ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE APLICACIÓN MÓVIL PARA LA BÚSQUEDA DE INSTRUCTORES ACADÉMICOS BASADA EN UNA ARQUITECTURA MULTINIVEL Y MICROSERVICIOS

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

SANTIAGO SEBASTIÁN SARASTI DUTÁN

DIRECTOR: SORAYA LUCIA SINCHE MAITA PhD.

Quito, marzo 2022

AVAL

Certifico que el presente trabajo fue desarrollado por Santiago Sebastián Sarasti Dután, bajo mi supervisión.

SORAYA LUCIA SINCHE MAITA PhD.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Santiago Sebastián Sarasti Dután, 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.

SANTIAGO SEBASTIÁN SARASTI DUTÁN

DEDICATORIA

Me gustaría dedicar este trabajo de titulación, en primer lugar, a Dios, por el regalo de la vida y la libertad para vivirla.

A mis padres y hermanos, por ser el pilar más grande e importante en mi vida, por demostrarme día a día su cariño y apoyo incondicional. Por las enseñanzas a lo largo de este camino y por nunca dudar de mí, a pesar de los momentos difíciles.

Y finalmente, a mis abuelos, tíos y primos por ser mi mejor ejemplo e inspiración.

AGRADECIMIENTO

Me gustaría expresar un profundo agradecimiento a mi tutora, la profesora Soraya Sinche, por las guías proporcionadas, por su paciencia y experiencia aportada en el desarrollo de este trabajo de titulación.

ÍNDICE DE CONTENIDO

RESUMEN

El objetivo del presente proyecto de titulación es proporcionar un prototipo de aplicación móvil, que permita mejorar el manejo de búsquedas de personal capacitado para ofrecer una determinada tutoría, basado en el lugar de ubicación geográfica, mejorando la experiencia del usuario. El objetivo es evitar lo que se conoce como pérdidas económicas indirectas. Esto es, el tiempo que se genera en encontrar alguien que imparta una clase, así como el tiempo de traslado a dicha ubicación. El trabajo comprende 4 capítulos resumidos a continuación.

En el capítulo 1 se presenta el marco teórico sobre el cual se desarrolló el prototipo de aplicación móvil. En este capítulo se detallan las tecnologías y las herramientas utilizadas.

En el capítulo 2 se presenta la metodología utilizada para el desarrollo y la implementación del sistema prototipo. El diseño de la aplicación incluye los requerimientos funcionales, no funcionales, historias de usuario, diagramas de casos de uso, de base de datos y clases. Este capítulo finaliza con la implementación del sistema prototipo, en el que se incluye, el diseño de las *activities,* parte del código para la creación de la base de datos y el código mas relevante para el funcionamiento de la aplicación móvil.

En el capítulo 3 se incluyen las pruebas de funcionamiento, en el cual se agrega una breve descripción y el resultado obtenido después de realizar las pruebas por cada uno de los módulos implementados.

En el capítulo 4 se presentan las conclusiones en base a los resultados obtenidos en las pruebas de aceptación. Además, se agregan recomendaciones para futuras implementaciones y mejoras del sistema prototipo.

Finalmente, en el apartado de anexos se incluye el procedimiento de instalación de los ambientes de desarrollo, el despliegue de la instancia EC2 y los métodos para obtener los ejecutables de la aplicación móvil, como de los microservicios.

PALABRAS CLAVE: Tutorías Académicas, Arquitectura Multinivel, Extreme Programming, Cliente REST, Android, PostgreSQL, EC2.

ABSTRACT

Due to the growing market demand for private tutoring to reinforce students' knowledge, there has been a need to create a space for those who want to offer tutoring services. The present degree work shows the design and development of a prototype system that allows improving the management of searches for tutors to offer a certain tutoring, based on the place of geographical location, improving the user experience. The objective is to avoid what is known as indirect economic losses. That is, the time it takes to find someone to teach a class and the travel time to that location.

The project is developed using the agile Extreme Programming (XP) methodology, which has the functional requirements and is making use of user stories. The components of the system are defined in the different diagrams, in order to model the prototype of the system for its implementation. For the coding of the client, the Android Studio platform is used, which is based on the Android programming language, the server is developed on Spring Boot Framework platform using the Java programming language and is running on an Elastic Computing (EC2) instance through Amazon Web Services (AWS). The mobile application will be able to connect to the server through endpoints. Finally, the database engine is PostgreSQL.

KEYWORDS: Academic Tutoring, Multilevel Architecture, Extreme Programming, REST Client, Android, PostgreSQL, EC2.

1. INTRODUCCIÓN

Actualmente las tecnologías de la información y la comunicación permiten la construcción de redes de instructores para apoyar el aprendizaje. Desde la llegada de Internet se ha optado por hacer uso de las llamadas Fuentes de Información, las mismasque se pueden describir como una página web o aplicación móviles en la cual se presenta al usuario un conjunto de datos basados en búsquedas, donde la finalidad es presentar la información de instituciones educativas o personas que se encuentren en la capacidad de brindar tutorías académicas particulares como respuesta a la misma.

El mercado de oferta de tutorías académicas particulares ha crecido exponencialmente, debido a que representa una ayuda y acompañamiento importante a lo largo de la vida estudiantil, de jóvenes y adultos.

El presente trabajo propone el desarrollo de una plataforma virtual móvil que permita poner al alcance de quien lo necesite una base de datos de instructores basadas en búsquedas personalizadas.

Mediante este prototipo de aplicación móvil, se permitirá relacionar a estudiantes e instructores por medio de una búsqueda selectiva de asignaturas, dentro de un área geográfica cercana al lugar de la búsqueda y basada en la calificación de cada instructor. La calificación constará de un sistema de cinco estrellas, siendo cinco la calificación más alta y uno la más baja, esta calificación será dada por los estudiantes que realicen la contratación del servicio de tutorías académicas y ayudará al instructor a posicionarse en los lugares más altos de la búsqueda. De esta manera se puede garantizar resultados personalizados y específicos para cada búsqueda.

1.1. OBJETIVOS

El presente trabajo de titulación tiene como objetivo desarrollar un prototipo de aplicación móvil, para la búsqueda de instructores académicos basados en una arquitectura multinivel y microservicios.

Los objetivos específicos son:

- Analizar las diferentes herramientas que se encuentran disponibles para el desarrollo de aplicaciones móviles.
- Diseñar una aplicación móvil basado en el lenguaje de programación Android.
- Implementar la aplicación Android.

- Analizar los resultados del prototipo.

1.2. ALCANCE

El desarrollo de este prototipo móvil se basará en la metodología *Extreme Programming* (XP), debido a que ésta se enfoca en resultados a corto plazo y se encuentra dedicada a equipos pequeños o medianos. El uso de esta metodología permitirá verificar los resultados y en el caso de existir errores se podrán realizar las correcciones. [1]

Se establecerán los requisitos funcionales y no funcionales en base a encuestas a estudiantes de bachillerato y educación superior, para obtener los requisitos mínimos que deberá cumplir la aplicación.

En la Figura 1.1 se muestran las fases que posee XP y cómo estas se desarrollan de manera iterativa, es decir, es posible dividir cada una de ellas en pequeños avances o actividades, enfocándose en un resultado a corto plazo y que pueden ser verificados en el caso que existiese errores. [2]

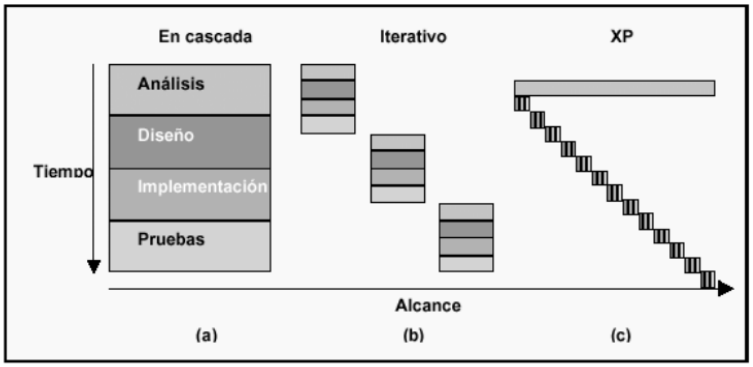

Figura 1.1. Fases de desarrollo de la metodología XP [5]

Se utilizará una arquitectura multinivel, con la finalidad de separar roles, facilitar el mantenimiento y ofrecer una posible escalabilidad a futuro. Además, se usará el patrón de diseño Modelo-Vista-Controlador (MVC) con la finalidad de reutilizar el código. Las capas de la arquitectura son:

- **Capa Presentación:** o interfaz de usuario, que es la dedicada a facilitar al usuario la interacción con la aplicación. Mantiene los objetos que se encargarán de comunicar al usuario con el sistema, todo esto basado en el intercambio de información desplegando y capturando datos. Además, se realizará el procesamiento de datos de manera superficial, es decir, verificando su validez y formato. La interfaz gráfica se la realizará en Android Studio. Por motivos visuales de diseño se hará uso plantillas ya prediseñadas, disponibles en internet y se las adaptarán a las necesidades del prototipo [3].

- **Capa Lógica de Negocios:** donde se definen todos los lineamientos que se deben cumplir para que el programa se ejecute de manera correcta. Es aquí donde se ubica la lógica del programa, la estructura de datos y los recursos necesarios para realizar la interacción entre ellos. Además, en esta capa se realiza el procesamiento de la información ingresada por el usuario final [3]. El servicio será codificado en la herramienta Spring Boot Framework de acceso libre [4] y usando el lenguaje de programación Java, la misma que será posteriormente cargada al servicio de *Elastic Compute Cloud* (EC2) de Amazon en la nube [5].
- **Capa de Datos:** donde se realizará las transacciones con la base de datos con el fin de obtener o ingresar información al sistema [6]. Para el almacenamiento se usará la base de datos de acceso libre PostgreSQL.

En la Figura 1.2 se muestra la arquitectura a utilizar en el diseño del sistema. El mismo que consta de un servidor, el cual se encontrará funcionando en la nube *Amazon Web Services* (AWS) en una instancia del servicio EC2. En este servidor se encontrarán la capa de Lógica de negocios y la capa Base de datos; la conexión con la capa presentación se realizará mediante el protocolo *Hypertext Transfer Protocol* (HTTP) y se usará el cliente REST denominado Retrofit [12], el mismo que permite usar peticiones como GET, POST, PATCH, DELETE y HEAD.

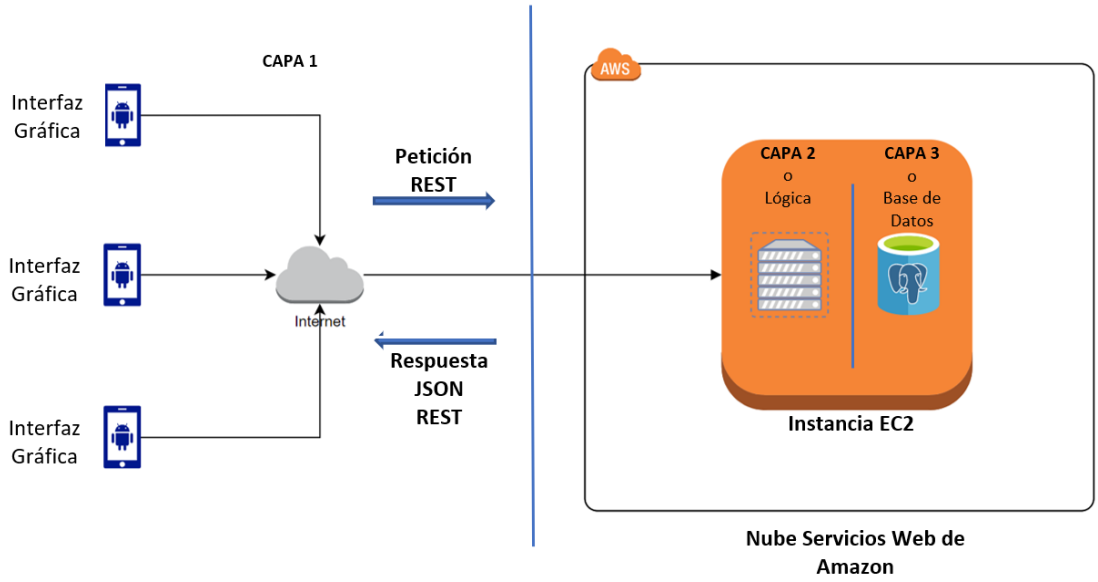

Figura 1.2. Arquitectura del Sistema a desarrollar [3]

El sistema constará de los siguientes módulos:

- **Gestión de Usuarios Estudiante:** que tendrá la capacidad de registrar usuarios al sistema y poder realizar cambios a los mismos de ser necesario.

- **Gestión de Usuario Instructor:** permitirá el registro de instructores y edición de datos personales.
- **Gestión de Búsqueda:** permitirá realizar la búsqueda de un instructor basado en la ubicación geográfica y la calificación media obtenida en base al historial de calificaciones registradas por los usuarios, de esta manera se desplegará un mapa y una lista con los detalles de la búsqueda por cada instructor.
- **Gestión de Calificaciones:** calculará el ponderado de todas las calificaciones obtenidas por cada instructor, lo que permitirá ubicar a los instructores en una lista de respuesta, mientras más alta la ponderación más alto se ubicará en las posiciones de la lista.

El sistema constará de 4 roles:

- **Administrador del Sistema:** Tendrá el control total de los módulos antes mencionados, así como la capacidad de deshabilitar cuentas; además estará a cargo de revisar los datos de todos los usuarios registrados. El administrador será capaz de hacer uso del CRUD, según la aplicación lo requiera. El usuario administrador tendrá la facultad de revisar, evaluar y aprobar los perfiles de los instructores que deseen registrarse en la plataforma.
- **Usuario Estudiante No Registrado:** Podrá realizar búsquedas de los instructores y de sus materias; el usuario estudiante no registrado tendrá acceso a los detalles de las búsquedas siempre y cuando éste se registre mediante su correo y contraseña, no se puede ingresar opiniones o calificaciones.
- **Usuario Estudiante Registrado:** Podrá realizar las búsquedas de instructores basados en filtros provistos por el sistema y visualizar los detalles de sus servicios. Los principales filtros que se usarán son: nivel de estudio, distancia, área del conocimiento y ponderación. El usuario estudiante registrado podrá calificar e ingresar comentarios del servicio prestado por el usuario instructor.
- **Usuario Instructor:** Será capaz de registrarse llenando su hoja de vida y aportando evidencias de su experiencia en el campo como requisito para ser aceptado en la plataforma, deberá ingresar certificados de experiencia laboral y especificar las áreas de conocimiento, ya sea en bachillerato o en educación superior.

