventEmitter();
  dibujarSemana = new EventEmitter();
  dibujarMes = new EventEmitter();
  dibujarMaxSemana = new EventEmitter();
  dibujarMaxMes = new EventEmitter();
  subsVar: Subscription;
  Calendar() {
    this.dibujarCalendar.emit();
  }
  Hoy() {
    this.dibujarHoy.emit();
  }
  Semana() {
    this.dibujarSemana.emit();
  }
  Mes() {
    this.dibujarMes.emit();
  }
  MaxSemana() {
    this.dibujarMaxSemana.emit();
  }
  MaxMes() {
    this.dibujarMaxMes.emit();
```

Figura 2.50. Configuración del servicio Event. Fuente: Ian Baquero.

A continuación, se detallará el procedimiento seguido en cada módulo para su desarrollo:

2.3.3.1. Módulo mapa

Para la implementación del mapa se instaló la librería Angular Google Maps (agm) con el comando:

$npm install @agm/core –save

Se importa el módulo de agm al archivo tab1.module.ts como se muestra en la Figura 2.51. El módulo requiere de una *apiKey* la cual puede ser creada desde su página oficial de angular maps. Una vez creada la *apiKey* se añade al módulo agm.

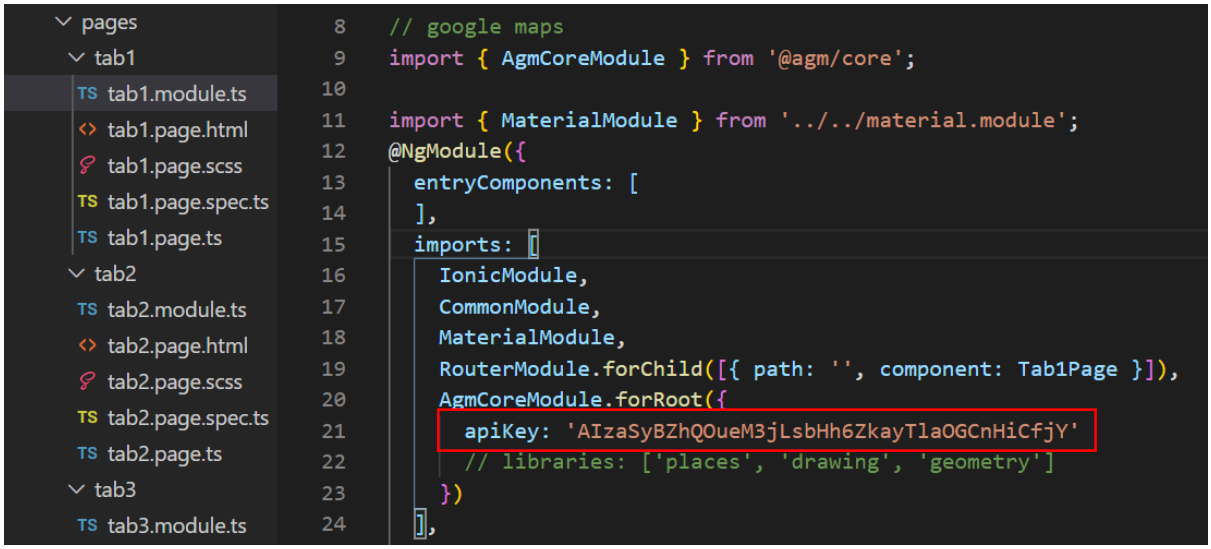


Figura 2.51. Módulo Angular Google Maps (agm). Fuente: Ian Baquero.

En el archivo tab1.page.ts se ingresa la latitud y longitud donde se va a observar el mapa cuando se ejecute desde la aplicación móvil. Para nuestra aplicación se utilizó las coordenadas latitud: -0.211 y longitud: -78.5 que se acerca al sector de estudio especificado en el alcance del Trabajo de Titulación.

En el archivo tab1.page.html se asigna el componente mapa con la etiqueta <agm-map> y dentro de la etiqueta se asigna la latitud y longitud configurada en el archivo tab1.page.ts.

A través del archivo tab1.page.css se configura el estilo del selector del módulo agm-map.

- **Clase Marcador**

Una vez creado el mapa se implementaron los llamados “marks”, que son marcadores propios de la librería agm, los mismos que a través de la clase marcador, se instanciarán objetos del tipo marcador. Cabe recalcar que los marcadores son los elementos dentro del mapa que representarán la información de las características de las muestras tomadas por los sensores UV.

Como se observa en el código de la Figura 2.52. se codifica la clase “Marcador” con sus atributos y constructor, la misma que se especifica en el diagrama de clases.

```
// aqui definimos la estructur que van a tener los marcadores
export class Marcador { // importante poner export ya que caso contrario no se podra exportar la clase
    public latitud: number;
    public longitud: number;
    public hora: Date;
    public uv: number;
    public ubicacion: string;
    public coordenadas: number[];
    public color: string;
    constructor( lat: number, lng: number, ubicacion: string, uv: number, hora: any, coordenadas: number[], color: string ) {
        this.latitud = lat;
        this.longitud = lng;
        this.ubicacion = ubicacion;
        this.uv = uv;
        hora.set('hour', hora.hour() + 5); // se compensa la diferencia de horas provocada por el ajuste de zona horaria que tenemos
        this.hora = hora.format('LLL');
        this.coordenadas = coordenadas;
        this.color = color;
    }
}
```

Figura 2.52. Clase Marcador. Fuente: Ian Baquero.

Para la variable hora que almacena fecha y hora de la muestra, se instaló la librería momentjs, la cual se explicará más adelante. Esta librería permite manipular variables del tipo fechas y tiene formatos pre configurados para ser representados en formato de texto.

- **Clase Polígono**

En el código de la Figura 2.53. se representa la codificación de la clase “Polígono” con tres atributos y un constructor. Debido que la clase marcador es utilizado en diferentes métodos para mostrar en el mapa y en el grafico dinámico se consideró, separar estas propiedades de las subáreas en una clase nueva ya que será utilizada unicamente en la interfaz mapa.

```
1  import { LatLngLiteral, LatLng } from '@agm/core';
2  export class Poligono {
3      public paths: Array<LatLngLiteral> = [];
4      public color: string;
5      public colorFilo: string;
6      constructor( paths: Array<LatLngLiteral>, color: string, colorFilo: string) {
7          this.color = color;
8          this.paths = paths;
9          this.colorFilo = colorFilo;
10     }
11 }
```

Figura 2.53. Clase Polígono. Fuente: Ian Baquero.

- **Implementación de los objetos marcadores y polígonos en el mapa**

En el archivo tab1.page.ts se crea una instancia del servicio RadiacionService, el cual a través del método getRecientes() que hace referencia a la ruta de marcadores recientes para obtener un JSON por la respuesta a la petición, el mismo que está conformado por un conjunto de objetos JSON como se observa en el código de la Figura 2.54.

```
constructor(private radiacionService: RadiacionService) {
}
ngOnInit() {
  this.radiacionService.getRecientes().subscribe(resp => {  //
    this.radiacion.push( ...resp.radiacion);
    this.mapJsonToObject(this.radiacion, this.marcadoresBase);
    this.dibujarPoligono(this.marcadoresBase, this.poligonos);
  });
```

Figura 2.54. Instancia del servicio radiacionService. Fuente: Ian Baquero.

A través del método mapJsonToObject convertimos de un objeto JSON a un objeto de la clase Marcador y guardamos todo el arreglo de los nuevos objetos marcador en un *array* de la clase Marcador como se observa en el código de la Figura 2.55.

```
85  public mapJsonToObject(jsonObject, marcadores) {
86    for (const json of jsonObject ) {
87      const fecha = this.moment(json.hora); // Se transforma a moment debido a que esta es la unica variable que n
88      json.latitud = parseFloat(json.latitud.toFixed(4)); // toFixed es para reducir los decimales a 4
89      json.longitud = parseFloat(json.longitud.toFixed(4)); // toFixed es para reducir los decimales a 4
90      const marcador = new Marcador(json.latitud, json.longitud, json.ubicacion, json.uv, fecha, json.coordenadas,
91      marcadores.push(marcador);
92    }
93  }
```

Figura 2.55. Método mapJsonToObject. Fuente: Ian Baquero.

Debido a que en la variable hora dentro de los objetos JSON no se reconocían como una variable tipo *date*, se procedió a instalar la librería momentjs la que ayudaría a manipular esta variable en el objeto marcador y podrá ser representada con un estilo propio de la librería en el mapa.

Esta librería momentjs se instaló a través del comando:

$npm install moment --save

A través del método dibujarPoligono se dibuja la subárea de cada marcador respectivo. Un hexágono regular se puede dibujar en la aplicación móvil a partir de tres distancias constantes respecto al centro del polígono como se observa en la Figura 2.56, estas distancias son L, L/2 y a.

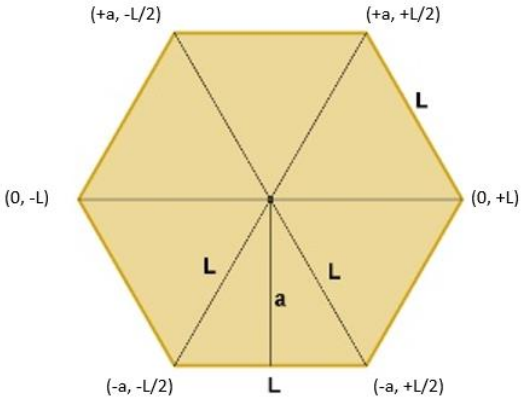


Figura 2.56. Hexágono Regular. Fuente: Ian Baquero.

Estas tres distancias constantes se almacenan en la variable coordenadas del objeto JSON creadas en el servicio web, la posición de las tres distancias en el arreglo son las siguientes:

- Posición 0: L/2, la distancia utilizada para sumar o restar en el valor de longitud.
- Posición 1: a, la distancia utilizada para sumar o restar en el valor de latitud.
- Posición 2: L o el radio del hexágono, la distancia utilizada para ser sumada o restada en el valor centro del polígono.

Cabe mencionar que la posición centro de cada polígono es utilizada de las variables latitud y longitud del objeto marcador, el mismo que es definido por el valor registrado en los sensores UV.

En el código de la Figura 2.57 se observa en el método dibujarPoligono, cómo se crea los nuevos objetos de la clase Polígono a partir de cada objeto Marcador. Se dibuja las seis líneas que conforman el polígono regular en el objeto ruta que se extiende de la clase LatLngLiteral y se asigna al objeto polígono junto a la variable color en los campos, cada polígono se agrega en un arreglo de objetos de la clase Polígono llamada polígonos.

```
101   public dibujarPoligono(marcadores, poligonos) { // dibujar los poligonos como centros de los marcadores
102     for (const marcador of marcadores ) {
103     console.log(marcador.coordenadas);
104     const ruta: Array<LatLngLiteral> = [
105               { lat: marcador.latitud + marcador.coordenadas[2],  lng: marcador.longitud  },
106               { lat: marcador.latitud + marcador.coordenadas[1], lng: marcador.longitud - marcador.coordenadas[0] },
107               { lat: marcador.latitud - marcador.coordenadas[1],  lng: marcador.longitud - marcador.coordenadas[0] },
108               { lat: marcador.latitud  - marcador.coordenadas[2], lng: marcador.longitud },
109               { lat: marcador.latitud - marcador.coordenadas[1], lng: marcador.longitud + marcador.coordenadas[0] },
110               { lat: marcador.latitud + marcador.coordenadas[1],  lng: marcador.longitud + marcador.coordenadas[0] },
111         ];
112     // el array del tipo number definido como coordenadas contienen unicamente 3 valores [x,y,radio]
113     // los mismos valores utilizados para dibujar los hexagonos a partir de una coordenada centro (latitud,longitud
114     const poligono = new Poligono(ruta, marcador.color, marcador.color);
115     poligonos.push(poligono);
116     }
117   }
118 }
```

Figura 2.57. Método dibujarPoligono. Fuente: Ian Baquero.

Para visualizar los objetos marcador y polígono en el Front-End, se los asigna uno a uno a través de la directiva *ngFor desde el módulo NgModule y se asigna cada característica en las propiedades respectivas de las etiquetas “marker” y “polygon” como se observa en el código de la Figura 2.58.

```
<agm-marker *ngFor="let marcador of marcadoresBase; let i = index" [latitude]="marcador.latitud" [longitude]="marcador.longitud">
    <agm-polygon *ngFor="let poligono of poligonos" [paths]="poligono.paths" [fillColor]="poligono.color"  [strokeColor]="poligono.color"
```

Figura 2.58. Asignación de objetos, marcador y polígono a sus etiquetas respectivas. Fuente: Ian Baquero.

A través de la etiqueta <agm-info-window> se crea una ventana informativa que, al ser pulsado encima de un marcador, se asigna todos los parámetros del marcador para posteriormente ser visualizado en el mapa como se observa en la Figura 2.59.

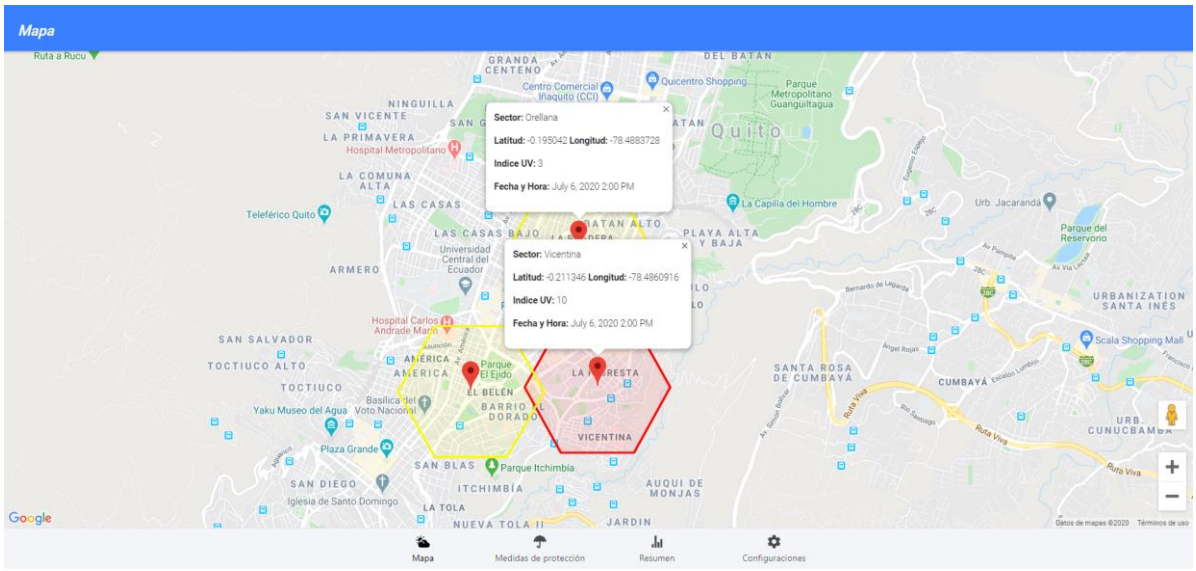


Figura 2.59. Módulo mapa. Fuente: Ian Baquero.

Las subáreas en el módulo mapa a través de un método asíncrono, envían cada treinta segundos una solicitud al servicio web con la ruta “recientes” para actualizar las muestras automáticamente en el mapa en caso de que existan nuevas muestras de los sensores UV.

- **Implementación de notificaciones.**

Para esta interfaz se implementó notificaciones que informan al usuario cuando uno de los marcadores supera el valor de 7 en el índice de radiación UV, como se puede observar en el código de la Figura 2.60, el método notificación es llamado y envía un mensaje con el texto “Riesgo de exposición solar, revisar en el mapa”.