  El usuario instructor tendrá acceso a prestar sus servicios siempre y cuando éste se haya registrado mediante su correo y contraseña. No puede ser parte de los resultados de búsqueda si éste no se encuentra registrado.

La evaluación de usabilidad del prototipo se realizará analizando el entorno y los usuarios que utilizarán el producto. Es decir, las pruebas de funcionamiento se realizarán con al menos cinco instructores, cincos usuarios de búsqueda, un historial de al menos cinco evaluaciones a dichos profesores, el uso de Google Maps para comprobar de manera empírica las distancias de geolocalización y un administrador del sistema que hará uso del CRUD. Las pruebas estarán restringidas a la zona geográfica de la ciudad de Quito.

Este trabajo de titulación tendrá un producto final demostrable.

1.3. MARCO TEÓRICO

1.3.1. SERVICIOS WEB

Los Servicios Web son diseñados para permitir la inter operabilidad entre los diferentes equipos que forman parte de una red de datos. Esto con la finalidad de acceder a las interfaces de programación de las aplicaciones (API), que se encuentran disponibles en Internet. Entre las implementaciones más comunes de un Servicio Web se puede mencionar *Simple Object Access Protocol* (SOAP) y *Representational State of Transfer* (REST). Ambos son ampliamente utilizados para realizar la comunicación cliente y servidor, la diferencia radica en que SOAP está basado en el intercambio de mensajes *Extensible Markup Language* (XML) y REST utiliza mensajes del tipo *JavaScript Object Notation* (JSON), para realizar la comunicación. Es necesario, que tanto cliente como servidor conozcan el tipo de mensajes que se usarán para permitir un encapsulado y des encapsulado correcto de peticiones y respuestas. [7]

Los servicios web son elementos que permiten la comunicación entre aplicaciones, independientemente del lenguaje, o la plataforma que se haya utilizado para su desarrollo.

1.3.1.1. Arquitectura del Sistema

La arquitectura que hace uso de un Servicio Web puede ser utilizado por clientes de Aplicaciones Móviles, Portales Web, etc. A continuación, se detallan los componentes [8]:

- **Portales Web, Aplicaciones Web, Servicios Web:** entidades que buscan la conexión con el Servicio Web. La comunicación puede darse de dos formas: directa, cuando se encuentra en el mismo dominio del negocio o inalámbrica, e indirecta, cuando se encuentra en un dominio distinto.
- **Servicio Web:** se encarga de administrar y permitir el acceso a la información. Además, es el encargado de recibir y responder peticiones. El Servicio Web permite

establecer la comunicación mediante el uso del protocolo HTTP con rutas específicamente asignadas.

- **Queues:** permite manejar colas de trabajos de manera organizada. Previene la pérdida de peticiones y respuestas almacenándolas en una cola.
- **Microservicios:** son unidades atómicas. Se las define como interfaces que procesan datos independientemente.

En la Figura 1.3, se visualiza la arquitectura de un servicio web con sus componentes.

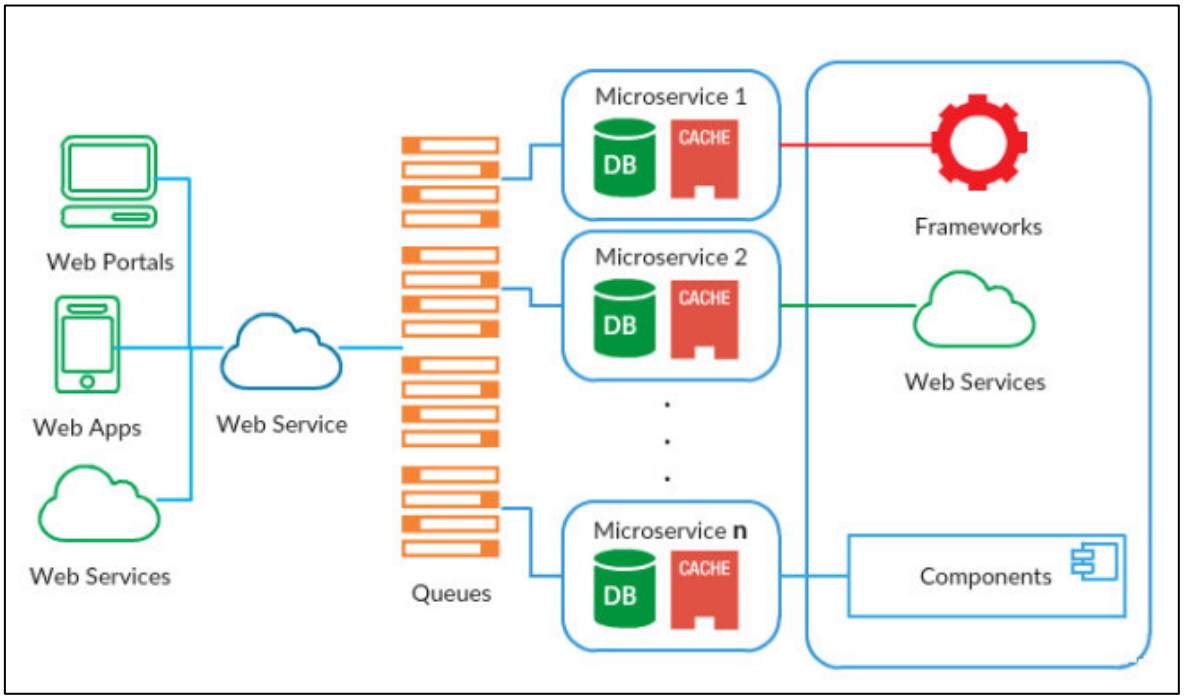

Figura 1.3. Arquitectura de un servicio Web [8]

1.3.1.2. Arquitectura Multinivel

Se lo define como una vista abstracta en la cual se detallan las definiciones, relaciones y reglas entre las diferentes estructuras que conforman la aplicación. Una arquitectura multinivel o modelo de servicios, establece tres conjuntos de funcionalidades, estos conjuntos son llamados lógicas. En la **Figura 1.*4*** se muestra como se compone una arquitectura multinivel de tres capas.

- **Lógica de Presentación:** controla los aspectos relacionados entre la interacción del usuario y la aplicación. Esta capa es la responsable de todas las tareas que el usuario puede realizar como cliente, en la arquitectura general. El objetivo de esta capa, es independizar la interfaz del usuario de los procesos, es decir, realiza la llamada a procesos que se encuentran en las otras capas, pero la ejecución es transparente. [9]

- **Lógica de Negocios:** Está a cargo de controlar la secuencia en la que se ejecutarán las acciones desde la lógica de presentación. Además, se encarga de asegurar la integridad de las transacciones. El objetivo principal es aislar las reglas que regirán en la aplicación y gestionar los datos obtenidos del usuario. [9]
- **Lógica de Datos:** Se encarga del mantenimiento de los datos y su almacenamieto. Esta tarea es realizada por sistemas de gestión de bases de datos relacionales, como MySQL, PostgreSQL Oracle, etc. [9]

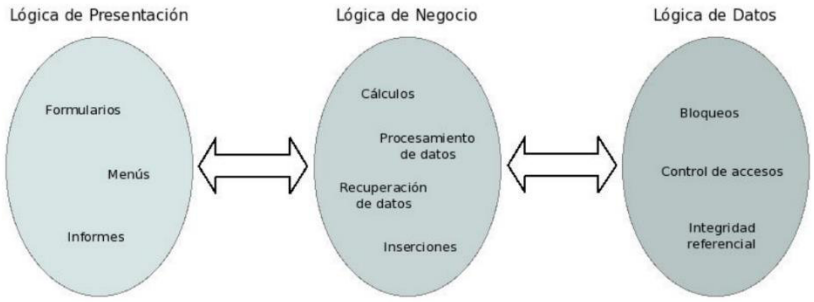

Figura 1.4. Arquitectura Multinivel de Tres Capas [9]

La comunicación para el intercambio de mensajes entre la capa de negocio y la capa presentación se llevará a cabo mediante una interfaz REST.

1.3.1.3. *REPRESENTIONAL STATE OF TRANSFER* - REST

REST es una interfaz que permite plantear una arquitectura del tipo cliente-servidor, en la cual cada uno de los servicios se identifica como un recurso del sistema y cada recurso tiene asignada una dirección URL.

En general, el concepto de REST es un grupo de URIs unificado. Cada una de estas URIs identifican recursos, los cuales se interpretan como objetos conceptuales. La forma de representar esos objetos es mediante el envío de mensajes a través de la web.

REST fomenta la intercomunicación de sistemas heterogéneos, debido a que cualquier servicio puede ser utilizado si un dispositivo soporta el protocolo de Internet HTTP.

REST hace uso mediante el protocolo HTTP de cinco métodos:

- **GET:** permite al servidor leer la información de un recurso.
- **PUT:** modifica o actualiza el estado de un determinado recurso.
- **DELETE:** borra un determinado recurso
- **POST:** crea un estado de un recurso.

- **HEAD:** realiza la comprobación de la información en el caso de que esta haya sido cambiada en el proceso de envío al servidor.

En la arquitectura de REST se puede resaltar tres tipos de elementos, que son: de datos, de conexión y de procesamiento.

Los elementos de datos son aquellos que permiten trasferir los componentes mediante el uso de recursos. Los elementos de conexión presentan una interfaz abstracta que encapsula sus actividades en: conectores clientes, servidor, cache y túnel. Finalmente, el elemento de procesamiento depende del rol que cumple en la aplicación permitiendo que actúen como cliente y servidor, con la finalidad de reenviar solicitudes y respuestas, como se observa en la **Figura 1.*5*** [10].

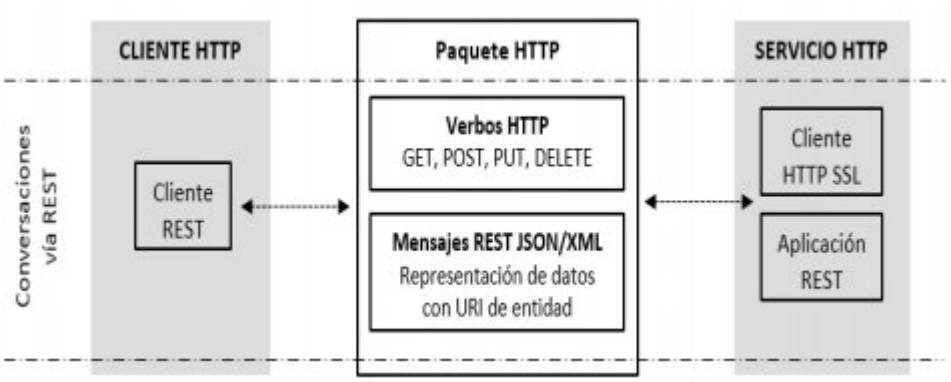

Figura 1.5. Estructura REST [10]

REST soporta el uso de varios formatos de mensajes: XML, HTML, JSON; debido a la simplicidad de los mensajes, JSON es el tipo de mensaje comúnmente usado.

JSON es un acrónimo de *JavaScript Object Notation* y es un formato de tipo ligero para el intercambio de información como alternativa a XML. JSON es un formato independiente, es decir, no tiene relación con ningún lenguaje de programación. Tanto el emisor como el receptor no necesitan hablar el mismo idioma, ya que contienen librerías para codificar y decodificar cadenas en este formato.

Este formato de intercambio de mensajes se emplea en entornos en los cuales el flujo de datos entre el cliente y servidor es de suma importancia. [12]

1.3.1.4. Ventajas de los servicios Web REST [11]

- Escalabilidad: REST posee la habilidad para poder manejar un crecimiento de trabajo de manera continua y de manera fluida, sin la necesidad de realizar grandes cambios y sin verse afectado su rendimiento.

- Funcionamiento independiente: el uso de cabeceras que provee el protocolo HTTP permite la extensibilidad de servidores y clientes ya que pueden funcionar por más tiempo en internet, haciendo uso de recursos URL, es decir, se puede añadir o actualizar las interfaces sin ninguna dificultad.
- Generalidad de interfaces: debido a que se hace uso de HTTP, es factible interactuar con cualquier servicio HTTP, lo que no pasa con SOAP en algunos casos.

1.3.1.5. Desventajas de los servicios Web REST [11]

- En un sistema con esquema REST en el cual se ha implementado más de un servidor, es posible que los servidores no tengan conocimiento entre sí, de su existencia.
- REST no maneja estados, así que no se sabe si un usuario inicio sesión o si se envió la información necesaria al cliente. Por lo que la implementación de *tokens* para indicar que se realizó una petición al servidor en la aplicación, es necesario.
- No es posible saber en qué servidor se ha realizado la petición desde la interfaz de usuario.

1.3.1.6. SOAP

SOAP es un protocolo que se utiliza para establecer el intercambio de mensajes con el Servicio Web. Es el encargado de proveer las directrices básicas para construir el servicio web, está basado en lenguaje XML y se compone de tres partes:

- *Envelope*, define cuál es el contenido del mensaje y cómo se procesa el mismo.
- Los tipos de datos poseen sus conjuntos y condiciones para ser expresados.
- Las llamadas de procedimientos y respuestas poseen su propia convención.