```
41          if ( marcador.uv > 7) {
42          this.notificacion();
43          }
```

```
69  async notificacion() {
70    this.localNotifications.schedule({
71      id: 1,
72      title: 'Alerta',
73      text: 'Riesgo de exposicion solar, revisar en el mapa',
74      data: { secret: 'key' }
75    });
76  }
```

Figura 2.60. Método notificación. Fuente: Ian Baquero.

Este método fue implementado gracias a la librería LocalNotifications de '@ionic-native/local-notifications/ngx'. La cual a través de una instancia de la librería poder acceder a las notificaciones del dispositivo móvil. La representación de la alerta se puede observar en la Figura 2.61.

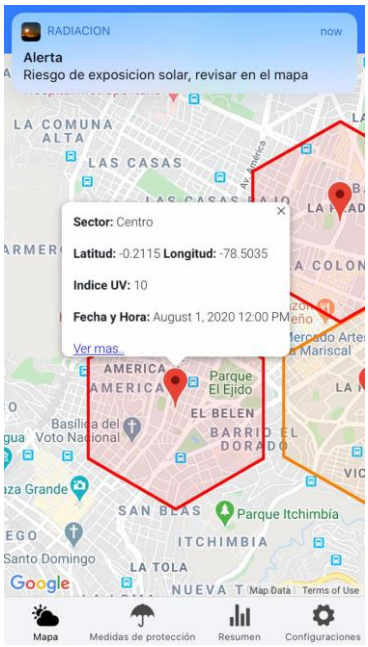


Figura 2.61. Notificación de riesgo de exposición. solar Fuente: Ian Baquero.

2.3.3.2. Módulo medidas de protección

Dentro de este módulo se importó el archivo material.module.ts para utilizar un componente *table* como contenedor de la tabla donde se mostrara las medidas de protección.

En el archivo tab2.page.ts se crea un objeto llamado ELEMENT_DATA que extiende de la interfaz *Element* que se encuentra en el archivo de interfaces, con la finalidad de almacenar el texto respectivo para cada sección en la tabla como se observa en el código de la Figura 2.62.

```
const ELEMENT_DATA: Element[] = [
  {
    uv: '0 a 2',
    riesgo: 'Bajo',
    exposicion: 60,
    description: 'Puede permanecer en el exterior sin riesgo.Gafas de sol en dias brillantes'
  }, {
    uv: '3 a 5',
    riesgo: 'Moderado',
    exposicion: 45,
    description: `Utilice gorra, crema con filtro, gafas y permanezca en areas sombrias`
  }, {
    uv: '6 a 7',
    riesgo: 'Alto',
    exposicion: 30,
    description: `Utilice gorra, crema con filtro, gafas y sombrero. Cuidado con bebes`
  }, {
    uv: '8 a 10',
    riesgo: 'Muy Alto',
    exposicion: 25,
    description: `Utilice gorra, crema con filtro, gafas y sombrero. Procure no exponerse al sol`
  }, {
    uv: '11+',
    riesgo: 'Extremo',
    exposicion: 10,
    description: `Evite la exposicion al sol`
  },
];
```

Figura 2.62. Contenido de la tabla medidas de protección. Fuente: Ian Baquero.

A través de las variables displayedColumns y dataSource propias del componente tabla tal como se observa en el código de la Figura 2.63. se asignará y desplegará la información en la tabla a través del archivo tab2.page.html. Posteriormente se configura los estilos de la tabla para su presentación final.

```
export class Tab2Page {
  displayedColumns: string[] = ['uv', 'riesgo', 'exposicion', 'description'];
  dataSource = ELEMENT_DATA;
}
```

Figura 2.63. Variables propias de la tabla angular material. Fuente: Ian Baquero.

Figura 2.64. Módulo medidas de protección. Fuente: Ian Baquero.

2.3.3.3. Módulo resumen

Dentro de este módulo se creó dos nuevos componentes, uno para manipular el gráfico dinámico llamado “Linea”, otro para manipular la elección del servicio web a mostrar representado dentro del gráfico dinámico llamado “Navbar” y de un componente calendar que permite mostrar el resumen del día seleccionado por el usuario a través de un calendario.

Se instaló la librería ng2-charts para utilizar un gráfico dinámico utilizando el comando:

$npm install chart.js –save

En el archivo tab3.page.ts se crea una instancia del servicio RadiacionService, el cual a través de sus métodos que hace referencia a las rutas en el servicio web, se obtiene el JSON respectivo por la respuesta a la petición. De manera similar se utiliza el método mapJsonToLinea del módulo mapa para convertir el objeto JSON en un objeto TypeScript. Posteriormente se pasa los objetos de las diferentes peticiones web del componente tab3 a la componente línea como se observa en el código de la Figura 2.65 para ser visualizado en el grafico dinámico.

```
<div class="chart-container ">
    <app-navbar></app-navbar>
    <app-linea [datos]="dataLineas" [semanal]="semanal" [mensual]="mensual" [maxsemanal]="maxsemanal" [maxmensual]="maxmensual"></app-linea
</div>
```

Figura 2.65. Paso de información entre formularios. Fuente: Ian Baquero.

- **Componente Línea**

En el archivo línea.component.ts se guarda los objetos TypeScript de las peticiones web realizadas en el componente tab3, a través del decorador @Input, el cual es importado de angular. Este decorador permite recibir datos desde afuera de la componente, en este caso del componente tab3 tal como se observa en el código de la Figura 2.66.

```
@Input() datos; // contiene el array de JSON
@Input() hoy; // contiene el array de JSON d
@Input() semanal; // contiene el array de JS
@Input() mensual; // contiene el array de JS
@Input() maxsemanal; // contiene el array de
@Input() maxmensual; // contiene el array de
```

Figura 2.66. Decorador Input. Fuente: Ian Baquero.

Dentro del archivo línea.component.ts se configura las propiedades del gráfico dinámico como:

- **lineChartData**: Variable utilizada para almacenar los datos de las muestras que serán mostradas.
- **lineChartLabels**: Variable utilizada para almacenar los valores del título del eje horizontal.
- **lineChartOptions**: Variable que almacena el título de los ejes horizontal y vertical.
- **lineChartColors**: Variable utilizada para editar el color de las gráficas.
- **lineChartLegend**: Variable utilizada para visualizar la leyenda de los datos.
- **lineChartType**: Variable utilizada para seleccionar la forma del grafico dinámico.

En el archivo línea.component.ts, utilizando métodos por cada servicio web se garantiza que cada dato se ingrese correctamente en su posición y sensor respectivo.

Se detallará el método actualizarData y se utilizará como método base para explicar los demás.

- **Método Base**

El método base a utilizar tiene el nombre de actualizarData y es el responsable de distribuir las muestras del resumen del día especificado en el componente calendario.

Se inicializa los valores del eje horizontal en la variable lineChartLabels. Se utiliza un método *distinct* llamado sinRepetidos para filtrar todas las ubicaciones diferentes de los objetos del *array* "datos", que es la información de la petición web que contiene todas las muestras del día. Se utiliza un lazo *for* por cada ubicación diferente y se asigna cada dato de la radiación UV al sensor respectivo. Utilizando un condicional *switch* se asigna cada valor de la radiación UV en su hora respectiva, de esta manera aseguramos que los datos de la radiación UV se encuentren en su lugar, tal como se observa en el código de la Figura 2.67. Debido a que las muestras se toman cada media hora, se implementó un condicional *if* antes del condicional *switch* para tomar la primera muestra entre la que tiene la hora con cero minutos y treinta minutos, por lo tanto, en este caso se tomara la muestra con la hora que marque cero minutos.

```
118   for (const ubicacion of ubicaciones) {  // todo estos lazos sirven
119     for (const marcador of this.datos) { // se protege a los valore
120       this.lineChartData[i].label = ubicacion; // escoge una ubicaci
121       if (marcador.ubicacion === ubicacion) {  // condicional para p
122         x = marcador.hora.hour() + 5; // si reseteamos directamente
123         if (x !== temp) { // con esto garantizamos q no tome la hora
124           switch (x) {  // utilizamos este switch para ordenar cada
125             case 7:
126                 this.lineChartData[i].data[0] = marcador.uv;
127                 break;
128             case 8:
129                 this.lineChartData[i].data[1] = marcador.uv;
130                 break;
131             case 9:
132                 this.lineChartData[i].data[2] = marcador.uv;
133                 break;
134             case 10:
135                 this.lineChartData[i].data[3] = marcador.uv;
136                 break;
137             case 11:
138                 this.lineChartData[i].data[4] = marcador.uv;
139                 break;
140             case 12:
141                 this.lineChartData[i].data[5] = marcador.uv;
142                 break;
143             case 13:
144                 this.lineChartData[i].data[6] = marcador.uv;
145                 break;
146             case 14:
147                 this.lineChartData[i].data[7] = marcador.uv;
```

Figura 2.67. Método base actualizarData. Fuente: Ian Baquero.

Los demás métodos de la componente línea para mostrar la información son similares con las siguientes excepciones:

- **Método actualizarMes y maxMes:** Se agrega los meses del mes dentro de la variable lineChartLabels y dentro del condicional *switch* se filtra cada muestra en base al mes y no a la hora como en el método base.
- **Método actualizarSemana y maxSemana:** Se creó un algoritmo para agregar los treinta días anteriores a partir del valor del día anterior a la fecha actual. Además, dentro del condicional *switch* se filtra cada muestra en base al día y no a la hora como en el método base.

Cabe mencionar que el método actualizarHoy realiza el mismo proceso que el método base con la diferencia de que el JSON recibido es unicamente la fecha actual.

- **Componente Navbar**

Este componente fue creado con la necesidad de tener una barra principal en el *header* de la interfaz para visualizar el nombre del módulo y que el usuario tenga la posibilidad de seleccionar el servicio web que desee.

Debido a que el componente Navbar y Linea son componentes del mismo nivel o conocido como componentes hermanos. Se utiliza el servicio Event para ejecutar los métodos del componente Linea a través de un evento “Click” generado desde el componente Navbar. Como se observa en el código de la Figura 2.68. se configuran los métodos de los diferentes eventos “Click” de cada opción del Navbar.

```
import { Component, OnInit, HostListener, Output, EventEmitter, Input } from '@angular/core';
import { DataService } from '../../services/data.service';


@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.scss'],
})
export class NavbarComponent  {
  constructor( private dataService: DataService) {}
  clickHoy() {
    this.dataService.Hoy();
  }
  clickSemana() {
    this.dataService.Semana();
  }
  clickMes() {
    console.log('Entro al click mes');
    this.dataService.Mes();
  }
  clickMaxSemana() {
    this.dataService.MaxSemana();
  }
  clickMaxMes() {
    this.dataService.MaxMes();
  }
}
```

Figura 2.68. Configuración de los métodos Click. Fuente: Ian Baquero.

Cabe mencionar que se creó una instancia del servicio Event en el archivo línea.component.ts para ejecutar los métodos de línea.component.ts, como se observa en el código de la Figura 2.69.

```
603     ngOnInit() {
604     this.dataService.dibujarCalendar.subscribe(() => {
605       this.actualizarRadiacion();
606     });
607     this.dataService.dibujarHoy.subscribe(() => {
608       this.actualizarHoy();
609     });
610     this.dataService.dibujarSemana.subscribe(() => {
611       this.actualizarSemana();
612     });
613     this.dataService.dibujarMes.subscribe(() => {
614       this.actualizarMes();
615     });
616     this.dataService.dibujarMaxSemana.subscribe(() => {
617       this.MaxSemana();
618     });
619     this.dataService.dibujarMaxMes.subscribe(() => {
620       this.MaxMes();
621     });
622   }
623 }
```

Figura 2.69. Métodos de la instancia dataService. Fuente: Ian Baquero.

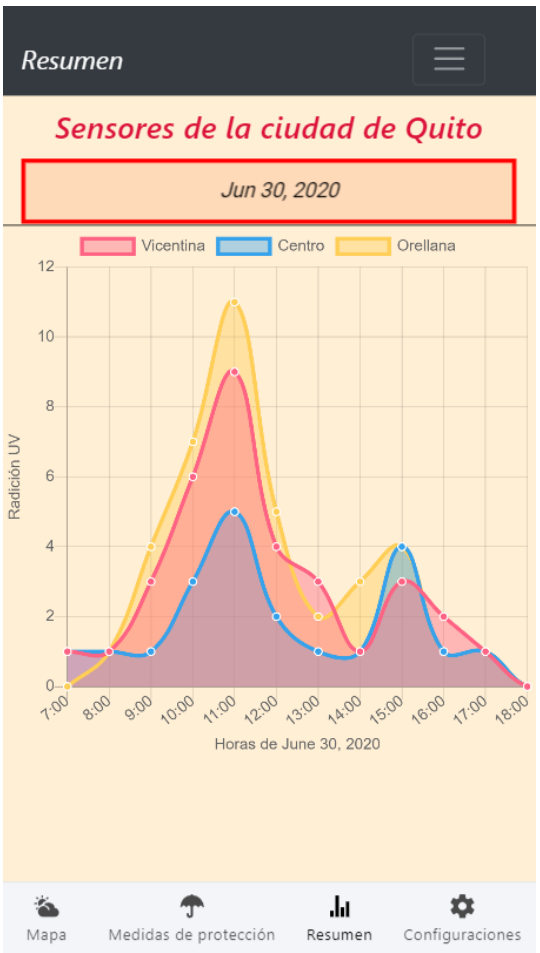


Figura 2.70. Módulo Resumen. Fuente: Ian Baquero.

2.3.3.4. Módulo administrador

Se creó una página con el nombre de tab4, en el cual, se tendrá un *login* con nombre de usuario y contraseña, para que el administrador pueda ingresar a las configuraciones del sistema.

Las configuraciones que este módulo de administrador tiene son: una sección para actualizar el radio de los hexágonos del módulo mapa, otra sección para descargar un archivo del resumen mensual y semanal del índice de radiación UV en el dispositivo móvil y una tabla con los errores capturados por fallas en la generación de los servicios web, con una opción para limpiar los errores guardados en el servidor.

- **Login Administrador**

En esta sección se implementó un *login* que valida su acceso mediante alarmas generadas por el componente AlertController de Ionic. Este componente genera tres alarmas: en caso de que cualquiera de los campos *username* o *password* no estén llenos, se genera una alarma con el mensaje 'Por favor llenar todos los campos'; en caso de un usuario incorrecto por el campo *username* o *password*, se generará una alarma con el mensaje 'Usuario Incorrecto' y cuando el ingreso sea exitoso una alarma con el mensaje 'Logeo Exitoso' se mostrará. Cabe mencionar que solo existe un nombre de usuario y una contraseña valida que se compara dentro de la aplicación móvil. En la Figura 2.71 se observa la interfaz de *login* del administrador.

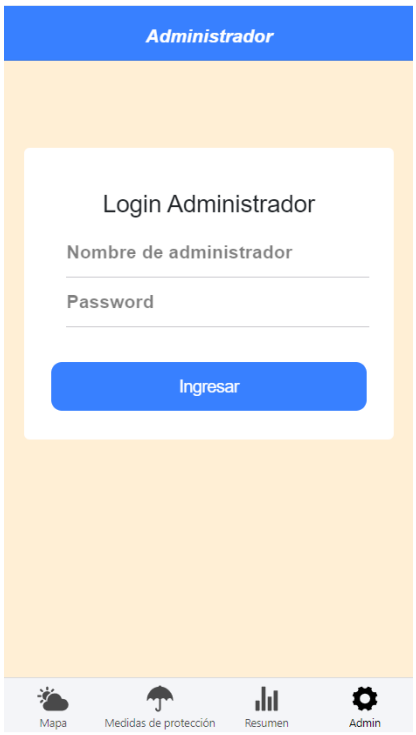


Figura 2.71. Login Administrador. Fuente: Ian Baquero.

- **Configuraciones Administrador**

Una vez que el administrador logre ingresar, se oculta el formulario de *login* con un atributo *hidden* y se habilita todos los componentes de configuración del administrador de la misma manera dentro del archivo tab4.page.html.

Esta interfaz de configuraciones cuenta con tres secciones, como se puede observar en la Figura 2.72:

- Un input para ingresar el nuevo valor en kilómetros del radio de los hexágonos y un botón para actualizar el cambio. Cabe mencionar que se configuro una alarma para que el radio sea mayor a cero y menor a cinco kilómetros. El servicio se solicita de la ruta de tipo post "/api/radio" del servicio web, definido en los servicios de radiación.
- Un calendario para ejecutar la exportación del archivo csv dentro del dispositivo móvil. El archivo a descargar contiene el resumen del día seleccionado en el calendario.
- Una lista con todos los errores capturados por los servicios web y un botón (limpiar) para eliminar el arreglo de todos los errores dentro de la colección "logs". El servicio se solicita de la ruta de tipo *get* "/api/errores" para obtener el JSON de los errores y "/api/limpiar" para vaciar la colección "logs" que almacena los errores del servicio web. Estos servicios se encuentran definidos en los servicios de radiación.

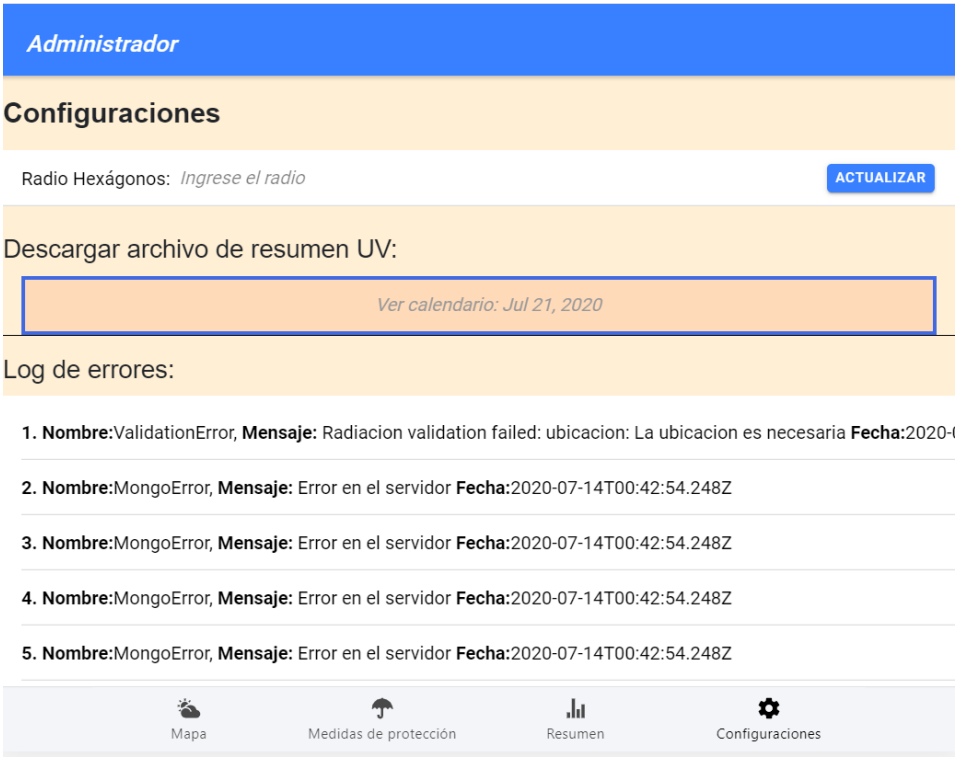


Figura 2.72. Configuraciones Administrador Fuente: Ian Baquero.

- **Exportación del archivo csv**

Se utilizó las siguientes librerías para el desarrollo de esta sección:

- ***ngx-papaparse@3:*** Para convertir un arreglo de objetos en un documento csv.
- ***@ionic-native/file*** y ***ionic Cordova plugin add Cordova-plugin-file:*** Para la creación de un archivo nativo en el dispositivo.
- ***@ionic-native/social-sharing*** y ***ionic Cordova plugin add Cordova-plugin-x-socialsharing:*** Para exportar el archivo csv a las diferentes redes sociales o localmente en el dispositivo.

El código del método a continuación es utilizado de la referencia [41]

Dentro del archivo tab4.page.ts, se crea una instancia de todas las librerías: Papa (papa), File (file), SocialSharing (socialSharing) y una librería propia de Ionic llamada Platform (plt).

Se instancia un objeto csv que extiende de la librería papaparse para utilizar el método *unparse*, el cual escribe cadenas de texto delimitadas dada una matriz de objetos. Dentro de este objeto csv asignamos la data de la matriz con el objeto JavaScript del resumen del índice UV mensual o semanal. A través de un lazo *if*, se analiza si la plataforma en la que se ejecuta la exportación es Cordova.

En caso de cumplir la condición, a través del *plugin file* el cual implementa una API para archivos con acceso de lectura o escritura que residen en el dispositivo, se crea el nuevo archivo 'data.csv'. Utilizando el *plugin* socialSharing, se dispone de compartir el archivo data.csv en las redes sociales o dentro de los archivos propios del dispositivo.

En caso de no cumplir la condición, se crea una implementación simple del navegador que crea un nuevo elemento y lo descarga, como se observa en el código de la Figura 2.73. [41]