El proveedor de servicios es el encargado de enviar un fichero *Web Services Description Language* (WSDL) al publicador del servicio, con la definición del servicio web. Posteriormente, el solicitante descubre quien es el proveedor del servicio, al interactuar con el publicador. El proveedor se encarga de validar la petición y envía la respuesta en formato XML y de manera estructurada como se muestra en la Figura 1.6. [12]

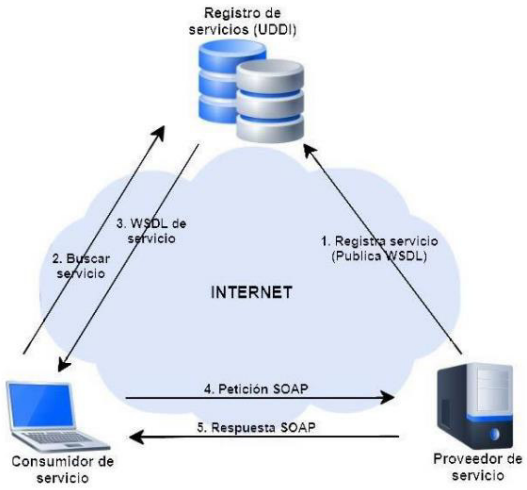

Figura 1.6. Servicio Web basado en el protocolo SOAP [12]

1.3.1.7. REST VS SOAP

SOAP y REST, son ampliamente utilizados para el intercambio de mensajes entre el cliente y el servidor, no obstante, se pueden resaltar varias diferencias entre ellos. SOAP puede ser testeado y depurado, mucho antes de poner en funcionamiento la aplicación. Por otro lado, REST permite una escalabilidad más rápida en todo tipo de sistemas y un escaso consumo de recursos. En la Tabla 1.1 se muestra las diferencias entre SOAP y REST

Tabla 1.1. SOAP vs REST [12]

REST	SOAP
Permite una gran variedad de formatos de datos, como HTML, JSON, XML, etc.	Solo permite el uso del formato de datos XML.
Es una arquitectura.	Es un protocolo.
No puede hacer uso de REST.	Puede hacer uso de SOAP, como protocolo subyacente para servicios web.
Requiere más ancho de banda, debido a que los mensajes contienen demasiada información, por lo que la transferencia de datos aumenta.	No necesita demasiado ancho de banda, ya que el mensaje con la información que se envía al servidor es relativamente pequeño.
El archivo WSDL permite obtener la información necesaria con la finalidad de comprender que servicios se encuentran disponibles en el servicio web.	Permite el uso de identificadores de recursos uniformes (URI) para identificar y localizar el servicio, al que se desea acceder.

1.3.2. PATRONES DE DISEÑO

Los patrones de diseño, son maneras de estandarizar y resolver problemas de diseño relacionados a desarrollo de software. Un patrón de diseño tiene como objetivo [13]:

- Utilizar una estructura estándar para realizar el diseño
- Facilitar la lectura del código para futuros programadores.
- Usar catálogos para elementos que se pueden reutilizar en el diseño de software.

1.3.2.1. Patrones de Diseño Modelo-Vista-Controlador (MVC)

El patrón de diseño MVC es un paradigma capaz de dividir las partes de un modelo en tres: el Modelo, la Vista y el Controlador, de esta manera, se permite la implementación por separado de cada una de ellas, facilitando la actualización y el mantenimiento del software.

El uso de este patrón, permite identificar errores de manera más fácil para poder arreglarlos sin mayor complicación, sin la necesidad de afectar a otras piezas dentro de la aplicación.

Los componentes de este patrón de diseño son los siguientes [13] [14]:

- Modelo: Es el encargado del manejo y control de los datos y sus transformaciones. El modelo no conoce nada de la vista y el controlador, por lo que cualquier proceso que se lleve a cabo en las otras capas, es transparente. El sistema es el encargado de realizar el enlace del modelo a las vistas y de manejar las notificaciones en caso de cambios.
- Vista: Permite la visualización de los datos que se representan en la capa modelo y los presenta al cliente. Esta capa interactúa con la capa Controlador.
- Controlador: Es el encargado de dar un significado a las peticiones que llegan desde el cliente y actúa sobre los datos representados en la capa Modelo. El controlador, es el encargado de entrar en acción cuando se recibe alguna notificación de cambio, ya sea dentro de la capa Modelo o por alguna alteración en las Vistas.

En la Figura 1.7 se muestra la interacción entre los componentes MVC.

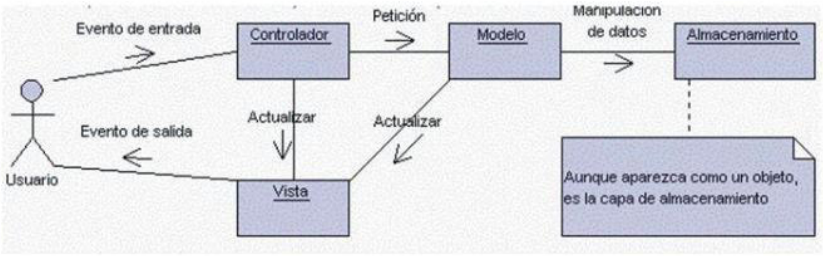

Figura 1.7. Relación entre los elementos del patrón de diseño MVC [13]

1.3.3. BASE DE DATOS RELACIONALES

Una base de datos relacional ayuda a modelar problemas de la vida real y permite administrar datos de manera dinámica. La idea fundamental, es permitir la relación de las

diferentes entidades del diagrama, como conjuntos de datos que tienen el nombre de tuplas. Cada relación, se puede interpretar como una tabla compuesta por registros que son representados por campos y por tuplas.

En una base de datos relacional es posible visualizar filas y columnas por tabla, en donde, cada una de las filas posee un identificador único llamado clave y cada columna representa los atributos de los datos, cada registro tiene un valor específico para cada atributo.

Las bases de datos relacionales son conocidas por el uso y manejo de políticas y reglas, es decir, cuando se necesitan hacer cambios permanentes en la base de datos. Esta característica se denomina atomicidad. La atomicidad permite precisión en los datos que conforman la base de datos, si una parte de la base de datos está disponible y las demás entidades que forman parte de esta no están disponibles, el cambio no se realiza, hasta realizar el cambio en todas las partes involucradas. [15]

1.3.3.1. PostgreSQL

PostgreSQL es una base de datos relacional, de código abierto, el mismo que es compatible con cualquier sistema operativo como Linux, Windows o Unix. PostgreSQL brinda una mayor escalabilidad debido a que no necesita usar bloqueos de lectura cuando se realiza una transacción. También hace uso de *Hot-Standby* que permite al cliente realizar búsquedas mientras la base de datos se encuentra en modo de recuperación o espera, cabe resaltar, que estas búsquedas solo pueden ser de lectura. Así, se pueden hacer tareas de mantenimiento sin la necesidad de bloquear el sistema por completo. [10]

La herramienta oficial para acceder y realizar cambios de la base de datos es pgAdmin. PgAdmin es un *Query Tool,* esto permite la ejecución de comandos SQL y permite analizar la base de datos, en caso que se presenten errores. [16]

1.3.4. ANDROID

Android es un sistema operativo que inicialmente fue diseñado para el uso en dispositivos móviles, es basado en Linux, gratuito y multiplataforma.

Android es de código abierto y desarrollo libre, lo cual hace que los fabricantes puedan hacer uso de el sin licenciamiento y permite el acceso a cualquier tipo de código con la finalidad de modificarlo para poder adaptarlo a cualquier dispositivo y luego poder distribuirlo. Además, posee gran cantidad de servicios y librerías todo sin la necesidad de hacer uso de librerías externas, como *global positioning system* (GPS), acelerómetro, lectores de barras, etc. [17]

Para desarrollar una aplicación en Android, es necesario un entorno de desarrollo y el kit de desarrollo conocido como *software development kit* (SDK). Además, se debe conocer el porcentaje de cada una de las versiones de Android, con la finalidad de elegir una API mínima para el desarrollo de la aplicación, esto se debe a que las APIs más antiguas no son compatibles con las más recientes. [17]

1.3.5. FRAMEWORK SPRING BOOT

SPRING, está basado en el enfoque "Convención sobre configuración", el objetivo de este enfoque es crear rápidamente aplicaciones basadas en Spring sin la necesidad de repetir la misma configuración una y otra vez. De esta manera, permite que los desarrolladores enfoquen sus esfuerzos en la lógica del programa. [18]

Con la finalidad de alcanzar este objetivo, Spring Boot hace uso de *starters* que son una colección de bibliotecas con dependencias ya preconfiguradas, necesarias para iniciar una funcionalidad en particular. [18]

Entre las características más importantes, es posible mencionar [18]:

- Al ser una extensión de Spring Framework posee las características del mismo.
- No necesita configuración basada en XML o código debido a que usa ficheros ".yaml" o ".properties"
- Soporta tecnologías basadas en base de datos relacionales, y no relacionales, procesamiento por lotes de código, cache y mensajería.
- Permite crear aplicaciones haciendo uso de microservicios

1.3.5.1. Microservicios

Un microservicio, es un componente de software que realiza operaciones específicas y cumple con la lógica de negocios, se ejecutan de forma independiente, autónoma y desacoplada. [19]

Los microservicios fueron creados con la finalidad de facilitar el desarrollo de las aplicaciones web en sistemas distribuidos, en donde sus componentes interactúan entre sí. [19]

Ventajas de arquitecturas con microservicios [19]:

- Debido al desacoplamiento, el escalamiento es mucho más sencillo de implementar.

- Al funcionar de forma independiente, el cambio o modificación resulta fácil y sin afectar a los otros entes de la aplicación.
- Pueden ser implementados en diferentes lenguajes de programación y son compatibles con diferentes tecnologías de bases de datos.

Los microservicios son los encargados de comunicarse con la base de datos, con la finalidad de enviar y obtener información almacenada en las tablas. El uso de Hibernate permite realizar estas transacciones de manera fluida.

1.3.5.2. Hibernate

Actualmente, Hibernate es una de las herramientas más utilizadas por el Framework de Spring, ya que permite el mapeo relacional de objetos. El uso de esta herramienta permite simplificar las transacciones, ya que permite agilizar la relación entre el servicio y la base de datos, evitando redundancia en el código. Hibernate genera el código para las sentencias SQL, como resultado mantiene la portabilidad de los datos, entre los diferentes motores de base de datos.

La combinación de Spring e Hibernate permite la creación de aplicaciones más ágiles ya que se encargan del manejo lógico y de datos, guardando y recuperando hacia y desde cualquier base de datos.

Ventajas del uso de Hibernate [19]:

- Mantenimiento más efectivo, ya que no usa demasiadas líneas de código y resolver errores resulta más fácil.
- Evita el hacer uso de demasiado código, permitiendo concentrarse en lógica del negocio.
- Permite un paradigma 100% orientado a objetos

Algunos de los métodos más usados por Hibernate son:

- save(object)
- update(object)
- saveOrUpdate(object)
- delete(object)

1.3.6. PLATAFORMA *ELASTIC COMPUTING* DE AMAZON - EC2-AWS

Amazon Elastic Compute Cloud es un servicio web en la nube que proporciona una instancia virtual de parámetros modificables. Entre los parámetros modificables es posible mencionar: sistema operativo, memoria, procesamiento y permisos de red [14]. En la Figura 1.8 se listan las instancias disponibles en el servicio EC2 de Amazon.

Instancia	CPU virtual	Memoria (GiB)	Almacenamiento	Rendimiento de red (Gbps)
a1.medium	1	2	Solo EBS	Hasta 10
a1.large	2	4	Solo EBS	Hasta 10
a1.xlarge	4	8	Solo EBS	Hasta 10
a1.2xlarge	8	16	Solo EBS	Hasta 10
a1.4xlarge	16	32	Solo EBS	Hasta 10
a1.metal	16*	32	Solo EBS	Hasta 10

Figura 1.8. Instancias de uso general de AWS – EC2 [20]

Características principales de la plataforma EC2 [21]:

- Instancias dedicadas: Están diseñadas para aquellas aplicaciones que necesitan acceso a las características de software y no pueden ser ejecutadas en entornos virtuales.
- Almacenamiento flexible: Los sistemas de almacenamiento cuentan con respaldo en caso de fallas, para su posterior recuperación.
- CPU optimizados: Permite la creación y configuración de CPU virtuales en base a la necesidad de la aplicación a ejecutarse en dicha instancia.
- Soporta tecnologías basadas en base de datos relacionales y no relacionales, procesamiento por lotes de código, cache y mensajería.

EC2, permite el uso de un entorno virtual auténtico de cómputo, que facilita el uso de interfaces de servicios web en instancias personalizables. [21]

1.3.7. *EXTREME PROGRAMMING* - XP

eXtreme Programming (XP) es una metodología ágil de desarrollo de software, que se centra en la adaptabilidad y previsibilidad, lo que la hace diferente a las metodologías tradicionales. La capacidad de adaptación a los cambios de requisitos, permite al programador terminar con los proyectos más rápido, ya que es más flexible a los contratiempos. [22]

Esta metodología ágil se encuentra basada en las llamadas “Historias de Usuario”, que son tarjetas de papel en las cuales se describen brevemente las características del programa, ya sean requisitos funcionales como no funcionales. El uso que se da a las historias de usuarios permite que en cualquier punto del desarrollo del proyecto estas puedan, cambiar, crearse, destruirse o reemplazarse por otras que se adapten mejor al objetivo del proyecto. Cada una de las historias de usuario debe ser comprensible y claramente delimitada con la finalidad de que cada una de ellas le tome al programador unas pocas semanas en completar. [22]

1.3.7.1. Roles

XP posee varios roles en los cuales se basa la metodología, todos con la finalidad de aportar dinamismo al desarrollo, a continuación, se detallan cada uno de los roles [22]:

- Programador: Es el encargado de desarrollar el código del programa y escribir las pruebas unitarias.
- Cliente: Es el encargado de realizar las historias de usuario y de verificar la funcionalidad del programa.
- Encargado de pruebas: Ejecuta pruebas de funcionamiento de manera regular y comparte los resultados al equipo.
- Encargado de seguimiento: Evalúa el tiempo para el cual está planificado el proyecto y si es alcanzable de acuerdo a los objetivos plateados. Adicionalmente, hace el seguimiento del progreso en las iteraciones.
- Entrenador: Está a cargo de los lineamientos necesarios para que la metodología XP se ejecute de manera correcta
- Gestor: Coordina el vínculo entre el cliente y el programador.