```
98   exportRadiacion() {
99     const csv = this.papa.unparse({
100      data: this.dataLineas
101    });
102    if (this.plt.is('cordova')) {
103      this.file.writeFile(this.file.dataDirectory, 'dataRadiacion.csv', csv, {replace: true}).then(res => {
104        this.socialSharing.share(null, null, res.nativeURL, null);
105      });
106    } else {
107        const blob = new Blob([csv]);
108        const a = window.document.createElement('a');
109        a.href = window.URL.createObjectURL(blob);
110        a.download = 'radiacion.csv';
111        document.body.appendChild(a);
112        a.click();
113        document.body.removeChild(a);
114    }
115  }
```

Figura 2.73. Método exportar archivo csv Fuente: Ian Baquero.

2.3.4. INSTALACIÓN DE LA BASE DE DATOS EN LA NUBE

Atlas es la base de datos como servicio IaaS que permite implementar, utilizar y escalar una base de datos MongoDB. Atlas permite crear cuentas gratuitas a través del correo electrónico. Atlas habilita la opción de *hostear* la base de datos en AWS, GoogleCloudPlatform y Azure. Dentro de la plataforma de administración de Atlas primero se crea un usuario con privilegios de escritura y lectura para posteriormente generar un URL para el acceso a la base de datos. Se habilita la opción para que cualquier dirección IP se pueda conectar a nuestra base de datos. Una vez realizada esas actividades en el *cluster* de la plataforma de administrador en la opción de conectar, se selecciona conectar tu aplicación. Dentro de esta opción se selecciona conexión corta por *string* la misma que genera un *string* que contiene el *password* del usuario creado y el nombre de la base de datos dentro del *cluster* a la que está apuntando. Este *string* apunta a una base de datos por defecto llamada test como se observa a continuación.

- **String de conexión:**

mongodb+srv://lan:<password>@cluster0-ru9rg.mongodb.net/test?retryWrites=true&w=majority

A través de este *string* de conexión se puede acceder a la base de datos especificada desde una instancia mongoose creada en el servicio web.

2.3.5. INSTALACIÓN DEL SERVIDOR WEB EN LA NUBE

El servicio de interés para el Trabajo de Titulación es un EC2 (Elastic Compute Cloud), este servicio permite tener capacidad de cómputo en la nube y lanzar una instancia o máquina virtual aprovisionándola de capacidad de procesamiento, memoria y almacenamiento. AWS permite crear cuentas gratuitas de 750 horas de uso para máquinas virtuales de Linux, Windows, entre otras. Para este Trabajo de Titulación se escogió la imagen de Ubuntu Server 16.04 LTS. El tipo de instancia seleccionada para la tarifa gratuita contiene un procesador virtual, una giga de memoria y almacenamiento de bloque. La instancia de Ubuntu seleccionada tiene ocho gigas de aprovisionamiento por defecto y un volumen SSD de propósito general. Se configura un *security group* que es similar a un firewall que controla el tráfico de entrada y salida a nuestra instancia. Se genera un nuevo *key pair* y se lanza la instancia.

Dentro de la plataforma de administración se muestra todos los detalles de las instancias creadas. Una de las características importantes de esta instancia es su dirección IP. A través de una herramienta para crear clientes ssh como “Putty” se ingresa el nombre de usuario en este caso Ubuntu seguido del DNS de la instancia como se observa en la Figura 2.74.

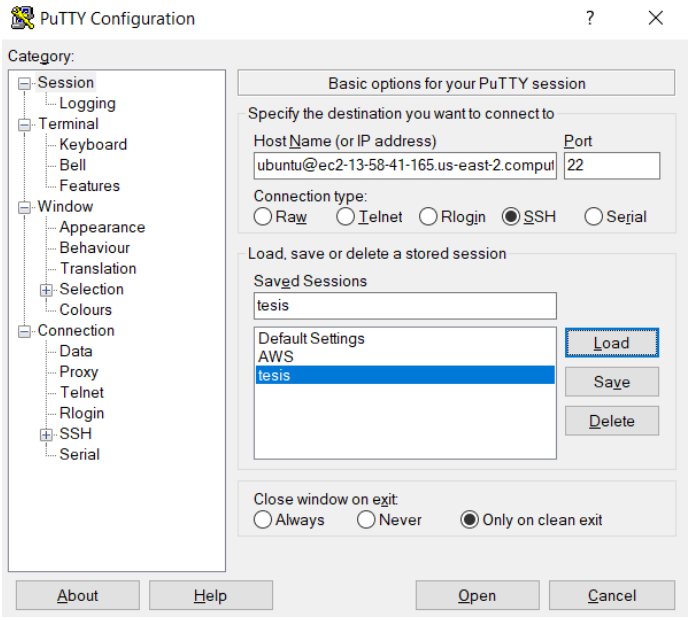


Figura 2.74. Dirección DNS de la instancia AWS Fuente: Ian Baquero.

En la herramienta Putty en la opción de conexión se selecciona la opción SSH (Auth), en la cual se carga la *key pair* generada previamente y se accede al Shell de la instancia de Ubuntu como se observa en la Figura 2.75.

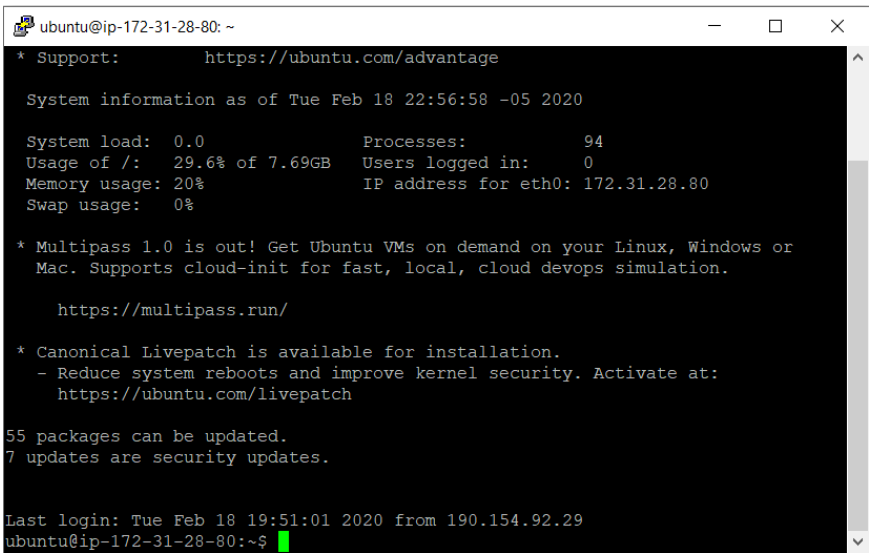


Figura 2.75. Shell de la instancia Ubuntu en AWS Fuente: Ian Baquero.

Dentro de la instancia se descarga los paquetes para NodeJS, MongoDB y nodemon para levantar el servicio web.

Se utilizó Github para almacenar una copia del servicio web codificado localmente en Visual Studio. Dentro del Shell de la instancia en un directorio a través del comando:

$git clone https://github.com/IanAriel96/mapa-server

Se descarga todo el código del servicio web alojado en el repositorio Github en la instancia de Ubuntu. Una vez alojado el código, se levanta el servicio web con el comando “nodemon dist/”.

2.3.6. PUBLICACIÓN DE LA APP EN LAS TIENDAS VIRTUALES

Se debe configurar las plataformas de destino donde se va a compilar la aplicación móvil. A través del comando a continuación, se crea las plataformas como un subdirectorio dentro del proyecto Ionic:

$ ionic Cordova platform add ios/android

Se configura la versión y el id único de la aplicación en el archivo config.xml que se encuentra dentro del proyecto de Ionic, como se observa en la Figura 2.76. Cabe recalcar que se debe cambiar la versión a una superior cada que se requiera hacer una actualización y desplegar una nueva versión en las tiendas virtuales.

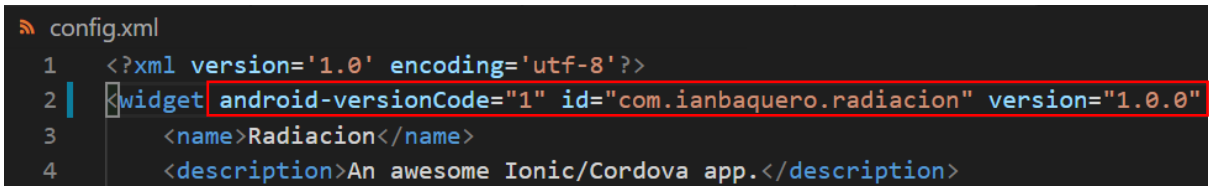


Figura 2.76. Configuración de versión e id único del archivo config.xml. Fuente: Ian Baquero.

Se construye la aplicación con el comando:

$ ionic Cordova build ios/android

Esto genera un código especifico de plataforma dentro del subdirectorio del proyecto. El comando ***Cordova build*** es una abreviatura de preparar y compilar el proyecto.

2.3.6.1. Android Play Store

Algunas versiones del sistema operativo Android tienen inhabilitado el tráfico en texto claro para dominios específicos; por lo tanto, es necesario habilitar el tráfico y añadir la dirección IP y el puerto para el acceso a nuestro servidor web en el archivo network_security_config,xml del subdirectorio *resources* Android, como se observa en la Figura 2.77.

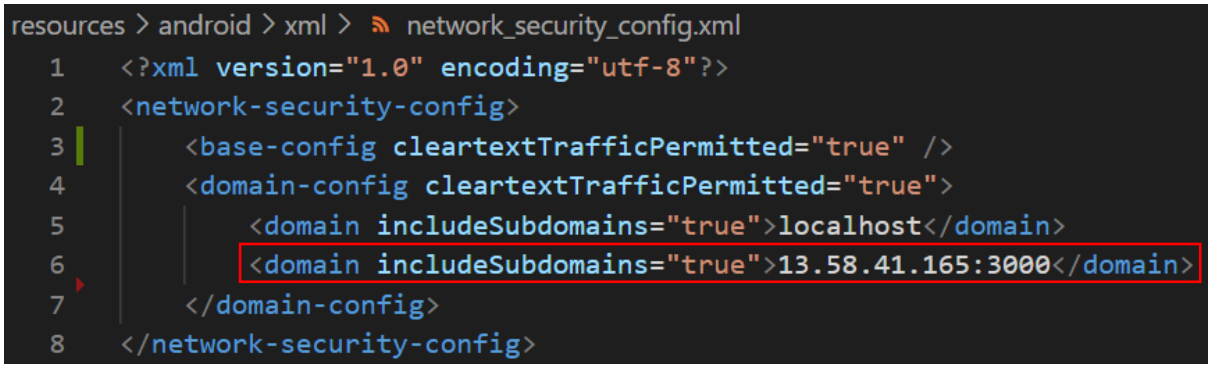


Figura 2.77. Configuración del archivo network_security_config,xml. Fuente: Ian Baquero.

Android requiere de APKs firmados de manera digital para desplegarlos en la Google Play Store. A través de la herramienta Android Studio se puede firmar el paquete de aplicación antes de subirlo a Play Console, y la firma de apps de Google Play se encargará del resto.

Google Play Console es la plataforma para desplegar las aplicaciones y juegos en la tienda virtual de Android. Esta plataforma exige una serie de pasos y el pago de una cuota de registro para tener una cuenta de desarrollador.

Una vez obtenida la cuenta de desarrollador, se debe llenar varios requisitos caso contrario Google no permitirá el despliegue de la aplicación. Dentro de estos requisitos se encuentran:

- Versiones de la app: Es un formulario donde se permite crear versiones Alfa, Beta y de producción de la aplicación. En esta opción se carga el APK firmado.
- Ficha de Play Store: Es un formulario para llenar la descripción de la aplicación y publicar las capturas de pantalla en las fichas de Google Play Store.
- Clasificación del contenido: Es un formulario que permite comprender el nivel de madurez que debe tener el usuario para usar una app.
- Precios y distribución: Es un formulario para especificar si la aplicación a descargar es de gratuita o de paga.

Completado todos los requisitos Google Play Console se tiene permiso para publicar la aplicación como se muestra en la Figura 2.78.

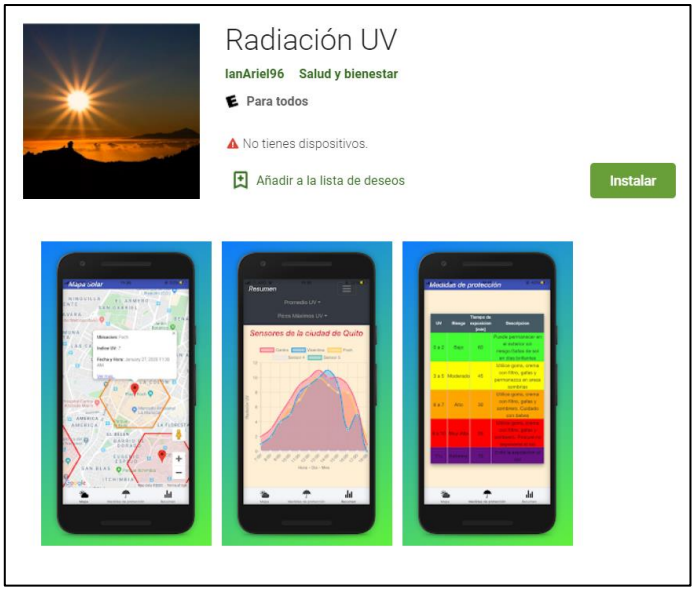


Figura 2.78. Aplicación Móvil publicada en la tienda Google Play Fuente: Ian Baquero.

2.3.6.2. IOS App Store

App Store Connect es la plataforma web propia de Apple para desplegar las aplicaciones y juegos en la tienda virtual de iOS. Esta plataforma exige una serie de pasos y el pago de una cuota de registro para tener una cuenta de desarrollador.

Se debe configurar el *team* de desarrollo con la cuenta de desarrollador en el archivo con extensión xcodeproj del subdirectorio *platforms* ios. Los archivos con extensión xcodeproj pueden ser configurados unicamente con la aplicación Xcode propia de las computadoras Mac.

Dentro de la plataforma web de Apple Developer propia de Apple, en la opción de registrar una app id o *bundle* se configura la información de nuestra aplicación y se configura las características que tiene nuestra aplicación. En el *bundle* id se agrega el id de la aplicación Ionic configurada en el archivo config.xml.

Dentro de la plataforma web de App Store Connect se ingresa a la opción de my apps y se crea una nueva app, utilizando el id de la aplicación configurada en la plataforma de Apple Developer para preparar el espacio donde se desplegará nuestra aplicación.

Se crea el archive de la aplicación unicamente en Xcode, para ello se configura el archive de la aplicación en modo *release* dentro de la opción Product → Choose Scheme y para generarlo se ingresa a la opción de Product → Archive. Finalmente se distribuye la aplicación

siguiendo los pasos dentro de archive. Una vez finalizado el proceso se puede observar nuestra aplicación en el App Store Connect.

Dentro de la plataforma web App Store Connect se configura detalles, capturas de pantalla y características de nuestra aplicación para su preparación y despliegue en la tienda virtual.

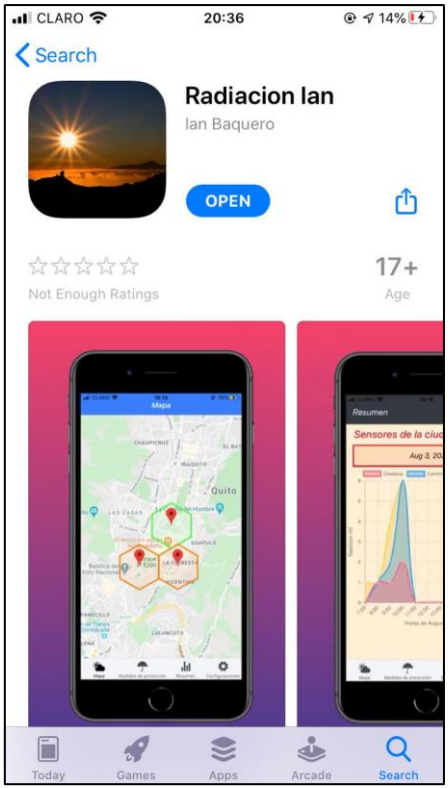


Figura 2.79. Aplicación Móvil publicada en la tienda App Store Fuente: Ian Baquero.

2.3.7. COSTOS

A continuación, se detalla el costo invertido para este Trabajo de Titulación, el cual consta de los gastos realizados tanto en licencias de desarrollador, mano de obra y materiales utilizados para la implementación del prototipo UV como se puede observar en la Tabla 2.7.

Los requerimientos solicitados a la empresa local para el desarrollo del prototipo UV fueron los siguientes:

- Lectura en tiempo real de la radiación UV, latitud, longitud, fecha y hora.
- Envío de datos mediante un módulo GSM en formato JSON.
- Diseño de un prototipo UV resistente a los cambios climáticos de la región como lluvia.

Dentro de las pruebas de funcionamiento correspondiente al capítulo tres, se comprobará el funcionamiento de estos requerimientos de lectura y envió de datos solicitados a la empresa local para realizar las correcciones necesarias en caso de ser necesarias.

Tabla 2.7. Costo de inversión. Fuente: Ian Baquero.

TIPO	CANTIDAD	DESCRIPCIÓN	VALOR UNITARIO	VALOR TOTAL
Licencias Desarrollador	1	Licencia de desarrollador Google Play	100	100
	1	Licencia de desarrollador Apple	25	25
Mano de Obra	3	Diseño y montaje del prototipo	20	60
	1	Programación del prototipo	125	125
Materiales	3	Arduino MEGA 2560	16	48
	3	Shield módulo SIM 900 GSM	28,8	86,4
	3	Sensor UVM30 A	13	39
	3	Caja Prototipo UV	15	45
	3	Módulo reloj para Arduino	3,5	10,5
	3	Display LCD 16X2	4	12
	3	Módulo GPS	12	36
	3	Potenciómetro	0,25	0,75
	3	Cables de conexión para Arduino	2,5	7,5
	3	Caja plástica sensor UVM30 A	1,5	4,5
	12	Tornillos de sujeción	1,1	13,2
	3	Cables de alimentación	1	3
	3	Protoboard mini	1	3
	3	Fuente 12V 5A	6	18
	24	Tapón pequeño para gabinete	1,2	28,8
	6	Tapón grande para gabinete	2	12
			TOTAL	677,65

2.3.8. TABLERO KANBAN ETAPA DE IMPLEMENTACIÓN

Esta etapa consistió en la implementación de todo el sistema. Se publicó una primera versión de la aplicación móvil en las diferentes tiendas virtuales, por lo que una vez analizado los primeros cambios a realizar, se procede a continuar con el capítulo de “RESULTADOS Y DISCUSION” para realizar los cambios pertinentes. Dentro del desarrollo de esta etapa no se presentó tareas adicionales o la adquisición de nuevas tarjetas Kanban.

Todas las tareas han sido completadas de manera exitosa, por lo tanto, las tarjetas Kanban pueden pasar al estado “LISTO” en el tablero como se observa en la Figura 2.80.

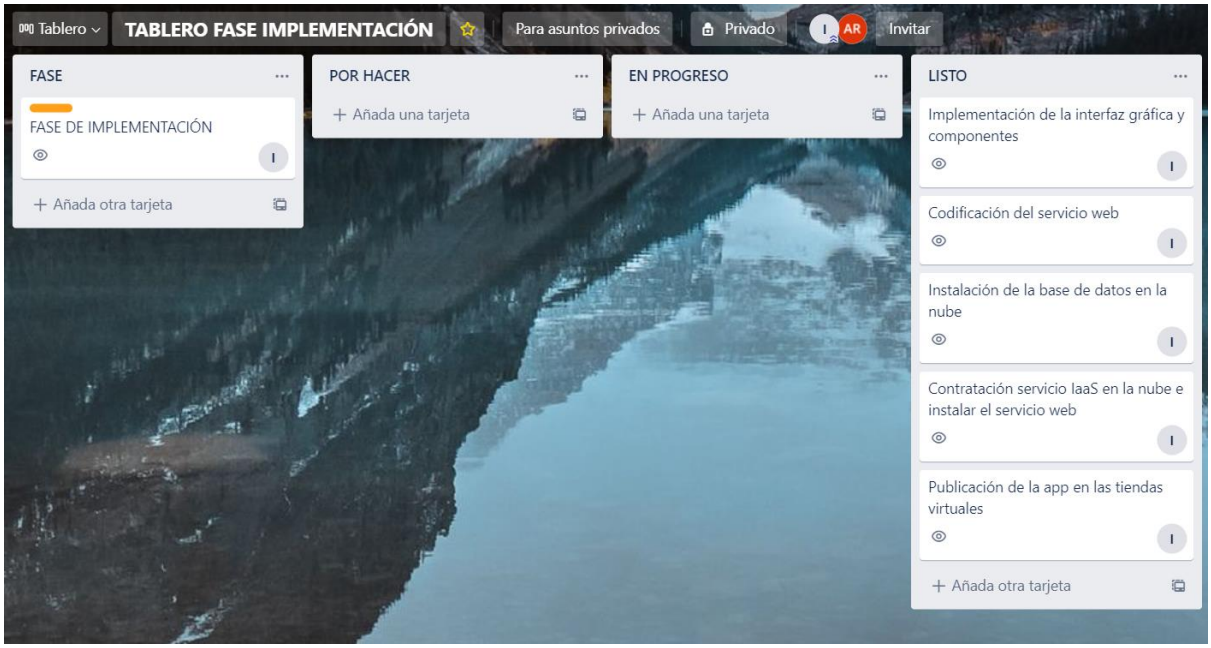


Figura 2.80. Tablero Kanban etapa de implementación estado LISTO. Fuente: Ian Baquero.

3. RESULTADOS Y DISCUSIÓN

En este capítulo se debatirá y validará lo implementado en el capítulo anterior. Se mostrarán los resultados de las pruebas de funcionamiento del sistema en conjunto. Además, se analizará e implementará las correcciones necesarias del sistema, tanto del Back-End como Front-End, todo en función de los errores presentados en las pruebas de funcionamiento. Las correcciones de la aplicación móvil o Front-End se las solucionara subiendo una nueva versión de la aplicación móvil a las tiendas virtuales. Las correcciones en el servidor o Back-End se las solucionara modificando el código del servicio web. Finalmente, se realizará un test para cuantificar la calidad de experiencia del usuario con el test MOS.

3.1. PRUEBAS DE FUNCIONAMIENTO

El funcionamiento del sistema consta de las siguientes etapas:

- **Etapa de adquisición de datos:** Consiste en obtener el índice de radiación y todos los parámetros definidos en la fase de análisis para cada sensor UV.
- **Etapa de envío de muestra:** Enviar los datos de las muestras UV al servidor web, para posteriormente ser almacenados en la base de datos de la nube.
- **Etapa de presentación de datos:** La aplicación móvil solicita los diferentes servicios al servidor web, el mismo que hace consultas a la base de datos, recibe las muestras, las modifica según las configuraciones del servicio, y finalmente envía los objetos en formato JSON a la aplicación móvil para ser representados en sus diferentes módulos.

3.1.1. ETAPA DE ADQUISICIÓN DE DATOS

En esta etapa se verifica los parámetros de la muestra por medio de un *display*, el cual se encuentra localizado dentro de una caja que protege el prototipo UV como se observa en la Figura 3.1. La lectura que se observa consiste de los parámetros latitud, longitud, índice UV, fecha y hora.

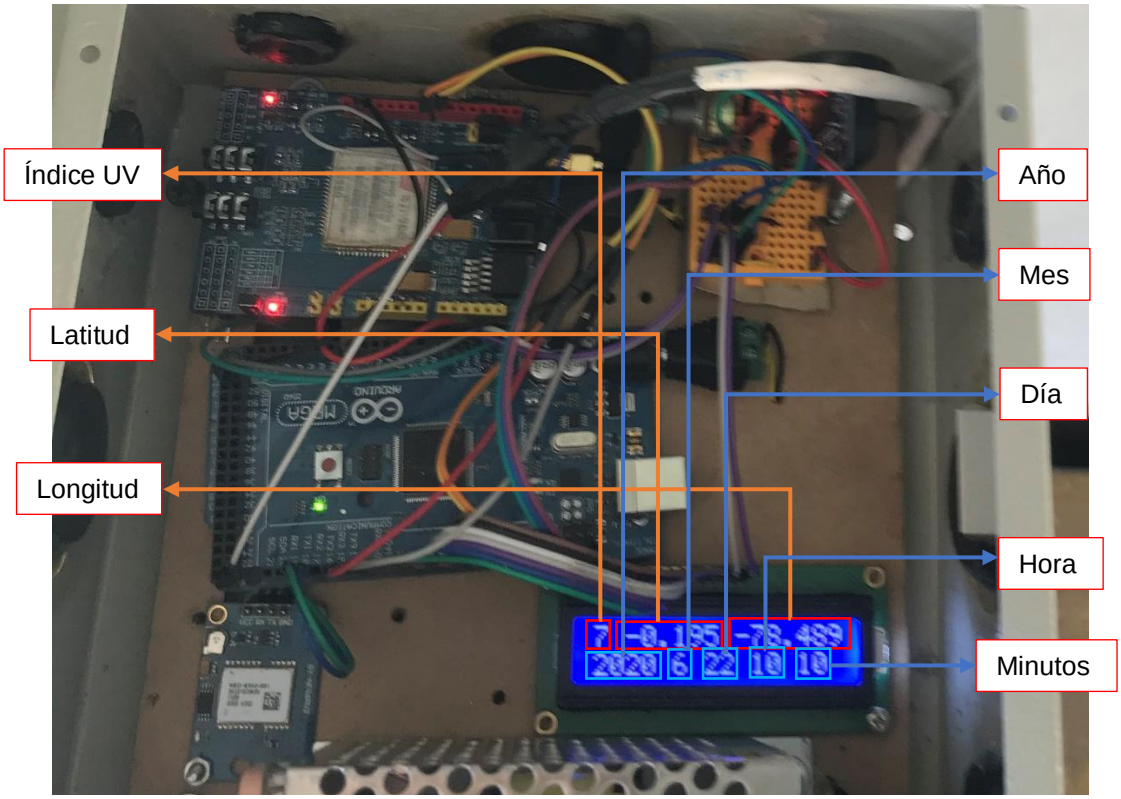


Figura 3.1. Display del sensor UV Fuente: Ian Baquero.

De esta manera se verifica los parámetros disponibles del sensor UV, listos para ser enviados a las horas que este programado en el código de la placa de desarrollo Arduino. Para este Trabajo de Titulación, se programó las horas de envió de muestras cada media hora dentro del rango de 7 am a 6 pm.

3.1.2. ETAPA DE ENVÍO DE MUESTRAS

Esta etapa consiste en verificar el envío de la muestra al servidor web, para que una vez dentro del servidor se valide la muestra y posteriormente se almacene en la base de datos de la nube. Para ello, a través del terminal de la placa de desarrollo Arduino se podrá observar el proceso de configuración y envió de la muestra por parte del módulo sim 900 como se muestra en la Figura 3.2. Cabe mencionar que el módulo sim 900 o GSM/GPRS es el encargado de enviar las muestras por la red GSM de teléfonos celulares para recibir y enviar datos.

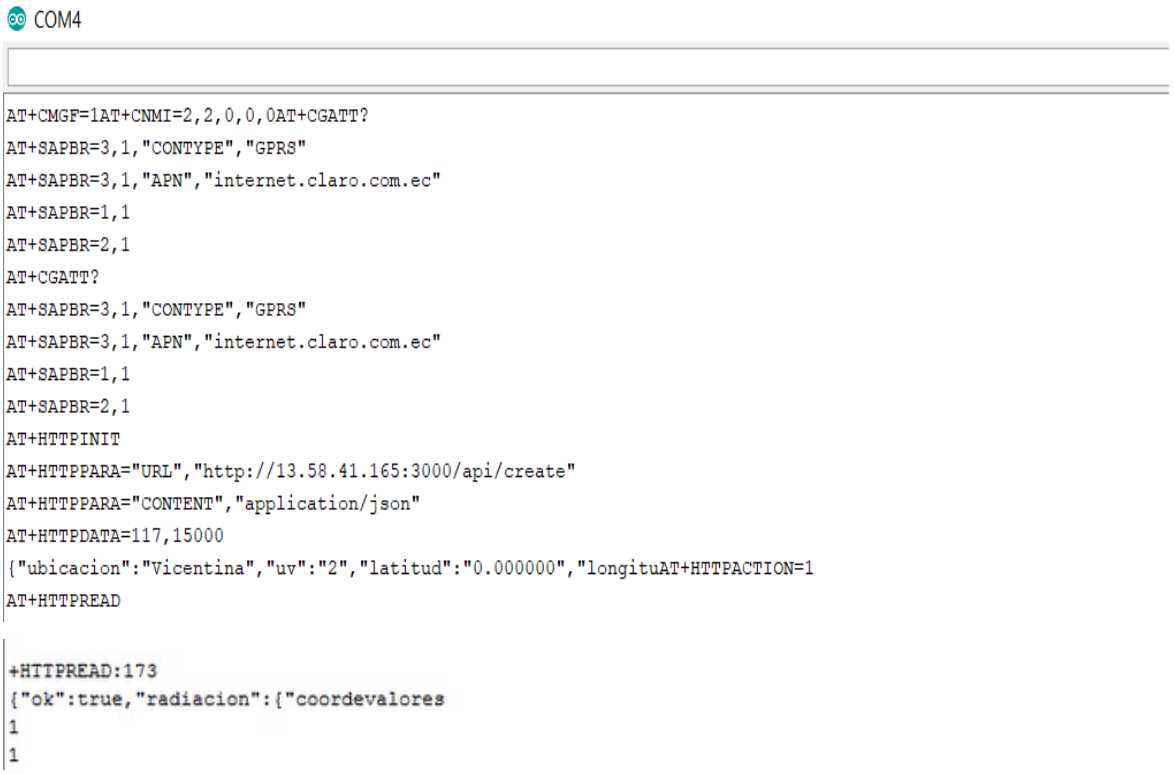


Figura 3.2. Configuración y envió de muestra. Fuente: Ian Baquero.

Tabla 3.1. Tabla de comandos AT utilizados en el módulo sim 900 [42]

Comando AT	Respuesta	Descripción Comando AT	Descripción Respuesta
AT+CMGF	1	Selecciona el formato del mensaje SMS.	Modo Texto.
AT+CNMI	2,2,0,0,0	Indicaciones del mensaje SMS.	Configuración del módulo para que nos muestre los SMS recibidos por comunicación serie.
AT+CGATT	?	Adjuntar o desconectar del servicio GPRS.	OK.
AT+SAPBR	3,1, “CONTYPE”, ”GPRS”	Ajustes del portador para aplicaciones basadas en IP.	Preparar la conexión GPRS.
	3,1, “APN”, “internet.claro.com.ec”		Configuración del proveedor de internet.
	1,1		Conexión con el proveedor de internet.

	2,1		Conexión con el proveedor de internet.
AT+HTTPINIT		Inicializar el servicio HTTP.	
AT+HTTPPARA	“URL”, “dirección IP del servidor web”	Configuración de los parámetros HTTP.	Destino: servicio crear muestra del servidor web.
	“CONTENT”, “application/json”		Configuración del formato de mensaje.
AT+HTTPDATA	117, 15000	Introducción de la data HTTP.	117 (tamaño en bytes de la data a ingresar), 15000 (tiempo máximo en milisegundos para ingresar la data).
AT+HTTPACTION	1	Configuración del método HTTP.	Configuración del método POST.
AT+HTTPREAD	173	Leer la respuesta HTTP.	Tamaño en bytes de la respuesta HTTP.

El módulo sim 900 envía la muestra en formato JSON usando el protocolo de comunicación HTTP, a la URL con las siguientes especificaciones: dirección IP, puerto del servidor web en la nube y ruta del servicio crear muestra. Si la muestra se almacena de manera exitosa, el servidor web responde con un objeto JSON, dentro del cual contiene un campo ok = true, para verificar la correcta inserción de la muestra en la base de datos. No obstante, si la muestra no puede ser almacenada en la base de datos, el servidor web responde con un mensaje detallando de manera breve el error capturado. El módulo sim 900 enviará dos veces más la muestra en caso de una inserción fallida y si el problema persiste, este enviará un SMS al abonado preestablecido en el código del Arduino, para especificar que hubo un error en la inserción de la muestra, como se observa en la Figura 3.4.