1.3.7.2. Fases de la Metodología XP

El ciclo de vida que propone XP consta de 6 fases [23]:

- Exploración: Las historias de usuario se realizan de manera macro, albergando los detalles generales del producto. Se realiza la elección de las herramientas y tecnologías que mejor se adapten al proyecto. Generalmente, esta etapa toma muy pocas semanas.
- Planificación de entrega: En base a las historias de usuario se establece prioridad y el personal encargado de la programación. Se establece el esfuerzo que tomara

realizar cada una de ellas y en base al esfuerzo se establece un cronograma para la entrega. El tiempo para cada una de las entregas no debe exceder los tres meses. Esta fase no debería durar más de siete días.

- Iteraciones: Para realizar el plan de iteración, se debe tomar en cuenta las historias de usuarios que no se tomaron en cuenta, la velocidad de desarrollo del proyecto, pruebas de aceptación fallidas y tareas pendientes. Cada una de estas acciones debe ser dividida en tareas y asignarlas a un programador para su desarrollo.
- Producción: Se realizan pruebas de rendimiento antes de que el producto sea entregado al cliente, basada en estas pruebas se decide si es necesario agregar características adicionales al producto final entregable.
- Mantenimiento: La metodología XP garantiza que el producto una vez entregado se mantenga en funcionamiento y al mismo tiempo de necesitar nuevas características, se puede agregar iteraciones para desarrollarlas.
- Muerte del Proyecto: No se generan más historias de usuario para ser desarrolladas e incluidas en el sistema. Se realiza la documentación del producto sin la posibilidad de cambios de estructura.

1.3.7.3. Características Entre Metodologías Ágiles y Metodologías Tradicionales

Es posible enumerar las principales características entre estos dos tipos de metodologías, como se muestra en la Tabla 1.2.

Tabla 1.2. Características de metodologías ágiles y tradicionales [23]

Metodología Ágil	Metodología Tradicional
No se maneja gran cantidad de roles dentro del desarrollo del proyecto, ya que, lo roles son flexibles y genéricos.	Se maneja mayor cantidad de roles y son más específicos.
No se hace uso de un contrato tradicional, ya que debe ser flexible a cambios según las iteraciones que se adicionen, quiten, modifiquen o reemplacen.	El contrato es fijo y nada flexible.
La arquitectura puede modificarse y mejorarse mientras dure el desarrollo del proyecto.	Se define una arquitectura fija, generalmente a principios del proyecto.
Se esperan que se implementen cambios durante la ejecución del proyecto.	Se espera que no ocurran cambios durante la ejecución del proyecto que generen un gran impacto en el proyecto.

Metodología Ágil	Metodología Tradicional
El cliente es una parte fundamental del equipo de desarrollo	El cliente no es parte del equipo, solo interactúa con el equipo por medio de reuniones.

2. METODOLOGÍA

El desarrollo de este prototipo se llevó a cabo empleando la metodología ágil XP. Es necesario resaltar que esta metodología de programación es adecuada para sistemas de pequeña y mediana infraestructura. Su adaptabilidad a cambios en la infraestructura, la hacen la mejor opción para el desarrollo de esta aplicación.

2.1. PLANIFICACIÓN

2.1.1. ACTORES

Identificar los actores, determina su función en el sistema. Además, define las acciones permitidas por el sistema para cada uno de ellos. En la Tabla 2.1 se detalla la responsabilidad de cada uno de los actores en el sistema.

Tabla 2.1. Actores del sistema

Actores	Descripción	Responsabilidad
Tutores	Persona encargada de cargar sus datos en la aplicación, para su posterior aceptación y publicación en las búsquedas.	• Cargar los datos requeridos por la plataforma • Encargarse de pedir una calificación al cliente. • Actualizar su información en el sistema
Estudiantes	Usuario encargado de cargar sus datos a la aplicación y realizar búsquedas según los filtros propuestos.	• Registrarse en la aplicación mediante la introducción de sus datos. • Realizar búsquedas. • Calificar a los tutores.
Administrador de la aplicación	Administra a los usuarios de la aplicación	• Crear, modificar, eliminar a los usuarios. • Acepta las aplicaciones de tutores.

2.1.2. HISTORIAS DE USUARIO

La metodología XP emplea las historias de usuario para obtener los requerimientos por parte del cliente o por parte del desarrollador con la finalidad de llevar un control de los avances del proyecto y estimar un tiempo adecuado para su desarrollo.

Actualmente no existe una convención o formato aplicable a las historias de usuario. Por este motivo los elementos a usar en las historias de usuario, serán las siguientes:

- **Identificador:** Se lo conoce como un identificador único de historia de usuario, éste se encuentra definido por las siglas “UH” y a continuación el número que define dicha historia.
- **Nombre:** Se agrega una breve descripción sobre el contenido de dicha historia de usuario.
- **Usuario(s):** Los usuarios que, una vez finalizada la iteración, harán uso de dicho módulo.
- **Prioridad en Negocio:** Se define el grado de prioridad de la historia de usuario, para el desarrollo se utilizará tres niveles: Alta, media y baja
- **Iteración Asignada:** El número de iteración que le corresponde a la historia de usuario.
- **Puntos Estimados:** Especifica un tiempo estimado en horas, la cual tomará en completar una historia de usuario.
- **Programador Responsable:** Programador encargado de desarrollar la historia de usuario.
- **Descripción:** Permite agregar una breve descripción y especifica los requerimientos funcionales para el módulo.

2.1.3. ITERACIONES

En esta sección se especificará las iteraciones que se realizaron en el desarrollo del proyecto; además de una breve descripción de cada una de ellas.

2.1.3.1. Iteración 1

Basados en los requerimientos funcionales, en la primera iteración, se definió los servicios que permiten acceder al sistema, mediante el correo y la contraseña. Adicionalmente, se definió un servicio que permita la administración total del sistema, con el objetivo de validar el perfil del usuario. El administrador del sistema podrá: modificar, activar, desactivar y eliminar cuentas de los usuarios tutor y estudiante.

Tabla 2.2. Historia de Usuario, Login del Sistema

Historia de Usuario	
Código: UH-001	**Nombre:** Login del Sistema
Usuario(s): Estudiante, Tutor y Administrador del Sistema	

Prioridad en Negocio: Alta	**Iteración Asignada:** 1
Puntos Estimados: 20	**Programador Responsable:** Santiago Sarasti
Descripción: El servicio de Login, debe permitir a los usuarios autenticarse en el sistema, dependiendo del rol de cada usuario, será direccionado a las diferentes *Activities*.	

Tabla 2.3. Historia de Usuario, Validar Perfil del Usuario Tutor

Historia de Usuario	
Código: UH-002	**Nombre:** Validar perfil del usuario tutor
Usuario(s): Administrador	
Prioridad en Negocio: Alta	**Iteración Asignada:** 1
Puntos Estimados: 20	**Programador Responsable:** Santiago Sarasti
Descripción: Al momento de que un tutor se registre en el sistema, el mismo debe pasar por un proceso de validación, previo a ser parte del proceso de búsqueda en el sistema, con el objetivo de validar los datos ingresados en el registro. Esta tarea la lleva a cabo el Administrador del sistema.	

Tabla 2.4. Historia de Usuario, Modificar, Activar, Desactivar y Eliminar Cuentas de Usuarios Tutor y Estudiante

Historia de Usuario	
Código: UH-003	**Nombre:** Modificar, activar, desactivar y eliminar cuentas de un usuario tutor.
Usuario(s): Administrador	
Prioridad en Negocio: Alta	**Iteración Asignada:** 1
Puntos Estimados: 20	**Programador Responsable:** Santiago Sarasti
Descripción: El administrador podrá visualizar los perfiles de los usuarios tutor y estudiante con la finalidad de modificar, activar, desactivar y eliminar las cuentas, según se requiera, basado en su uso o mal uso.	

2.1.3.2. Iteración 2

En la segunda iteración, se definieron los servicios de registro. El registro permitirá a los clientes tutor y estudiante ser parte del sistema por medio del ingreso de información.

Tabla 2.5. Historia de Usuario, Registro de Usuario

Historia de Usuario	
Código: UH-004	**Nombre:** Registro de usuario
Usuario(s): Tutor y Estudiante	
Prioridad en Negocio: Alta	**Iteración Asignada:** 2
Puntos Estimados: 40	**Programador Responsable:** Santiago Sarasti
Descripción: El servicio permite el registro de estudiantes y tutores, en el cual se ingresará la información basada en el rol que se seleccione en la aplicación. De esta manera se tiene la siguiente información a ingresar: **1.** El usuario estudiante deberá ingresar la siguiente información: Nombre, apellido, e-mail, contraseña **2.** El usuario tutor deberá ingresar la siguiente información: Nombre, apellido, e-mail, contraseña, fecha de nacimiento, una descripción personal, foto de perfil, hoja de vida y materias de tutoría.	

2.1.3.3. Iteración 3

En la tercera iteración se definió el servicio de búsqueda, el servicio permite encontrar tutores basados en la ubicación geográfica y filtros de materia y nivel de educación de la materia.

Tabla 2.6. Historia de Usuario, Búsqueda de Tutores

Historia de Usuario	
Código: UH-005	**Nombre:** Búsqueda de tutores
Usuario(s): Estudiante	
Prioridad en Negocio: Alta	**Iteración Asignada:** 3
Puntos Estimados: 30	**Programador Responsable:** Santiago Sarasti
Descripción: El módulo de búsqueda permite a un usuario estudiante encontrar tutores en base a su localización geográfica. **1.** Los filtros usados son: la materia, el nivel de educación de la materia y la posición geográfica de la búsqueda. **2.** Se mostrarán los resultados que concuerden con la búsqueda en el mapa. **3.** Se mostrará adicionalmente una lista de tutores, los mismos que tienen la mayor calificación en el sistema.	

2.1.3.4. Iteración 4

En la cuarta iteración se definió los módulos de calificaciones y comentarios, para los tutores. Esta información será proporcionada por parte de los clientes estudiantes que hayan hecho uso de un servicio de tutoría.

Tabla 2.7. Historia de Usuario, Calificaciones del Usuario Tutor

Historia de Usuario	
Código: UH-006	**Nombre:** Calificaciones del usuario tutor
Usuario(s): Estudiante	
Prioridad en Negocio: Media	**Iteración Asignada:** 4
Puntos Estimados: 10	**Programador Responsable:** Santiago Sarasti
Descripción: Las calificaciones se llevarán a cabo mediante un sistema de estrellas, en donde, cinco estrellas es la calificación más alta que se puede obtener y una estrella, la más baja. El resultado del promedio de todas las calificaciones, será la calificación a mostrar en el perfil del tutor.	

Tabla 2.8. Historia de Usuario, Comentarios para Tutores

Historia de Usuario	
Código: UH-007	**Nombre:** Comentarios para tutores
Usuario(s): Estudiantes	
Prioridad en Negocio: Media	**Iteración Asignada:** 4
Puntos Estimados: 10	**Programador Responsable:** Santiago Sarasti
Descripción: El servicio de comentarios debe permitir al usuario estudiante ingresar un comentario en el perfil del tutor con el cual interactuó, con la finalidad de agregar información extra sobre el servicio de tutorías. Los comentarios no podrán ser modificados una vez ingresados en un perfil tutor y podrán ser visualizados por todos los usuarios estudiantes.	

2.1.4. DIAGRAMAS DE CASO DE USO

Los diagramas de casos de uso son utilizados con la finalidad de identificar a los actores que interactuarán con el sistema, así como representar la relación entre los requerimientos del sistema y los usuarios. Los diagramas de casos de uso que se presentan a continuación, han sido realizados de acuerdo a los módulos del cliente.

En la **Figura 2.*1***, es posible visualizar la interacción entre el usuario tutor, el usuario administrador y el sistema. Para acceder al sistema, es necesario que el tutor proporcione los datos necesarios especificados en la *Activity*, para completar el registro. El usuario administrador debe autenticarse, revisar los datos ingresados y aprobar el perfil. Una vez que el perfil de un tutor ha sido aprobado, será posible visualizarlo como resultado, en las búsquedas.

Para realizar cualquier modificación de información en un perfil tutor, será requerida la autenticación del usuario, al cual, pertenece del perfil.

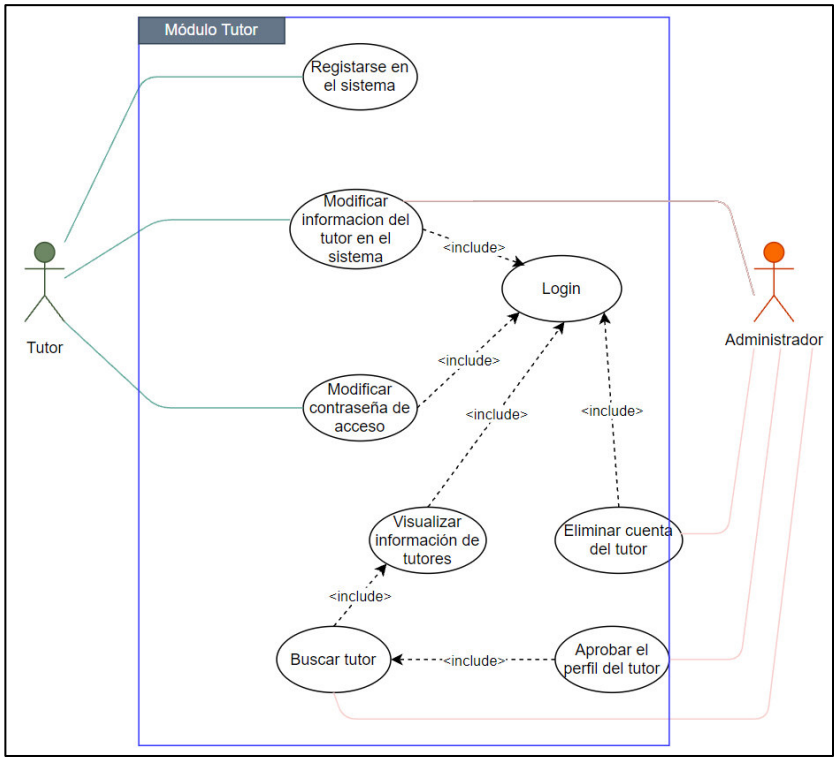

Figura 2.1. Diagrama de casos de uso del módulo Gestión del Usuario Tutor

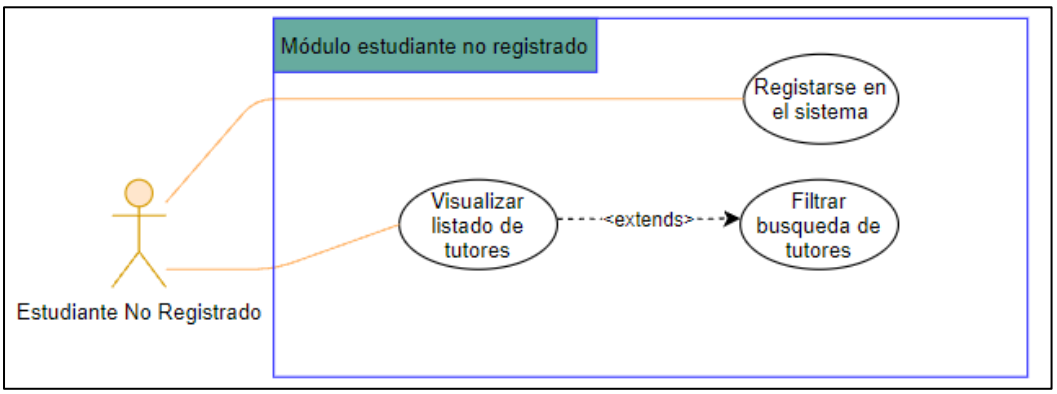

Figura 2.2. Diagrama de casos de uso del módulo Estudiante no Registrado

En la **Figura 2.*2***, se ha diagramado la relación entre el usuario Estudiante no Registrado y el sistema. En este módulo un usuario estudiante que no ha llenado el formulario de registro, solo se le permite realizar búsquedas en la aplicación, mas no visualizar la información de los resultados que sean arrojados.

En la **Figura 2.*3***, se ha diagramado la relación entre el usuario Estudiante Registrado, el administrador y el sistema. Es necesario para ambos actores autenticarse en el sistema. El usuario estudiante registrado, tendrá la capacidad de realizar búsquedas basados en filtros, la aplicación responderá con resultados de la búsqueda basada en la ubicación y podrá visualizar los resultados y detalles de cada uno de ellos. También podrá registrar una calificación para el tutor. El usuario administrado una vez autenticado, podrá hacer uso de los botones de administración del perfil estudiante, en caso de verificar un mal uso de la aplicación.

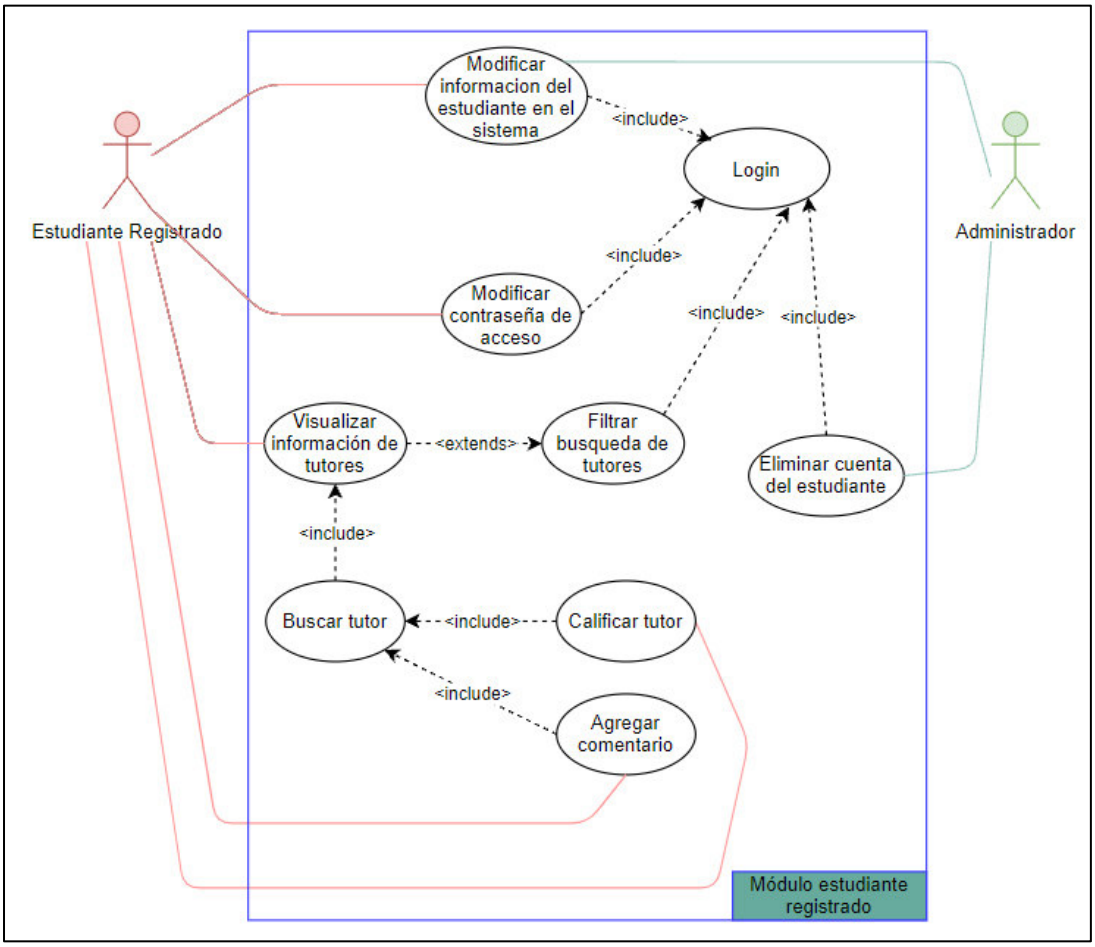

Figura 2.3. Diagrama de casos de uso del Estudiante Registrado

2.1.5. ANÁLISIS DE REQUERIMIENTOS FUNCIONALES

La aplicación tendrá las siguientes funcionalidades de sistema:

- El sistema debe permitir realizar búsquedas, pero no visualizar la información hasta que el cliente se registre
- El sistema debe permitir el registro y actualización del cliente tutor.
- El sistema permite el registro y actualización del cliente estudiante.
- El sistema debe permitir la admisión, modificación y eliminación de los clientes tutores registrados en base a los datos ingresados para su visualización en los resultados de las búsquedas.
- El sistema permite realizar búsquedas basada en filtros.
- El sistema permite visualizar los resultados de la búsqueda en el mapa, adicionalmente los resultados con mayor calificación.
- El sistema permite agregar comentarios sobre los servicios de cada tutor.
- El sistema permite visualizar la información de cada tutor

2.1.6. REQUERIMIENTOS NO FUNCIONALES

La aplicación tendrá los siguientes requerimientos no funcionales:

- El dispositivo debe tener contar con el servicio de GPS y con una localización válida para que sea posible visualizar la posición en el mapa y los resultados de la búsqueda a su alrededor.
- El dispositivo debe poseer conexión a internet para poder hacer uso del servicio que se encuentra alojado en la nube de AWS.
- El dispositivo debe tener una versión Android2.0 o superior.
- Todas las funcionalidades de la aplicación deberán estar accesibles a través de la interfaz de usuario por medio de peticiones que cumplan con la arquitectura REST.
- Los datos de usuarios que se obtienen mediante la aplicación, deberán estar almacenados en un sistema gestor de bases de datos, sobre el cual puedan realizarse consultas no previstas en la actualidad.
- Los datos de la aplicación solo podrán ser modificados por personas autorizadas mediante los permisos que maneja la aplicación. Los perfiles de usuario de la aplicación serán los siguientes: administrador, usuario tutor, usuario estudiante registrado y usuario estudiante no registrado.

2.2. DISEÑO

Detalla el diseño del sistema prototipo, para la búsqueda de tutores en un área geográfica. Se ha utilizado el Lenguaje de Modelado Unificado (UML), con la finalidad de documentar los módulos que forman parte del sistema.

Mediante el uso del lenguaje UML, se realizaron los siguientes diagramas:

- Diagrama de la base de datos: permite mostrar la estructura de las tablas y la relación entre ellas, para el desarrollo del sistema.
- Diagrama de clases: define la estructura que tendrán los objetos en el sistema.

2.2.1. ARQUITECTURA DEL SISTEMA PROTOTIPO

El sistema está basado en una arquitectura cliente – servidor, como se visualiza en la **Figura 2.*4***. La interacción la inicia el cliente por medio de la aplicación Android, al enviar una petición de tipo REST hacia el servidor. La petición por medio de *endpoints* llega al microservicio que funciona en una instancia EC2 de AWS. Una vez la petición se haya procesado, se encapsula en un mensaje de tipo JSON y se envía como respuesta al cliente. La comunicación cliente- servidor, se llevará a cabo mediante el protocolo HTTP.

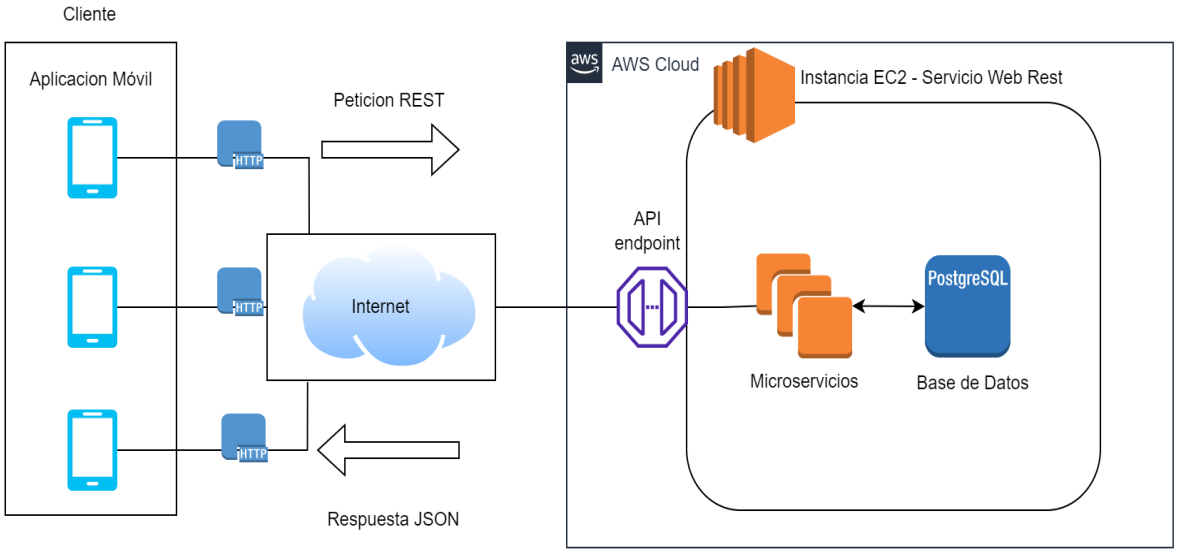

Figura 2.4. Arquitectura del sistema prototipo

2.2.2. DIAGRAMA DE LA BASE DE DATOS

En la **Figura 2.*5*** se puede observar el diagrama de la base de datos, en el cual constan, las tablas utilizadas en el desarrollo del sistema prototipo y la relación entre las tablas que lo componen. A continuación, se presenta un breve resumen de las tablas utilizadas:

- **Usuario:** Almacena la información básica para los usuarios administrador, tutor y estudiante.
- **Cliente:** Contiene la información de los clientes que utilizan los servicios del sistema, como son el tutor y estudiante.
- **Tipo Cliente:** Esta tabla es utilizada para diferenciar los tipos de clientes.
- **Profesor:** Contiene información específica de un tutor.
- **Categoría:** Almacena las materias que dispone el sistema para la búsqueda.
- **Nivel Escolar:** Contiene información de los niveles escolares que maneja el sistema.
- **Curso:** En esta tabla se almacena la relación de una categoría y el nivel escolar al que pertenece.
- **Comentario:** Registra los comentarios ingresados por el estudiante hacia un tutor en específico.

2.2.3. DIAGRAMA DE CLASES

El diagrama de clases identifica la estructura de los objetos que se utilizarán en el sistema prototipo y las asociaciones entre los objetos, como se muestra en la **Figura 2.*6***.