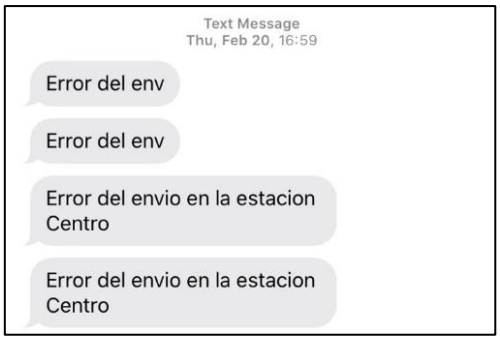


Figura 3.3. SMS de error de envió de muestra. Fuente: Ian Baquero.

Debido que el mensaje SMS de error de envió de muestra se detalla en el abonado como en la Figura 3.3. solo cuando el SIM tiene una cuenta de prepago o post pago en la operadora CLARO, se vio conveniente solo recibir mensajes SMS como se muestra en la Figura 3.4. con la desventaja de que no se especificaría el sensor UV que tuvo el error de envió, pero con la ventaja de saber que hubo una alerta de un posible error en alguno de los sensores UV.

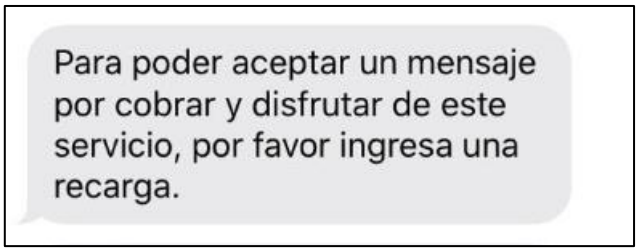


Figura 3.4. Nuevo mensaje SMS de error de envió de muestra. Fuente: Ian Baquero.

3.1.3. ETAPA DE PRESENTACIÓN DE DATOS

Esta etapa consiste en presentar las muestras de la base de datos, en los diferentes módulos de la aplicación móvil. Se verifica el funcionamiento de todo el sistema con pruebas, tanto para un dispositivo iOS como para un Android como se observa en las siguientes Figuras:

- **Dispositivo iOS**

El nombre del modelo de dispositivo móvil utilizado para las pruebas de funcionamiento es iPhone 7 perteneciente al sistema operativo iOS con versión de software 13.6. A través de la tienda virtual App Store se descargó la aplicación **Radiacion** en el dispositivo móvil. Posteriormente se realizó las pruebas en cada uno de los módulos del dispositivo móvil como se muestra en la Figura 3.5.

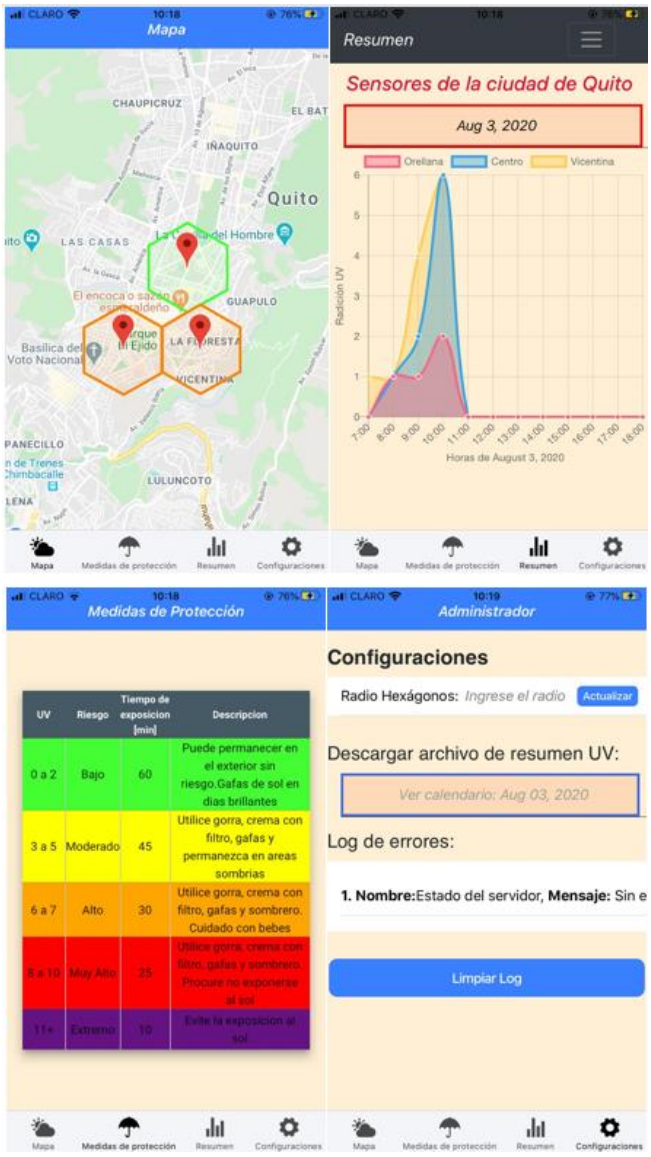


Figura 3.5. Pruebas de funcionamiento dispositivo iOS. Fuente: Ian Baquero.

- **Dispositivo Android**

El nombre del modelo de dispositivo móvil utilizado para las pruebas de funcionamiento es Samsung-G950U perteneciente al sistema operativo Android versión 8.0.0 (API 24). A través de la tienda virtual Play Store se descargó la aplicación **Radiacion** en el dispositivo móvil. Posteriormente se realizó las pruebas en cada uno de los módulos del dispositivo móvil como se muestra en la Figura 3.6.

Figura 3.6. Pruebas de funcionamiento dispositivo Android. Fuente: Ian Baquero.

3.2. CORRECCIÓN DE ERRORES

Los errores presentados durante el proceso de implementación del sistema se resolvieron paulatinamente, añadiendo y actualizando las soluciones a la sección respectiva del documento escrito, tales como:

- Habilitación del tráfico en texto claro a la dirección y puerto del servidor web para los dispositivos Android como se observa en la Figura 2.77. A pesar de haber corregido mucho de los niveles API o versiones de este sistema operativo habilitando el tráfico de datos en texto claro, no se logró habilitar estas autoridades de certificación (CA) en algunas versiones por lo que esta se encuentra funcionando en niveles de API 27 e inferiores.

- Implementación de la librería momentjs en la aplicación móvil para manipulación del campo hora que contiene la fecha y hora de las muestras tomadas.
- Mensaje SMS de error de envió de muestra como se muestra en la Figura 3.4.
- El lenguaje PHP/PDO y la arquitectura del sistema fueron modificados del esquema principal de la Figura 1.2. Se descartó el uso del lenguaje PHP/PDO, ya que la comunicación entre la base de datos en la nube con cualquier servidor sea local o esté en la red; se realiza la conexión a través del protocolo TCP y a través del lenguaje de consulta NoSQL que utiliza JavaScript existe la comunicación entre estas dos capas lógica y de datos. Para los servidores web de datos de sensores y de la aplicación móvil como se muestra en la Figura 1.2. El objetivo principal de tener ambos servidores operativos era separar los servicios creación de muestras por parte de los sensores UV y de acceso de datos para la aplicación móvil. Debido a que el servicio IaaS contratado en la nube para AWS cuenta con una prueba gratuita durante doce meses con 750 hora al mes, no se tomó en cuenta que el servicio correspondía unicamente a la creación de un servidor web. Esto quiere decir que la creación de un servidor web adicional representaría un cargo efectivo extra que saldría de la prueba gratuita. La desventaja de este suceso, provoco que se integren ambos servidores en uno solo. Debido a que la carga de procesamiento de las peticiones por parte de los sensores UV no es tan considerable, no se considera una disminución del rendimiento del servidor web. Por lo tanto, la arquitectura final del sistema se puede observar en el diagrama de despliegue de la Figura 2.16.
- Las funciones del rol administrador dentro de la aplicación móvil como se menciona en el plan de titulación se vieron modificadas por las funciones de la Tabla 2.1. debido a que el proceso CRUD en este rol no tenía un objetivo claro. Realizar CRUD en las subáreas del mapa, significaría poder crear, leer, actualizar y borrar; ya que estas subáreas se crean y actualizan automáticamente por cada muestra tomada por el sensor UV, realizar el proceso CRUD quitaría validez a las mediciones tomadas. Por otra parte realizar el proceso CRUD en la tabla de medidas de protección, no tendría un objetivo claro, debido a que esta tabla es de propósito informativo y en caso de querer actualizar la misma, se realizaría una nueva actualización de versión de la aplicación para las tiendas virtuales Google Play y App Store. Finalmente el proceso CRUD para el grafico dinámico no tendría un objetivo claro. Ya que solo se necesita un gráfico dinámico para mostrar todos los servicios web.

3.3. IMPLEMENTACIÓN DEL TEST MOS

El objetivo de realizar el test MOS consiste en medir la calidad de experiencia del usuario (QoE), después de utilizar la aplicación móvil. El método de respuesta cuantificada para medir la satisfacción del usuario se explica en el diseño del test MOS del capítulo anterior.

El test MOS se diseñó para nueve personas calificadas con siete preguntas cada una. Una para la escala de opinión y otra para la escala de dificultad. El modelo del test MOS se encuentra en el ANEXO II. El test se desarrolló en Google Forms.

El test se realizó a personas anónimas calificadas que cumplieron con el siguiente perfil: Estudiantes egresados o graduados de tercer nivel en las carreras de ingeniería electrónica en redes de la información. Con personas calificadas se referirá a individuos con conocimiento básico en tecnologías de desarrollo de aplicaciones y programación.

A continuación, se mostrará las nueve preguntas y sus resultados respectivos:

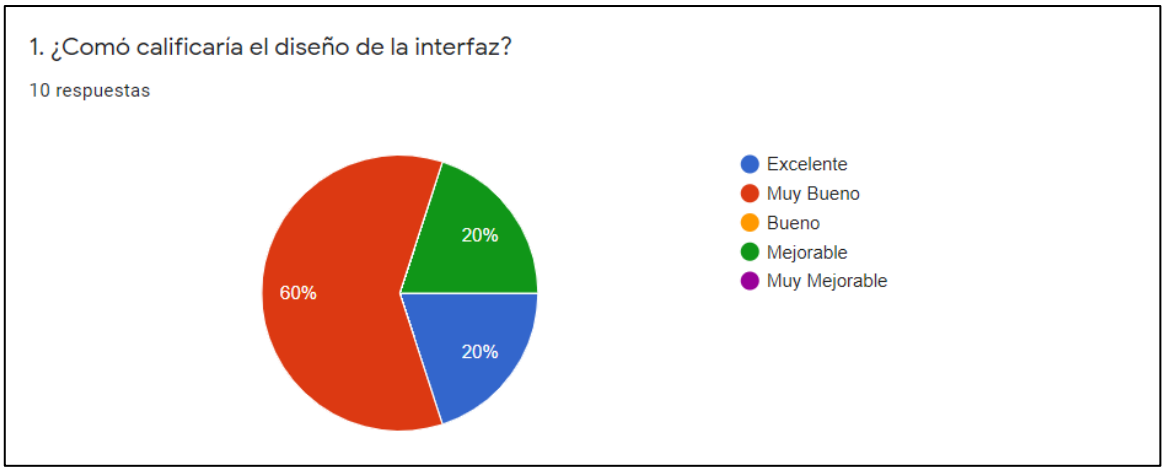


Figura 3.6. Resultados pregunta 1.

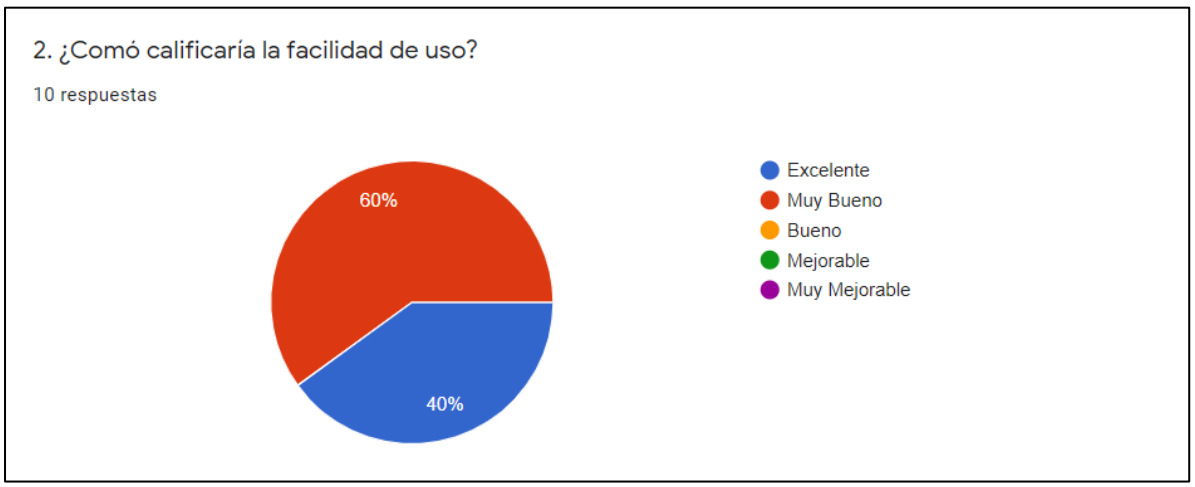


Figura 3.7. Resultados pregunta 2.

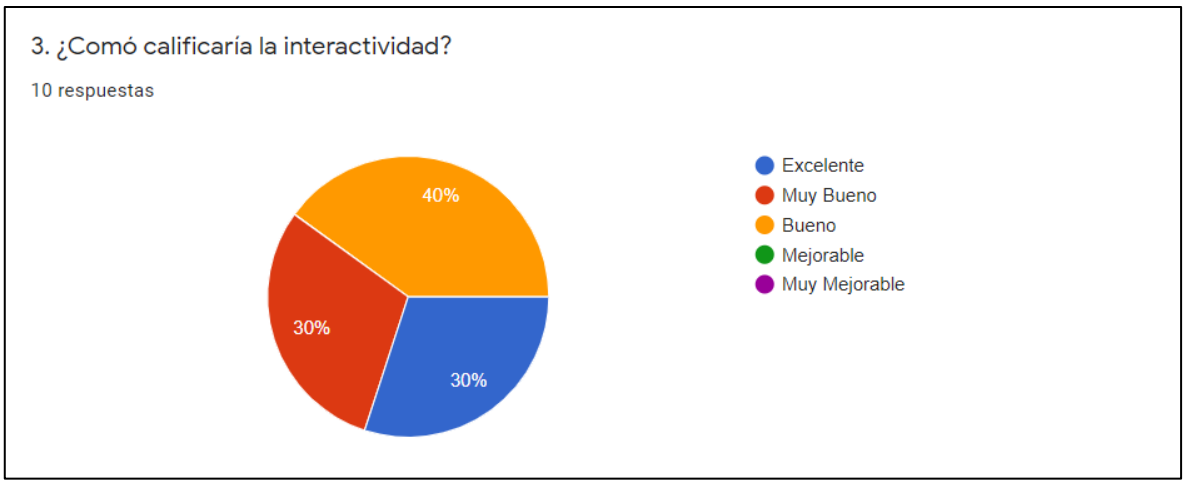


Figura 3.8. Resultados pregunta 3.

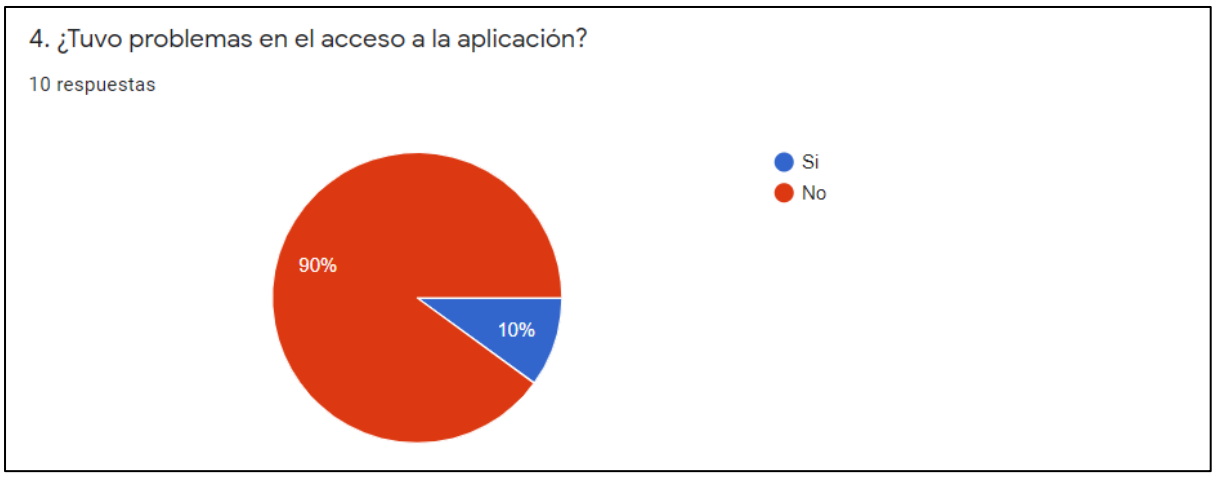


Figura 3.9. Resultados pregunta 4.

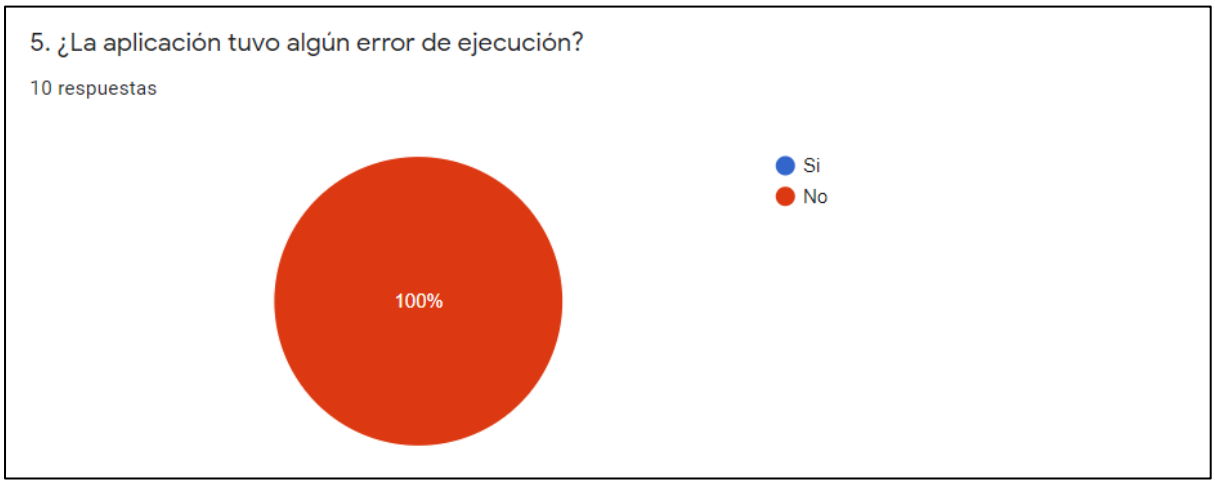


Figura 3.10. Resultados pregunta 5.

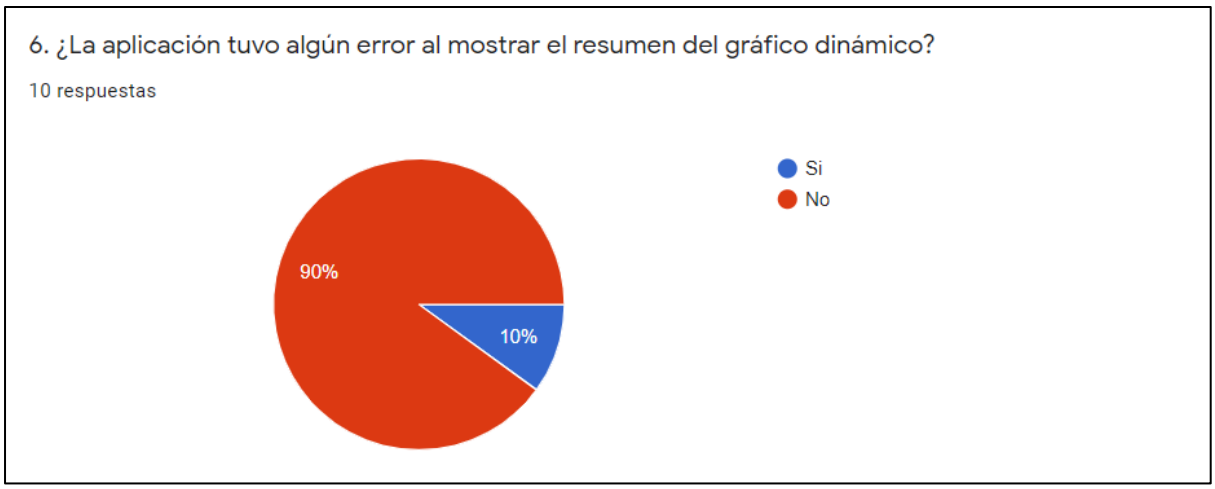


Figura 3.11. Resultados pregunta 6.

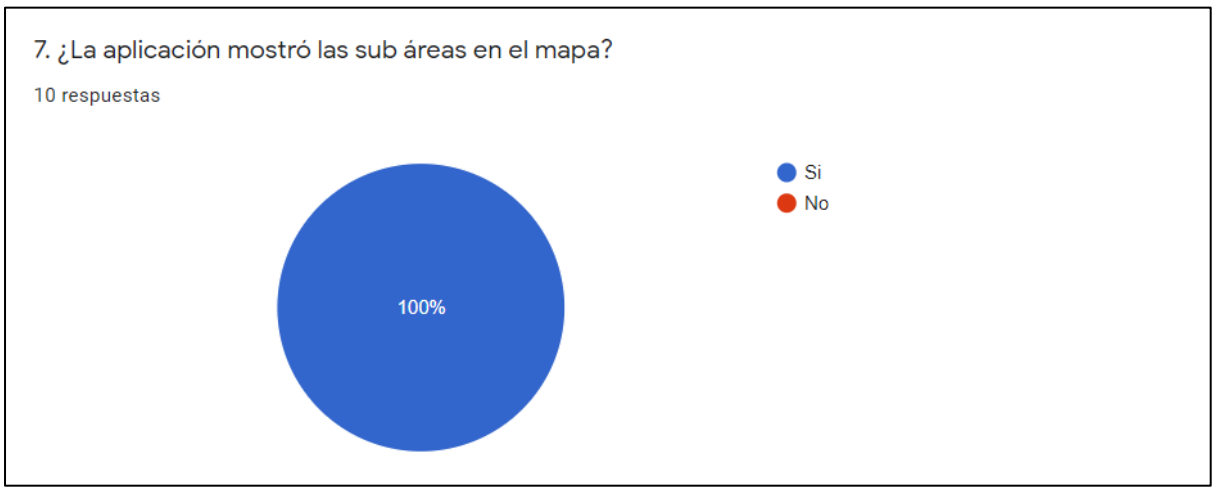


Figura 3.12. Resultados pregunta 7.

El problema de la pregunta 4 se debe a un usuario con problemas al iniciar la aplicación móvil explicado como un bloqueo o cierre automático al momento de ser ejecutada. Este problema puede deberse a un motivo de escasez de almacenamiento, ya que la aplicación se bloquea por que no se puede compilar de manera correcta.

El error de la pregunta 6 se debe a que un usuario que rindió el test MOS con un dispositivo Android con nivel de API mayor a 27 no pudo recibir el tráfico en texto claro. Este error se menciona en el subcapítulo 3.2 de “CORRECCIÓN DE ERRORES”.

En base a los resultados del test de dificultad la cual está conformada por las preguntas 4, 5, 6 y 7. Podemos concluir que las preguntas 4 y 6, tienen un resultado negativo del 10%. Por lo tanto, esto representa facilidad de uso en su mayoría. Para las demás preguntas 5 y 7, tuvieron una buena acogida por parte de los usuarios, lo que representa que no tuvieron dificultad de uso.

En base a los resultados del test de opinión que conforman las preguntas 1, 2 y 3. Se cuantifica el resultado total, a través de una regla de tres para obtener el resultado sobre cinco puntos y se interpreta los resultados como se observa en la Tabla 3.2.

Tabla 3.2. Resultado total test MOS preguntas escala de opinión. Fuente: Ian Baquero.