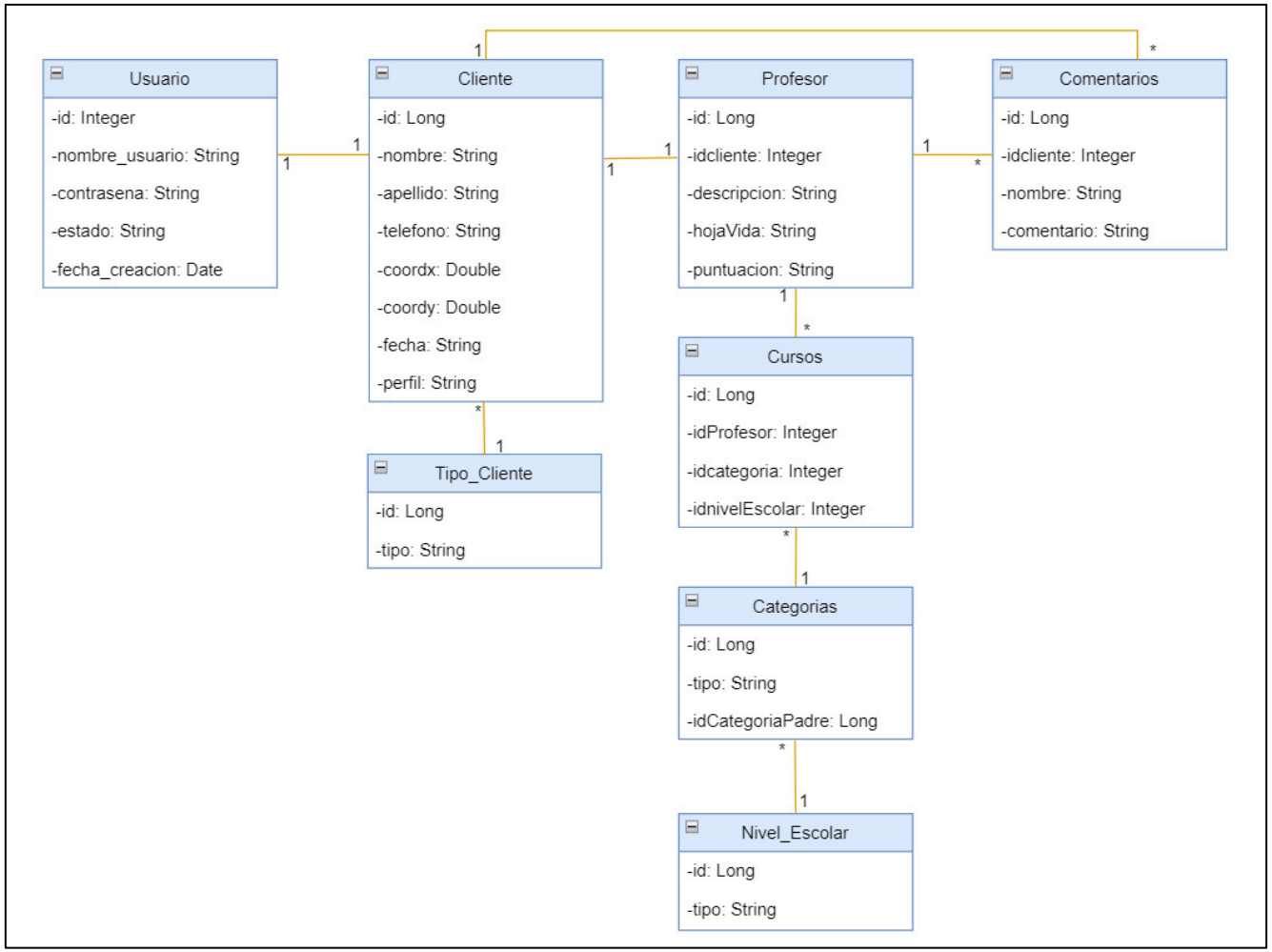

Figura 2.5. Diagrama de base de datos

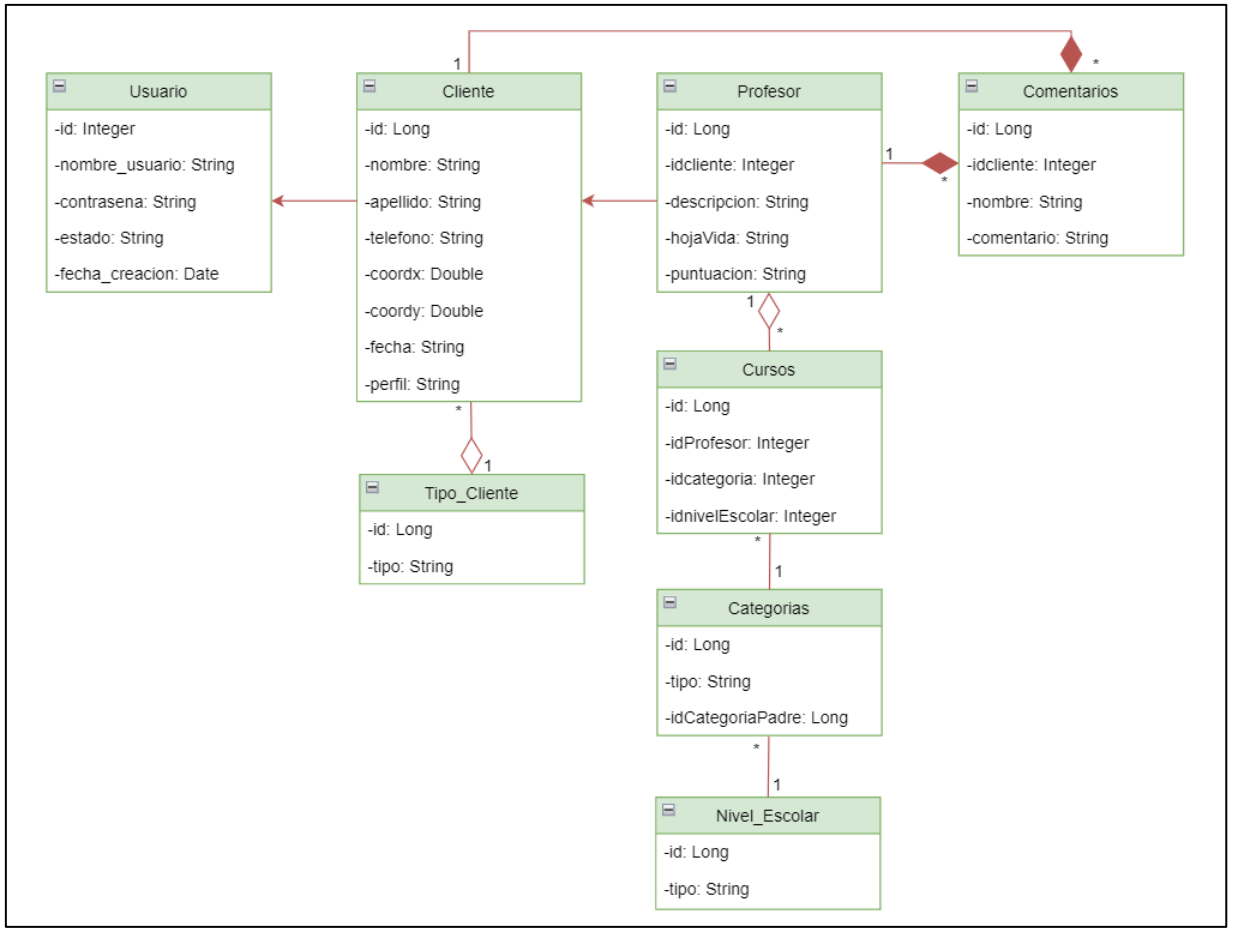

Figura 2.6. Diagrama de clases del servidor

2.2.4. DISEÑO DE LAS INTERFACES GRÁFICAS

En esta sección se presentan las plantillas que corresponden al diseño inicial de las interfaces gráficas del sistema prototipo, las plantillas fueron diseñadas en función de los requerimientos del sistema. El bosquejo que se presenta en esta sección, corresponde al diseño de la capa presentación del sistema prototipo.

La **Figura 2.*7*** corresponde a la *Activity Login*, que permite que los diferentes usuarios se autentiquen en el sistema, por medio del correo y la contraseña.

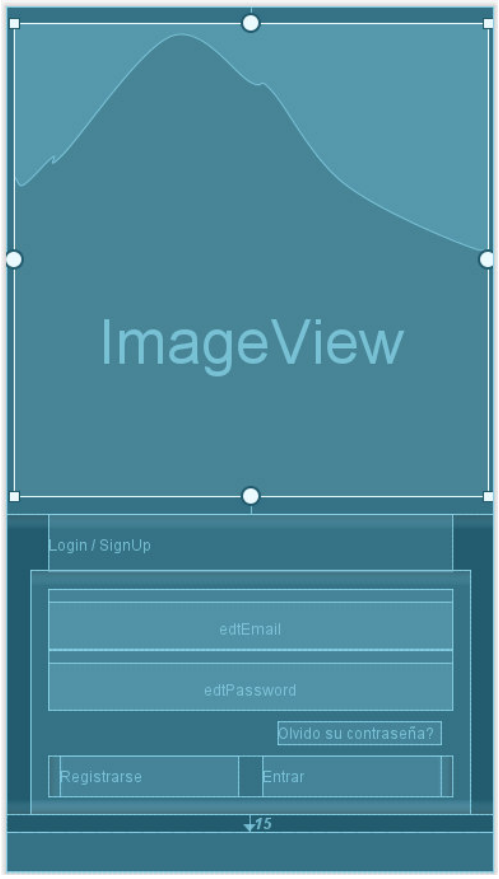

Figura 2.7. Diagrama de la *Activity* Login

En la **Figura 2.*8*** se encuentra la plantilla correspondiente a la búsqueda de tutores. Esta *Activity* permite realizar búsquedas de un tutor por medio de filtros. Permitiendo a un cliente estudiante, buscar un curso específico, de un nivel de educación especifico, dentro de un área geográfica.

Figura 2.8. Diagrama de la *Activity* para realizar búsquedas

La **Figura 2.*9*** corresponde a la *Activity* en la cual se muestra los resultados una vez realizada una búsqueda. Al desplegar el mapa, es posible navegar en él, con el fin de encontrar un tutor cercano. Además, es posible ver los tutores con mejor calificación del sistema.

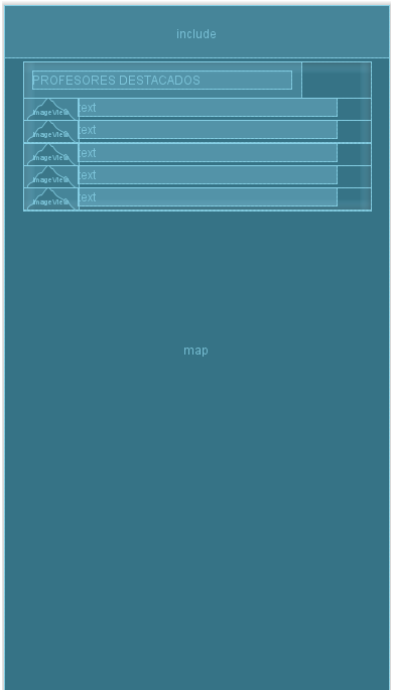

Figura 2.9. Diagrama de la *Activity* para desplegar resultados de búsquedas en el mapa

La **Figura 2.*10*** permite el registro común de clientes estudiantes y profesor. Parte de la información requerida en esta *Activity*, es el elegir el rol en el sistema.

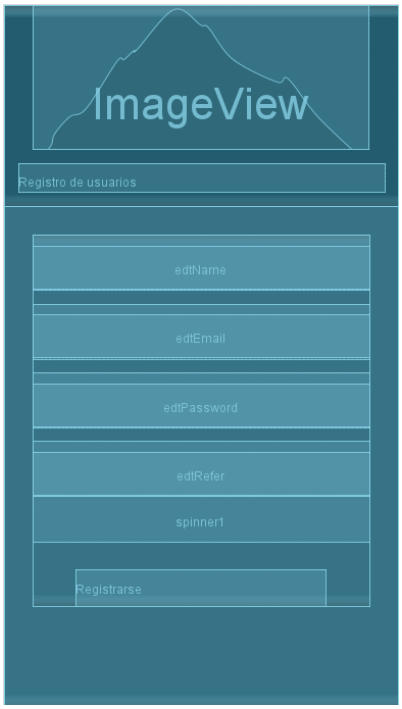

Figura 2.10. Diagrama de la *Activity* para el registro completo del tutor

La **Figura 2.*11*** corresponde al diagrama de la *Activity* de registro de tutores. Permite recolectar la información de un tutor, para completar su registro en el sistema.

Figura 2.11. Diagrama de la *Activity* para el registro completo del tutor.

En la **Figura 2.*12*** se muestra la plantilla de administración. Mediante el uso de un *spinner*, despliega los clientes registrados en el sistema para poder visualizar su perfil.

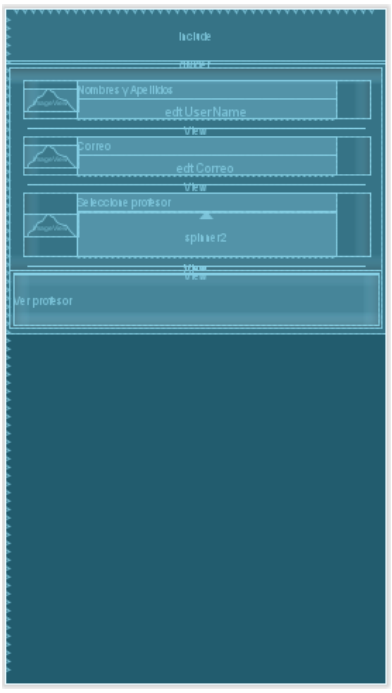

Figura 2.12. Diagrama de la *Activity* de administración para listar clientes

La **Figura 2.*13*** muestra la *Activity* para administración de clientes. El administrador podrá visualizar un perfil y administrarlo según requiera. Entre las opciones de administración se encuentran: activar cuenta, desactivar cuenta, eliminar cuenta y guardar cambios.

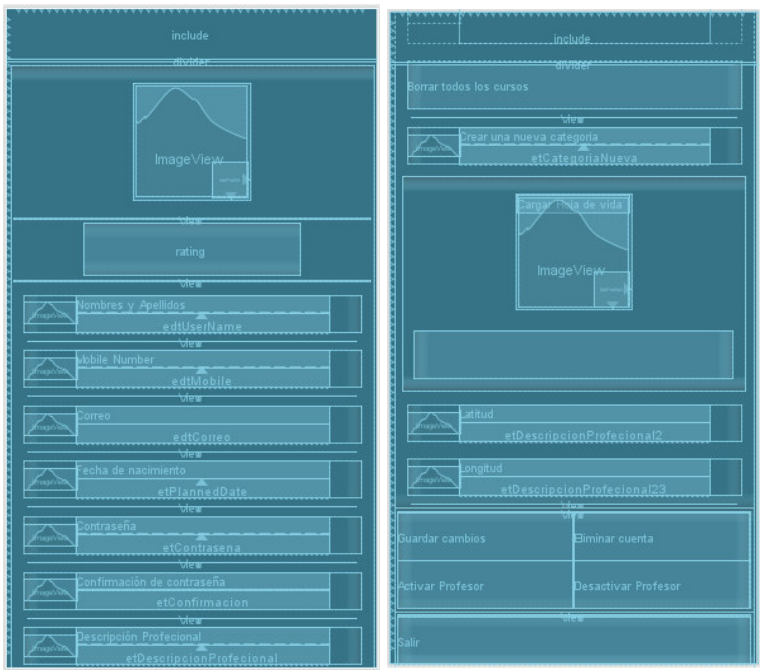

Figura 2.13. Diagrama de la *Activity* de administración de perfil

En la **Figura 2.*14*** se muestra la *Activity* para visualización de un archivo PDF.

Figura 2.14. Diagrama de la *Activity* de visualización de un archivo PDF

En la **Figura 2.*15*** se muestra la *Activity* para la administración de un perfil estudiante, por parte del propio estudiante. En la cual es posible, editar los datos y guardarlos.

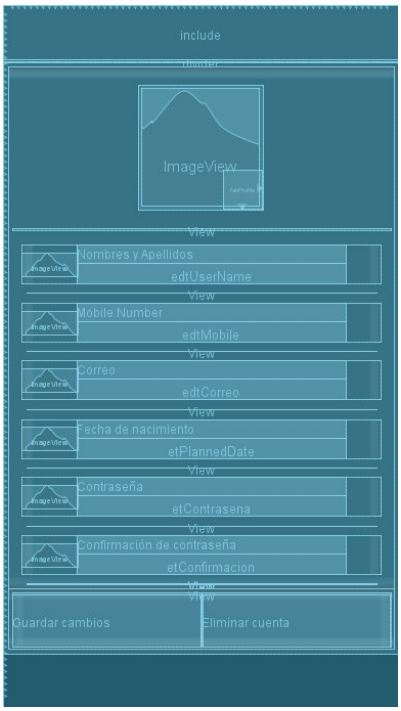

Figura 2.15. Diagrama de la *Activity* administración de un perfil estudiante, desde la perspectiva del dueño del perfil

En la **Figura 2.*16*** se muestra la *Activity* para la administración de un perfil tutor, por parte del propio tutor. En la cual es posible editar la información y enviarla al servidor para que sea almacenada en la base de datos.

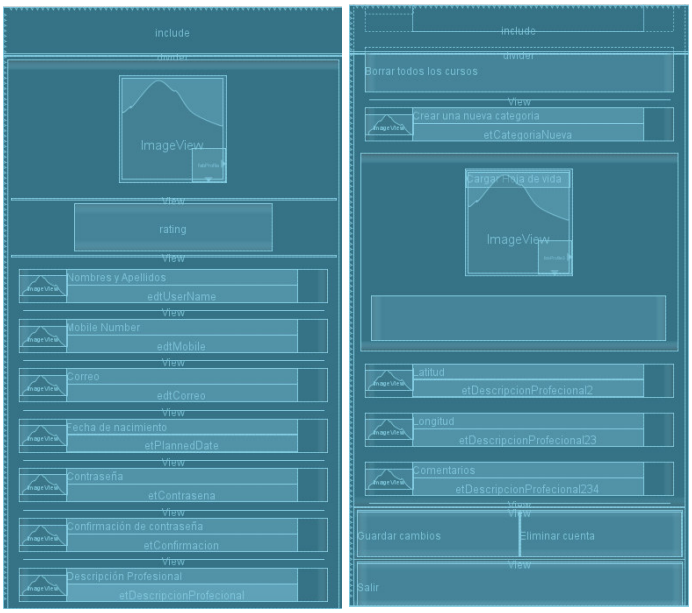

Figura 2.16. Diagrama de la *Activity* de administración de un perfil tutor, desde la perspectiva del usuario dueño del perfil

En la **Figura 2.*17*** se muestra la *Activity* de visualización del perfil tutor, por parte de un estudiante. En esta *Activity* es posible ingresar un comentario y una calificación para un tutor en específico.

Figura 2.17. Diagrama de la Activity de visualización, ingreso de comentarios y calificación en un perfil tutor

2.3. IMPLEMENTACIÓN

En este apartado se ha incluido el código más relevante desarrollado por cada una de las iteraciones realizadas. De esta manera es posible verificar el proceso de intercambio de mensajes que sigue cada una de ellas, entre la aplicación, servicio y base de datos.

2.3.1. INSTALACIÓN DE LOS AMBIENTES DESARROLLO

La instalación de los diferentes ambientes de desarrollo y herramientas para el funcionamiento del sistema prototipo son: PostgreSQL, Android Studio, Spring Boot se detallarán en el ANEXO A.

La creación de la instancia correspondiente a los servicios AWS – EC2 se encuentran detallados en el ANEXO B.

Finalmente, la creación del archivo APK y JAR para replicar la aplicación móvil y microservicios, se detallarán en el ANEXO C.

2.3.2. ITERACIÓN 1

La primera iteración se la realizó en base a los módulos de login y administración de los usuarios tutor y estudiante. En esta primera fase se procedió a la creación de la base de datos empleando el gestor de base de datos llamado PostgreSQL. Además, se creó el microservicio con la herramienta denominada Spring Boot.

Finalmente, se codificó la interfaz gráfica de la capa presentación haciendo uso de la librería Retrofit, como cliente REST, para el consumo de los microservicios.

2.3.2.1. Creación de la Base de Datos

La base de datos se realizó en base al diseño del diagrama de base de datos, procediendo a crear todas las tablas detalladas en dicho diagrama.

En primer lugar, se ejecutó la aplicación de escritorio “pgAdmin”, la cual permite ejecutar *scripts* del tipo SQL, para la elaboración de la base de datos y todas las tablas que lo componen.

En la **Figura 2.*18*** se puede apreciar la ejecución de pasos necesarios para crear una base de datos SQL y en el **Código 2.*1*** se puede visualizar una parte del *script* utilizado para la creación de la tabla `Usuario`, que guarda la información básica de usuarios que se registran en el sistema.

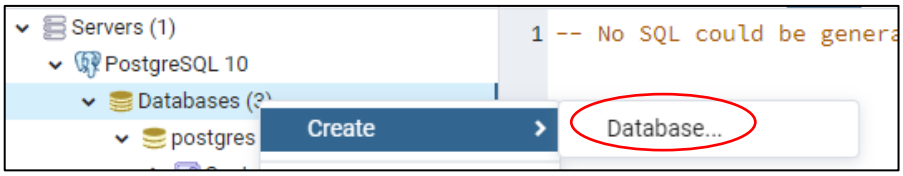

Figura 2.18. Creación de base una base de datos en el motor PostgreSQL

```
CREATE TABLE IF NOT EXISTS public.usuario
(
    id integer NOT NULL DEFAULT nextval('usuario_id_seq'::regclass),
    contrasena character varying(255) COLLATE pg_catalog."default",
    estado integer,
    fecha_creacion timestamp without time zone,
    nombre_usuario character varying(255) COLLATE pg_catalog."default",
    CONSTRAINT usuario_pkey PRIMARY KEY (id)
)

TABLESPACE pg_default;

ALTER TABLE public.usuario
    OWNER to tesis;
```

Código 2.1. Creación de la tabla Usuario

La **Figura 2.*19*** muestra todas las tablas que se encuentran en la base de datos.

Figura 2.19. Tablas disponibles en la base de datos

2.3.2.2. Login del Sistema

En primer lugar, se procedió a detallar las direcciones y los puertos desde la aplicación por cada microservicio como se muestra en el **Código 2.*2***, con la finalidad de que al llamar al cliente REST, usando la librería Retrofit, tenga la información necesaria para realizar las peticiones.

```
public static String REST_URL = "http://18.223.151.35:8080/";
public static String REST_URL2 = "http://18.223.151.35:8081/";
```

Código 2.2. Creamos las URL de conexión al servidor

Se crea el objeto de tipo Retrofit para hacer uso del cliente REST, y se realizan las configuraciones para permitir las conexiones HTTP, usando el método `getOkHttpClient()`. En donde, se detallan los parámetros de conexión como el tiempo límite de lectura y de escritura como se muestra en el **Código 2.*3***.

```
//Creo el objeto de tipo Retrofit; adapta la interfaz de java a HTTP
//Variable para mapear interfaces entre HTTP y Java
private static Retrofit retrofit = null;
//Metodo que devuelve objeto OkHttpClient
public static OkHttpClient getOkHttpClient() {
    // Objeto para imprimir logs sobre el metodo HTTP
    HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
    //Setea el nivel sobre el Body
    interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
    // Crea objeto okHttpClient
    // Crea una instancia con los parametros
    OkHttpClient okHttpClient = new OkHttpClient.Builder()
            .addInterceptor(interceptor)
            // Tiempo limite para establecer la conexion
            .connectTimeout( timeout: 100, TimeUnit.SECONDS)
            // Se setea un tiempo de lectura
            .readTimeout( timeout: 100, TimeUnit.SECONDS)
            // Se setea un tiempo de escrituras
            .writeTimeout( timeout: 100, TimeUnit.SECONDS)
            // Crea el objeto con las caracteristicas antes detalladas
            .build();
    // Devuelvo el objeto
    return okHttpClient;
}
```

Código 2.3. Configuración de los parámetros de conexión

Una vez creado el objeto para el cliente REST y las configuraciones necesarias para la conexión HTTP, es necesario crear el objeto del tipo Retrofit que hará uso de la URL, el cliente, las características de serialización y deserialización. A partir de este método se realizará la conexión al servicio, como se muestra en el **Código 2.*4***.

```
public static Retrofit getClient() {
    //Verifica si el atributo es nulo
    if (retrofit==null) {
        // Crea un objeto JSON; con thread safe (Cada hilo accede al objeto sin que
        // otros hilos intercedan)
        // GsonBuilder(): configura la instancia
        // setLenient: Configuraciones del JSON
        //create(): entrega el objeto construido para trabajar
        Gson gson = new GsonBuilder().setLenient().create();
        //Creo el objeto de tipo retrofit para hacer uso de metodos HTTPS
        //Builder(): Configuracion del objeto
        retrofit = new Retrofit.Builder()
                // Define la URL sobre la cual se va a trabajar
                .baseUrl(Constant.REST_URL)
                // Pasamos el cliente para metodos
                .client(getOkHttpClient())
                // Añade objeto para caracteristicas de serialziacion y des-serialziacion
                //Crea una instancia con el objeto GSON configurado en la libreria google
                .addConverterFactory(GsonConverterFactory.create(gson))
                // Construye el objeto
                .build();
    }
    return retrofit;
}
```

Código 2.4. Objeto de tipo Retrofit para realizar las peticiones al servicio

Finalmente, dentro del servicio se detallará la conexión a la base de datos en el archivo "Aplication.yaml". El archivo de configuración contiene la URL, que permite saber en donde se encontrará alojada la base de datos, debido a que estará en la misma instancia EC2 se la describe como localhost, se debe agregar el puerto de conexión y el nombre de la base de datos. Además, se debe especificar el *driver* correspondiente a PostgreSQL, así como el usuario y contraseña para realizar la autenticación, como se muestra en el **Código 2.*5***.

```
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/tutoriopolis
    username: tesis
    password: tesis*1234
    driver-class-name: org.postgresql.Driver
```

Código 2.5. Conexión a la base de datos, desde el servicio

Se hará uso de un *token* de verificación, con la finalidad de que las peticiones que lleguen al servicio, sean realizadas desde un usuario válido. Para esto se ejecuta la función `getToken()`, la misma que realizará la llamada hacia el servidor para validar ser un usuario de la aplicación. En caso de una respuesta exitosa se ejecuta el método `loginUser()`, el cual envía los parámetros de *token*, correo electrónico y contraseña para su autenticación en el sistema, como se muestra en el **Código 2.*6***.

```
// Realiza una llamda al servidor para realizar el login y ejecuta loginUser() con
//los datos obtenidos en los editText
//En caso de error ejecuta hideProgressDialog();
public void getToken()
{
    ApiRestInterfaceMicroServer2 apiService = ClientRestMicroServer2.getClient().create(ApiRestInterfaceMicroServer2.class);
    Call<String> tokencall = apiService.login2( authorization: "Basic cm9vdDpwcnVlYmE=");
    tokencall.enqueue(new retrofit2.Callback<String>() {
        @Override
        public void onResponse(Call<String> call, retrofit2.Response<String> response) {
            try {
                if (response.body() != null) {
                    String  tokenHash = response.body().toString();
                    loginUser(tokenHash ,edtEmail.getText().toString(),edtPassword.getText().toString());
                }
            } catch (Exception e) {
                hideProgressDialog();
                e.printStackTrace();
            }
        }
        @Override
        public void onFailure(Call<String> call, Throwable t) {
            int a =3;
            hideProgressDialog();
        }
    });
}
```

Código 2.6. Validación de usuario y llamada al método de autenticación

Una vez la petición de validación del *token* llega al servidor, se realiza la verificación mediante la variable “log” del tipo *boolean*. Si se aprueba la validación del *token* se dará paso a la búsqueda del cliente por medio del usuario y contraseña, el usuario se almacena en la lista “listaUsuario”. Al obtener el usuario se realizará una nueva consulta a la base de datos del cliente con la finalidad de obtener sus atributos y se envía la respuesta hacia el cliente, como se muestra en el **Código 2.*7***. Las consultas a la base de datos, se las realiza mediante *hibernate.*

```java
@PostMapping("/entrar")
public ResponseEntity<Object> entrar(
        @RequestParam("tokenID") String  tokenID,
        @RequestParam("email") String  email,
        @RequestParam("passw") String  passw) {

    Map<String, Object> respuesta = new ConcurrentHashMap<>();

    try {

            boolean log = false;

            try {
                Jwts.parser().setSigningKey(DatatypeConverter.parseBase64Binary(Constantes.getSecretPassword()))
                        .parseClaimsJws(tokenID).getBody();
                log = true;
            } catch ( Exception e) {
                log = false;
            }

            if(log){

                List<Usuario> listaUsuario = usuarioRepository.login(email,passw);
                if(listaUsuario.size()>0)
                {
                    Usuario usuario = listaUsuario.get(0);
                    List<Cliente> cliente = null ;
                    try {
                        cliente = clienteRepository.findByUsuario(usuario);
                    }catch (Exception e) {

                    }

                    respuesta.put("respuesta", 1);
                    respuesta.put("id", usuario.getId());
                    respuesta.put("tipo", cliente.get(0).getTipocliente_id().getTipo());
                    respuesta.put("nombre", cliente.get(0).getNombre());
                    respuesta.put("email", usuario.getNombre_usuario());
                }
                else
                respuesta.put("respuesta", 0);
            }
        else
        {   respuesta.put("respuesta", 0);}
    }catch (Exception e) {   respuesta.put("respuesta", 0);}

    return ResponseEntity.ok(respuesta);
}
```

Código 2.7. Proceso de autenticación en el servidor

Una vez finalizado las configuraciones de conexión y el servicio para el módulo *login*, se procedió a crear la interfaz gráfica para la interacción con el cliente. En la **Figura 2.*20***, se presenta el diseño de la *Activity* para la autenticación.