Pregunta MOS	Total	Interpretación
¿Cómo calificaría el diseño de la interfaz?	3.7	Es buena, pero podría ser muy buena después de realizar algunas mejoras.
¿Cómo calificaría la facilidad de uso?	4.4	Es muy buena, pero podría ser excelente después de realizar algunas mejoras.
¿Cómo calificaría la interactividad?	3.8	Es buena, pero podría ser muy buena después de realizar algunas mejoras.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Una vez desarrollado el Trabajo de Titulación se llegó a la conclusión de que se cumplió el objetivo de informar a los ciudadanos en un mapa el índice de radiación UV más precisos para el área de estudio definido en el alcance, a través de una aplicación móvil, el cual se encuentre disponible en las tiendas virtuales tanto para los sistemas operativos iOS y Android de forma gratuita, para que se tome las medidas de mitigación respectivas y salvaguarde su salud.
- Se analizaron fundamentos teóricos referentes a la radiación UV y técnicos para desarrollo software como: Kanban, computación en la nube, arquitectura de capas, servicios web, APIs, entre otros; además del uso de tecnologías como: MongoDB, NodeJS, Ionic y Express, lo cual fue de beneficio para generar nuevas habilidades e integrar todos estos conceptos de bases de datos en la nube, servicios web y aplicaciones móviles en el presente Trabajo de Titulación.
- El valor del índice de radiación UV se calibró en base a sensores UV operativos de áreas específicas como la del parque Metropolitano o el parque de la Alameda. Después de las pruebas de calibración para los sensores UV, se llegó a la conclusión que las medidas UV obtenidas para los prototipos UV de este Trabajo de Titulación, están acorde a la radiación UV presente en el área de estudio y esto se ve reflejado con el estado climático.
- Mientras se realizaba las pruebas de funcionamiento, se llegó a la conclusión que definir un área que mantenga el índice de radiación UV constante es complejo, ya que

como se mencionó en la introducción existen varios factores que intervienen en el valor del índice de radiación UV, además tal como se puede apreciar en las muestras de los sensores UV, a pesar de que los sensores se encuentran relativamente cercanos las mediciones no siempre son las mismas.

- El cuestionario inicial permitió definir los requerimientos funcionales propuestos en el alcance y conocer nuevos requerimientos funcionales de ser el caso que el usuario tenga interés. En base a las respuestas del cuestionario inicial, se concluye que existe aceptación de los usuarios respecto a los requerimientos funcionales propuestos, ya que no se vieron en la necesidad de añadir nuevos.
- Los resultados del test MOS dieron una calificación positiva en todas las categorías. Por lo tanto, se concluye que gran parte de los usuarios se encuentran satisfechos con la aplicación móvil. Ya que este Trabajo de Titulación es basado en la metodología ágil Kanban, muchos de los requerimientos de la aplicación móvil fueron definidos en el alcance del proyecto, por lo tanto, el uso de esta metodología ágil permitió gestionar el proyecto y tener un acercamiento al uso de metodologías ágiles en el entorno laboral. Para mejorar este Trabajo de Titulación respecto a la calidad de experiencia del usuario, una estrategia puede ser, orientar más la metodología ágil hacia el cliente, esto quiere decir, que el cliente pueda definir los requerimientos funcionales completamente en base a sus necesidades y gustos, además de informarlo de cada avance y logro para retroalimentar el producto final tantas veces sea necesario y tener la máxima aceptación.
- Los gastos estimados invertidos para el presente Trabajo de Titulación en: mano de obra del prototipo UV, materiales, licencia de desarrollador de Apple y Google Play; son de $677,65.

4.2. RECOMENDACIONES

- Analizar detalladamente los términos y condiciones de contratación de servicio IaaS, ya que a pesar de utilizar un Free Tier (nivel gratis), como se mencionó en este Trabajo de Titulación para el caso de AWS, incrementar las horas del servicio integrando un servidor web adicional provocaría un cargo extra por su uso.
- Para los prototipos hardware que son utilizados en este tipo de proyectos, se recomienda contar con asistencia técnica de varios establecimientos, para que, en caso de existir un mal funcionamiento o problema con el prototipo, no depender de un establecimiento de asistencia técnica unicamente.
- Para el desarrollo de todo este Trabajo de Titulación, se utilizó el MEAN STACK (Mongo Express Angular NodeJS), el cual, facilito la programación y diseño de todo el sistema, ya que este STACK cuenta con una amplia gama de librerías fáciles de

instalar y componentes, gracias a que todos utilizan Java Script como lenguaje de programación. Además, toda la integración del sistema que se conforma por la aplicación móvil, servidor web y base de datos fue muy amigable. Por lo tanto, se recomienda manejar un entorno de desarrollo donde todas las capas del sistema utilicen el mismo lenguaje.

- Se recomienda utilizar plataformas de alojamiento de proyectos en la nube como Github para tener copias de respaldo de las versiones del código tanto del servidor web, como de la aplicación móvil, para que en caso de un fallo masivo poder retornar a una versión anterior y partir nuevamente desde esa versión del código.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] "B. La Capa de Ozono y la Radiación Ultravioleta | Ministerio de Ambiente y Desarrollo Sostenible". [Online]. Available: https://www.minambiente.gov.co/index.php/component/content/article/1706-plantilla-asuntos-ambientales-y-

[2] R. Soto, "¿Qué es el índice mundial UV?", El blog de MAPFRE, ago. 20, 2015. [Online]. Available: https://blogmapfre.com/salud/indice-mundial-uv-cuidado-piel/. Accessed on: Aug. 12, 2019.

[3] P. Pérez y J. J. Ruiz, "Calidad de Experiencia en servicios multimedia sobre IP", p. 6. [Online]. Available: http://oa.upm.es/9282/1/INVE_MEM_2010_85904.pdf

[4] "Glosario: Radiación ultravioleta - Comisión Europea". [Online]. Available: https://ec.europa.eu/health/scientific_committees/opinions_layman/artificial-light/es/glosario/pqrs/radiacion-ultravioleta.htm. Accessed on: Jan. 24, 2020

[5] "UV Radiation - The Skin Cancer Foundation". [Online]. Available: https://www.skincancer.org/risk-factors/uv-radiation/. Accessed on: Jan. 24, 2020

[6] H.Q. Urías, B.R. Pérez, "Efectos de la radiación UVB" in *Estadística para Ingeniería y Ciencias*, México, 2014. [Online]. Available: https://books.google.com.ec/books?id=jvLhBAAAQBAJ&printsec=frontcover&hl=es&num=100#v=onepage&q&f=false. Accessed on: Jan. 24, 2020

[7] "Ultraviolet (UV) radiation", The American Cancer Society medical and editorial content team [Online]. Available: https://www.cancer.org/es/cancer/cancer-de-piel/prevencion-y-deteccion-temprana/que-es-la-radiacion-de-luz-ultravioleta.html.Accessed on: Jan. 24, 2020.

[8] "¿Qué medidas sencillas pueden tomarse para protegerse del sol?", OMS [Online]. Available: https://www.who.int/features/qa/40/es/. Accessed on: Jan. 24, 2020.

[9] "Concepto de BASE DE DATOS". [Online]. Available: https://concepto.de/base-de-datos/.

[10] "Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar", Feb. 28, 2014 [Online]. Available: https://www.acens.com/comunicacion/white-papers/bases-de-datos-nosql/ Accessed on: Jan. 24, 2020.

[11] R. H. Gómez and A. M. Iglesias. Maqueda, "MongoDB", *BASES DE DATOS NOSQL: ARQUITECTURA Y EJEMPLOS DE APLICACIÓN*, p. 116, Jun. 2014. [Online]. Available: https://core.ac.uk/download/pdf/44310803.pdf

[12] "ARQUITECTURA 3 CAPAS", *PROGRAMACIÓN WEB*. [Online]. Available: https://edgarbc.wordpress.com/arquitectura/.

[13] “Cloud computing: una guía de aproximación para el empresario”, INCIBE, Oct.17, 2017. [Online]. Available: https://www.incibe.es/protege-tu-empresa/blog/cloud-computing-guia-aproximacion-el-empresario

[14] “Modelos de servicios en la nube de SaaS PaaS IaaS", IBM. [Online]. Available: https://www.ibm.com/es-es/cloud/learn/iaas-paas-saas Accessed on: Jan. 24, 2020.

[15] “¿Qué son los web services y qué tecnología usar en su desarrollo?”, Arsys. [Online]. Available: https://www.arsys.es/blog/programacion/web-services-desarrollo/. Accessed on: Jan. 24, 2020

[16] “Servicios Web RESTful. Invocación de servicios”, Universidad de Alicante, Jun. 19, 2011 [Online]. Available: http://www.jtech.ua.es/j2ee/2010-2011/restringido/servc-web/sesion03-apuntes.html

[17] “¿Qué es una API REST? | Guía 【2020】 - Idento”. [Online]. Available: https://www.idento.es/blog/desarrollo-web/que-es-una-api-rest/. Accessed on: Jan. 24, 2020

[18] “Ejemplo de la estructura de documento en formato JSON”. [Online]. Available: https://www.researchgate.net/figure/Figura-3-Ejemplo-de-la-estructura-de-documento-en-formato-JSON-Fuente-autor_fig3_305311329. Accessed on: Jan. 24, 2020

[19] “El desarrollo de aplicaciones móviles nativas, Web o híbridas”, IBM Software, WebSphere, United States, Apr. 2012 [Online]. Available: https://pdfslide.net/documents/27754-ibm-wp-native-web-or-hybrid-2846853.html

[20] “Cross platform approach for mobile application development: a survey”. [Online]. Available:https://www.coursehero.com/file/73397751/Crossplatformapproachformobileaplication-1pdf/.

[21] "File:Apache Cordova Architecture.png", *Commons.wikimedia.org*, 2020. [Online]. Available: https://commons.wikimedia.org/wiki/File:Apache_Cordova_Architecture.png. Accessed on: Jan. 24, 2020

[22] “HYBRID MOBILE APPLICATION BASED ON IONIC FRAMEWORK TECHNOLOGIES”. [Online]. Available: https://www.researchgate.net/publication/322397904_HYBRID_MOBILE_APPLICATION_BASED_ON_IONIC_FRAMEWORK_TECHNOLOGIES. Accessed on: Jan. 24, 2020

[23] “Ionic platform architecture”. [Online]. Available: https://www.researchgate.net/figure/onic-platform-architecture-1-Apache-Cordova-framework-Apache-Cordova-is-an-open-source_fig9_319135581. (consultado Jan. 24, 2020).

[24] “¿Qué es kanban? Una introducción a la metodología Kanban”. [Online]. Available: https://resources.collab.net/agile-101/what-is-kanban. Accessed on: Jan. 29, 2020

[25] "What is Kanban?", *Everyday Kanban*, 2020. [Online]. Available: https://www.everydaykanban.com/what-is-kanban/. (consultado Jan. 29, 2020).

[26] “¿Qué es una tarjeta Kanban?” *Kanban Software for Agile Project Management*. [Online]. Available: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-tarjeta-kanban/. Accessed on: Jan. 29, 2020.

[27] "¿Qué es un tablero Kanban?" *Kanban Software for Agile Project Management*. [Online]. Available: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-tablero-kanban/. Accessed on: Jan. 29, 2020.

[28] "Why Visual Studio Code?" [Online]. Available: https://code.visualstudio.com/docs/editor/whyvscode. Accessed on: Jan. 30, 2020.

[29] Pablo. M. S, "Administra bases de datos MongoDB con Robomongo", Node.js y MongoDB, may 17, 2014. [Online]. Available: https://nodeymongo.wordpress.com/2014/05/17/administra-bases-de-datos-mongodb-con-robomongo/. Accessed on: Jan. 30, 2020.

[30] "Trello. Qué es, Para Qué Sirve y Cómo Funciona", nov. 29, 2019, *Expertosnegociosonline.com*. [Online]. Available: https://www.expertosnegociosonline.com/que-es-trello-para-que-sirve/. Accessed on: Jan. 30, 2020.

[31] C. Kaundart, "La gestión de proyectos como herramienta para desencadenar el máximo potencial de tu equipo de trabajo". *Blog.trello.com*. [Online]. Available: https://blog.trello.com/es/gestion-de-proyectos. Accessed on: Mar. 08, 2020.

[32] "Postman: gestiona y construye tus APIs rápidamente" *Paradigmadigital.com*. [Online]. Available: https://www.paradigmadigital.com/dev/postman-gestiona-construye-tus-apis-rapidamente/. Accessed on: Jan. 30, 2020.

[33] "DIAGRAMAS CLASES, CASOS DE USO, SECUENCIA, DESPLIEGUE OBJETOS, Y ESTADOS", Issuu. [Online]. Available: https://issuu.com/luisfernandosh/docs/diagarmas__1_. Accessed on: Jul. 19, 2020.

[34] "▷ Diagrama de despliegue. Teoria y ejemplos". [Online]. Available: https://diagramasuml.com/despliegue/. Accessed on: Jul. 19, 2020).

[35] "Cómo crear un proyecto en Ionic", CódigoFacilito [Online]. Available: https://codigofacilito.com/articulos/create-project-ionic4. Accessed on: Jul. 29, 2020.

[36] "¿Cómo seleccionar a un proveedor IaaS?" [Online]. Available: http://tuempresaalanube.com/index.php/88-como-funciona/135-como-seleccionar-a-un-proveedor-iaas Accessed on: Feb. 05, 2020.

[37] "AWS vs AZURE - 6 Most Amazing Differences You Should Know", EDUCBA, jun. 19, 2018. [Online]. Available: https://www.educba.com/aws-vs-azure/. Accessed on: Feb. 05, 2020.

[38] "Latitud", Wikipedia la enciclopedia libre. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Latitud&oldid=123545030. Accessed on: Feb. 20, 2020.

[39] "Grados decimales - Decimal degrees", Wikipedia la enciclopedia libre. [Online]. Available: https://es.qwe.wiki/wiki/Decimal_degrees. Accessed on: Feb. 20, 2020.

[40] "Utilización del middleware de Express". [Online]. Available: https://Expressjs.com/es/guide/using-middleware.html. Accessed on: Feb. 24, 2020.

[41] "How to Import & Export CSV Data using Papa Parse with Ionic", Devdactic, Ago. 20, 2019. [Online]. Available: https://devdactic.com/csv-data-papa-parse-ionic/. Accessed on: Feb. 24, 2020.

[42] “SIM900_AT.pdf”. [Online]. Available: https://www.espruino.com/datasheets/SIM900_AT.pdf. Accessed on: Jul. 31, 2020.

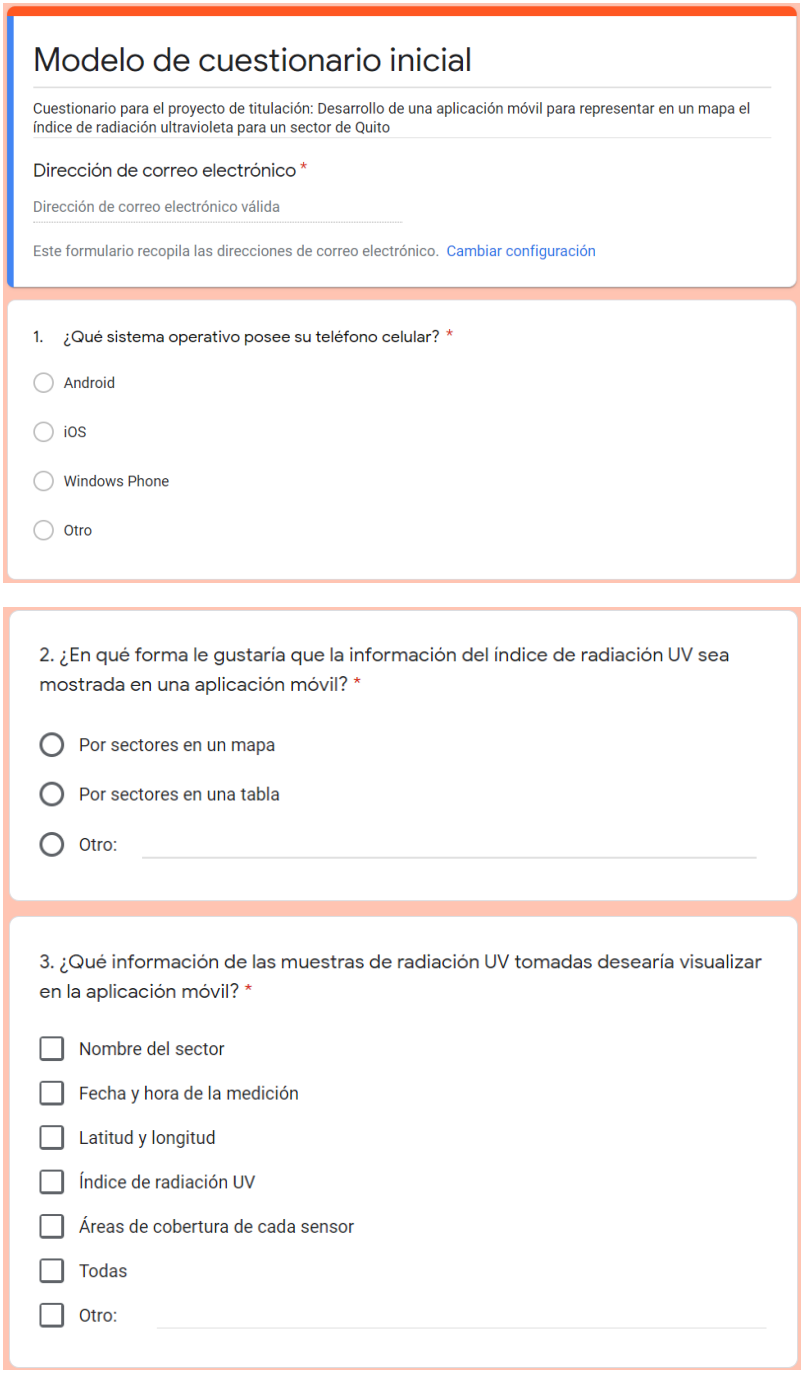
ANEXOS

ANEXO A. Modelo de cuestionario inicial.

ANEXO B. Test MOS.

ANEXO C. Muestras Base de Datos.

ANEXO A



Modelo de cuestionario inicial

Cuestionario para el proyecto de titulación: Desarrollo de una aplicación móvil para representar en un mapa el índice de radiación ultravioleta para un sector de Quito

Dirección de correo electrónico *

Dirección de correo electrónico válida

Este formulario recopila las direcciones de correo electrónico. Cambiar configuración

1. ¿Qué sistema operativo posee su teléfono celular? *

- Android
- iOS
- Windows Phone
- Otro

2. ¿En qué forma le gustaría que la información del índice de radiación UV sea mostrada en una aplicación móvil? *

- Por sectores en un mapa
- Por sectores en una tabla
- Otro:

3. ¿Qué información de las muestras de radiación UV tomadas desearía visualizar en la aplicación móvil? *

- Nombre del sector
- Fecha y hora de la medición
- Latitud y longitud
- Índice de radiación UV
- Áreas de cobertura de cada sensor
- Todas
- Otro:

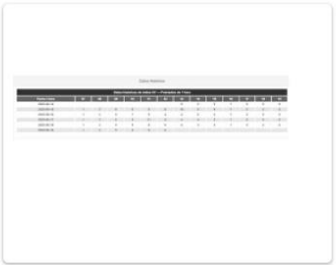
4. Para las medidas de protección contra la radiación UV. ¿Qué información desearía visualizar en la aplicación móvil? *

- [ ] Medidas de mitigación según la escala de riesgo
- [ ] Tiempo de exposición al sol permitida por escala de riesgo
- [ ] Rango del índice UV por escala de riesgo
- [ ] Todas
- [ ] Otro:

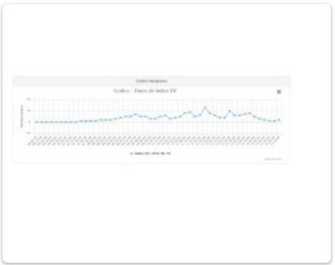
5. ¿Qué formato de notificación le gustaría recibir cuando el riesgo de exposición sea muy alto? *

- ○ Sonidos/Vibración
- ○ Tira Interactiva
- ○ Alerta Interactiva
- ○ Insignia

6. ¿Qué forma de representación le gustaría que tenga el resumen del índice de radiación UV? *



○ Tabla



○ Gráfico

○ Otro:

7. ¿Con qué frecuencia le gustaría que las actualizaciones de las mediciones del índice de radiación UV sean representadas en la aplicación móvil? *

- ○ Cada hora del día
- ○ Cada dos horas al día
- ○ Cada tres horas o mayor en el día

8. ¿En qué escala de tiempo le interesaría visualizar el resumen del índice de radiación UV dentro de la aplicación móvil? *

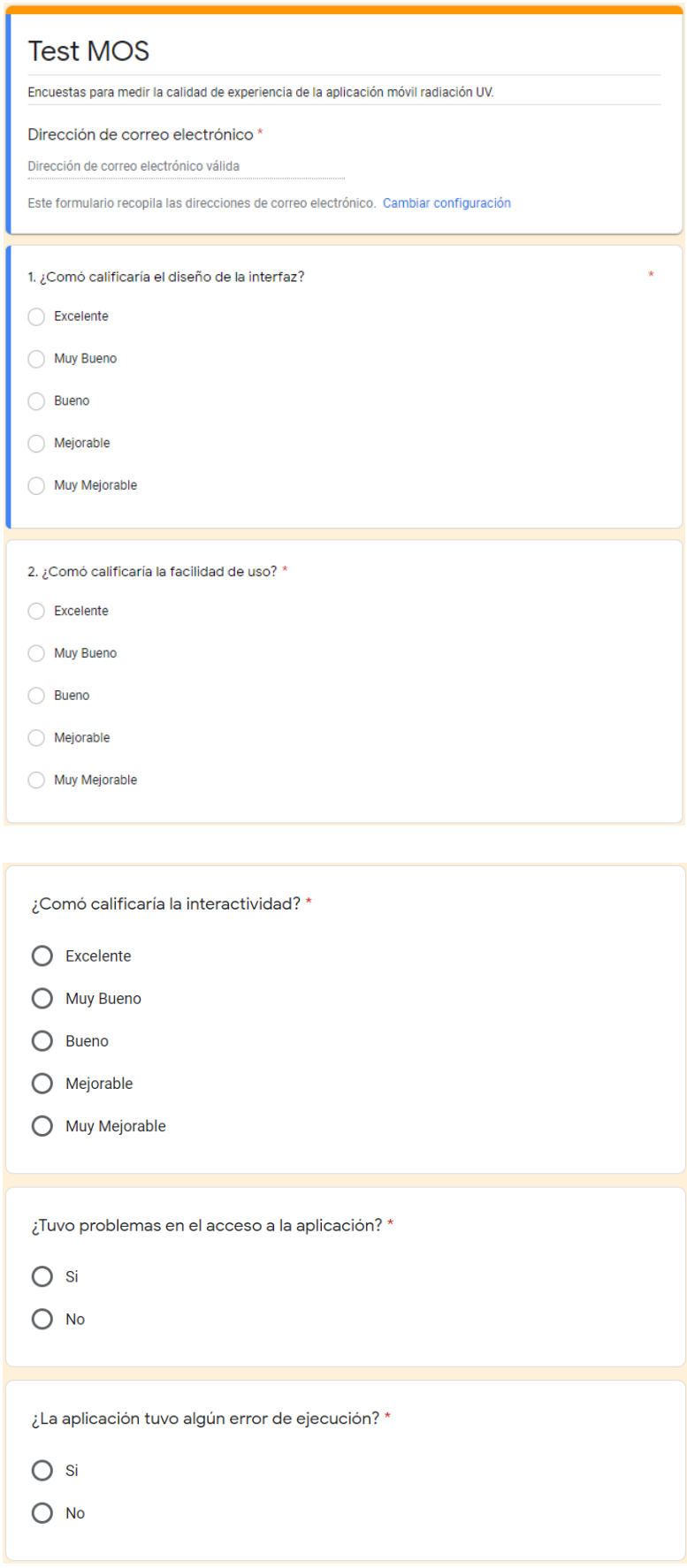
- [ ] Diario
- [ ] Semanal
- [ ] Mensual
- [ ] Todos

9. ¿En qué escala de tiempo le interesaría visualizar los picos de radiación máximos? *

- ( ) Semanal
- ( ) Mensual
- ( ) Ambos

Enviar

ANEXO B



Test MOS

Encuestas para medir la calidad de experiencia de la aplicación móvil radiación UV.

Dirección de correo electrónico *

Dirección de correo electrónico válida

Este formulario recopila las direcciones de correo electrónico. Cambiar configuración

1. ¿Comó calificaría el diseño de la interfaz? *

- Excelente
- Muy Bueno
- Bueno
- Mejorable
- Muy Mejorable

2. ¿Comó calificaría la facilidad de uso? *

- Excelente
- Muy Bueno
- Bueno
- Mejorable
- Muy Mejorable

¿Comó calificaría la interactividad? *

- Excelente
- Muy Bueno
- Bueno
- Mejorable
- Muy Mejorable

¿Tuvo problemas en el acceso a la aplicación? *

- Si
- No

¿La aplicación tuvo algún error de ejecución? *

- Si
- No

¿La aplicación tuvo algún error al mostrar el resumen del gráfico dinámico? *

- Si
- No

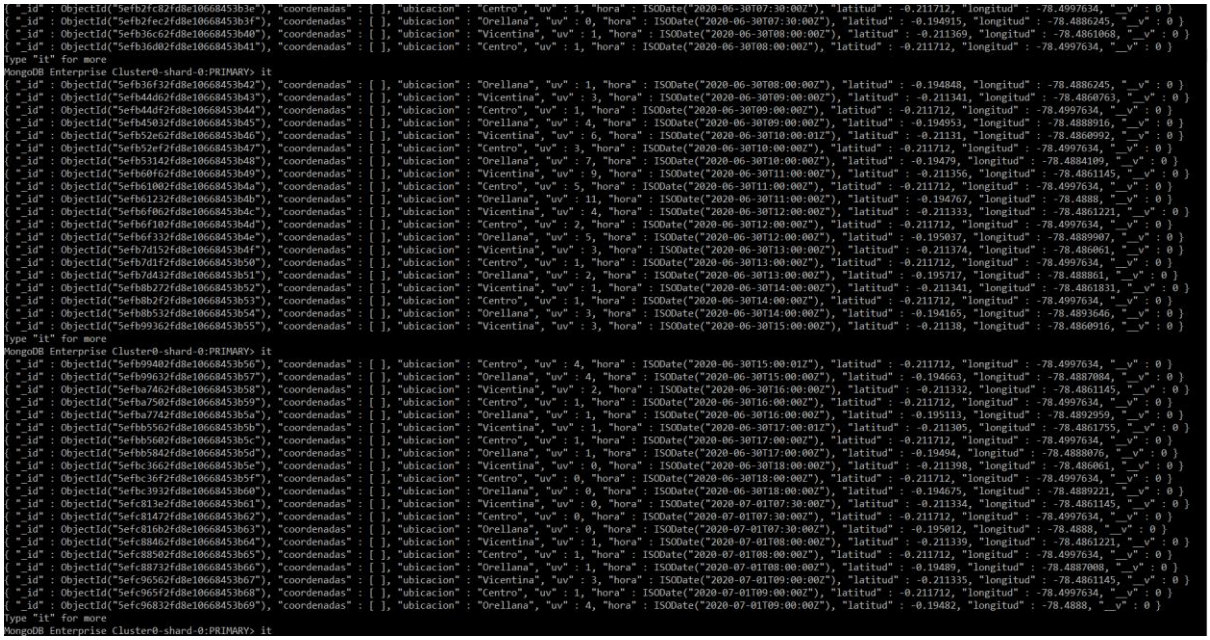
¿La aplicación mostró las sub áreas en el mapa? *

- Si
- No

Enviar

ANEXO C

- **Mongo Shell**



- **Mongo GUI**