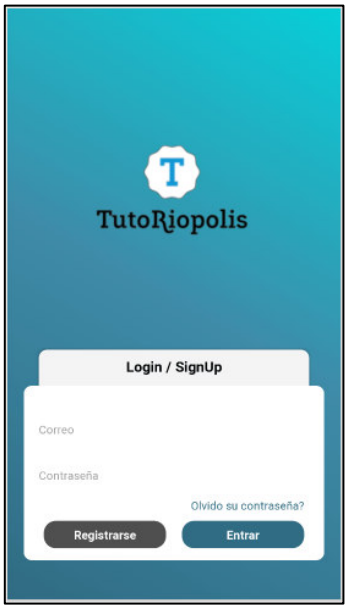

Figura 2.20. Diseño de la interfaz Login

2.3.2.3. Validar Perfil del Usuario Tutor

Para activar un perfil de usuario tutor, es necesario ingresar en la interfaz de administración, en la que se desplegarán los clientes registrados en el sistema. Al iniciar la *Activity* de administración, se hará el llamado al método `cargarDatosAlIniciar()`, que se encuentra a su vez alojada en el método `onCreate()`. El método `cargarDatosAlIniciar()`, realiza la validación del *token*, si es aprobado ejecuta dos métodos adicionales `cargarDatosAdmin()` y `cargarCliente()`, como se muestra en el **Código 2.*8***.

```
public String cargarDatosAlIniciar()
{
    // Validacion de usuario en el sistema
    ApiRestInterface apiService = ClientRest.getClient2().create(ApiRestInterface.class);
    tokenHash="";
    Call<String> tokencall = apiService.login2( authorization: "Basic cm9vdDpwcnVlYmE=");
    tokencall.enqueue(new retrofit2.Callback<String>() {
        //En caso de respuesta exitosa realiza la carga de herramientas visuales
        @Override
        public void onResponse(Call<String> call, retrofit2.Response<String> response) {
            try {
                if (response.body() != null) {
                    tokenHash = response.body().toString();
                    cargarDatosAdmin(tokenHash);
                    cargarClientes(tokenHash);
                }
                // En caso de error capturo excepcion
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        // En caso de error de validacion
        @Override
        public void onFailure(Call<String> call, Throwable t) {
            int a =3;
        }
    });
    return token;
}
```

Código 2.8. Proceso de carga de datos en la interfaz de administración

En la **Tabla 2.*9*** se describe cada uno de estos métodos, que permiten cargar la información recuperada del servidor, en los objetos visuales. Cada uno de estos métodos realiza la verificación del *token* que permite comprobar que la petición que se envía al servicio, es una petición realizada desde un cliente válido.

Tabla 2.9. Métodos utilizados para mostrar los datos recuperados del servicio, en los objetos visuales del *Activity*

MÉTODO	DESCRIPCIÓN
cargarDatosAdmin	Realiza la validación del *token*, como verificación de petición de usuario válido al servicio. Para ejecutar el método es necesario agregar el *endpoint* que tiene como atributos tokenID e ID. Si se procesa la petición en el servidor, recibe un mensaje de vuelta y verifica si el *body* del mensaje tiene contenido. En caso de que esta verificación sea diferente de *null*, extrae los valores de "nombre" y "nombre_usuario" y los muestra en pantalla mediante los elementos visuales "edtName" y "edtCorreo".
cargarCliente	Realiza la verificación del *token* y realiza la llamada al servidor mediante el *endpoint*, si la petición es válida, revisa el *body* del mensaje y recupera la lista de clientes, recorre la lista mediante el uso de un lazo "for" y los agrega a la herramienta visual mediante el método `getAdapter()`.

Cada una de las funciones realiza una petición a la base de datos, mediante un *endpoint* homólogo en el servicio. En el caso del método `cargarDatosAdmin()`, hace uso del *endpoint "*uri/getCliente", que realiza la verificación del "tokenID", si es una verificación exitosa, realiza una consulta a la tabla "Usuario" por medio del identificador único del cliente, al encontrarlo se almacena en la variable "user". Posteriormente, se realiza una consulta adicional a la tabla cliente mediante la función `findByUser()`, recuperando el usuario en la variable "client". Si la consulta se realiza con éxito, se envía la respuesta al cliente, si esta no se realiza se envía un 0, como se muestra en el **Código 2.*9***.

```
@PostMapping("/getCliente")
public ResponseEntity<Object> getEstudiante(
        @RequestParam("tokenID") String  tokenID,
        @RequestParam("id") String  id) {

    Map<String, Object> respuesta = new ConcurrentHashMap<>();
    List<Cliente> listaClientes = new ArrayList<>();
    try {
            // Verificacion del Token
            boolean log = false;
            try {
                Jwts.parser().setSigningKey(DatatypeConverter
                        .parseBase64Binary(Constantes.getSecretPassword()))
                        .parseClaimsJws(tokenID).getBody();
                log = true;
            } catch ( Exception e) {
                log = false;
            }

            if(log){
                //Consulta a la base de datos
                Usuario user = usuarioRepository.getByID(Integer.parseInt(id)).get(0);
                Cliente client = clienteRepository.findByUsuario(user).get(0);
                respuesta.put("respuesta", 1);
                respuesta.put("contrasena", user.getContrasena());
                respuesta.put("nombre_usuario", user.getNombre_usuario());
                respuesta.put("nombre", client.getNombre());
                respuesta.put("apellido", client.getApellido());
                respuesta.put("telefono", client.getTelefono());
                respuesta.put("fecha", client.getFecha());
                respuesta.put("perfil", client.getPerfil());

            }
            else
            {respuesta.put("respuesta", 0);}


        }catch (Exception e) {
            respuesta.put("respuesta", 0);
        }

       return ResponseEntity.ok(respuesta);
}
```

Código 2.9. Obtener el cliente Administrador en el servicio

En la **Figura 2.*21*** se presenta la interfaz de administración, para seleccionar un usuario y visualizar un perfil seleccionado en el *spinner*.

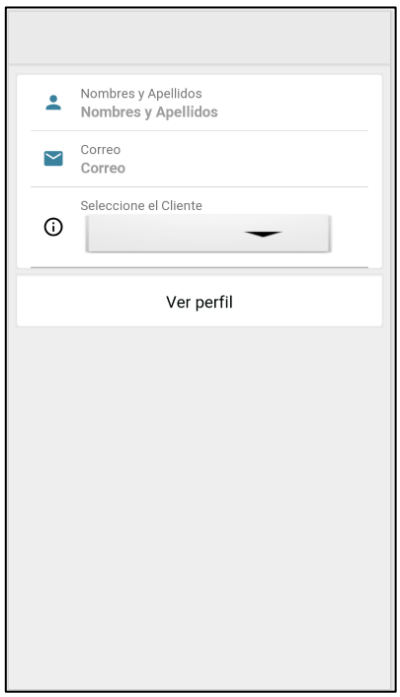

Figura 2.21. Interfaz de Administración selección de Usuarios Registrados

Al ingresar a un perfil de tutor, se visualiza los botones de administración, los cuales son: "Activar" y "Desactivar". Por defecto, todo tutor registrado por primera vez, se encuentra desactivado, es decir, con el estado "0".

Para activar una cuenta tutor se realiza el llamado a la función `activarProfesor()`, y se envía como parámetro de entrada el valor de "1", como se muestra en el **Código 2.*10***.

```
public void ActivarProfesor(View v)
{

    llamadaAlCambioDeEstado(1);
}
```

Código 2.10. Llamada al método para activar un tutor

El método `llamadaAlCambioDeEstado()`, verifica el token y ejecuta `cambiarEstadoUser()`. En donde, se realiza la petición al servidor mediante el *endpoint "updateStatusUser"*, como se muestra en el **Código 2.*11***. El servidor realiza la verificación del *token*, recupera los usuarios de la base de datos y realiza la búsqueda mediante el identificador único, si lo encuentra sobrescribe el valor del estado y envía "1" al servidor para confirmar el cambio, en caso de error, envía "0" como se muestra en el **Código 2.*12***.

```
@FormUrlEncoded
@POST("/api/updateStatusUser")
Call<String> updateStatusUser(
        @Field("tokenID") String tokenID,
        @Field("id") String id,
        @Field("estado") String estado);
```

Código 2.11. *Endpoint* utilizado para realizar la conexión al servicio y cambiar el estado de un tutor

```
@PostMapping("/updateStatusUser")
public ResponseEntity<String> updateStatusUser(
        @RequestParam("tokenID") String  tokenID,
        @RequestParam("id") String  id,
        @RequestParam("estado") String  estado) {

    String respuesta = "0";
    // Verificacion del token
    try {
            boolean log = false;

            try {
                Jwts.parser().setSigningKey(DatatypeConverter
                        .parseBase64Binary(Constantes.getSecretPassword()))
                        .parseClaimsJws(tokenID).getBody();
                log = true;
            } catch ( Exception e) {
                log = false;
            }
            // Se recuperan los usuarios
            //y se realiza la busqueda por medio del Id
            if(log){

                List<Usuario> listaUsuario = usuarioRepository.findAll();
                Usuario usuario= null;
                for(int i=0;i<listaUsuario.size();i++)
                {
                    if(id.equalsIgnoreCase(listaUsuario.get(i).getId()+""))
                    {
                        usuario =listaUsuario.get(i);
                    }
                }
                // En el caso de encontrar al usuario se cambia el estado,
                // se lo guarda y se retorna 1 al cliente
                if(usuario!=null) {
                usuario.setEstado(Integer.parseInt(estado));
                usuarioRepository.save(usuario);
                }

                respuesta="1";
            }
        // En caso de error se retorna 0
        }catch (Exception e) {
         respuesta="0";
       }


    return ResponseEntity.ok(respuesta);
}
```

Código 2.12. Actualización del estado para activar un perfil tutor en el servidor

2.3.2.4. Eliminar cuenta del usuario Tutor

Para eliminar una cuenta del usuario tutor, es necesario que se ejecute el método `deleteProfesorRequest()`, haciendo el uso del *endpoint "/api/*deleteProfesor", que usa los atributos de "token" y "id". Si la respuesta del servidor es un "1", reescribe el archivo

“login.data” con el número “0”, recupera la información del administrador mediante los “extra” para almacenar valores del *intent* e inicia la actividad como se muestra en el **Código 2.*13***.

```
public void deleteProfesorRequest(String token)
{
    // Verificacion de token
    ApiRestInterface apiService = ClientRest.getClient().create(ApiRestInterface.class);
    Call<UpdateDTO> tokencall = apiService.deleteProfesor(token,id);
    tokencall.enqueue(new retrofit2.Callback<UpdateDTO>() {
        @Override
        public void onResponse(Call<UpdateDTO> call, retrofit2.Response<UpdateDTO> response) {
            // En caso de respuesta exitosa, se revisa el cuerpo del mensaje
            try {
                if (response.body() != null) {
                    UpdateDTO listado = response.body();
                    if(listado !=null && listado.respuesta.equalsIgnoreCase( anotherString: "1"))
                    {
                        try {
                            // Se crea el objeto para la lectura del archivo
                            OutputStreamWriter fout =
                                    new OutputStreamWriter(
                                            openFileOutput( name: "login.data", Context.MODE_PRIVATE));
                            // Se edita su contenido por cero
                            fout.write( str: "0");
                            // Se cierra el archivo
                            fout.close();
                            // Se crea el objeto para realizar abrir la actividad AdminActivity
                            Intent intent1 = new Intent( packageContext: InformacionProfesionalAdmin
                                    .this, AdminActivity.class);
                            // Se almacenan los datos en el intent para ser enviados a la actividad
                            intent1.putExtra( name: "id", idAdmin);
                            intent1.putExtra( name: "tipo", tipo);
                            // Se inicia la actividad
                            startActivity(intent1);
                        }catch (Exception e){}
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        @Override
        public void onFailure(Call<UpdateDTO> call, Throwable t) { int a =3; }

    });
}
```

Código 2.13. Petición para eliminar un registro de tutor desde el perfil de administración

La petición que se realiza en el servicio mediante el *endpoint* “deleteProfesor”, inicia con la verificación del *token*, una vez que es aprobado, realiza la consulta a la tabla “usuario” por medio del identificador único, enviado desde cliente. Al encontrar el usuario se realiza una nueva consulta de tipo `findByUsuario()`. Con la finalidad de eliminar todos los registros de un tutor, se realiza búsquedas en las tablas de profesores y curso. Mediante el uso de

un bucle `for`, se recorre las listas obteniendo los resultados en “listaCursos” y “lprofesores”, y se ejecuta el *delete*, siempre y cuando el identificador único enviado desde el cliente, coincida con el identificador en las tablas. En la respuesta al cliente se incluye el numero “1” si fue exitosa o “0” en caso de error.

Interfaz creada para realizar la administración de los usuarios tutor como se muestra en la **Figura 2.*22***.

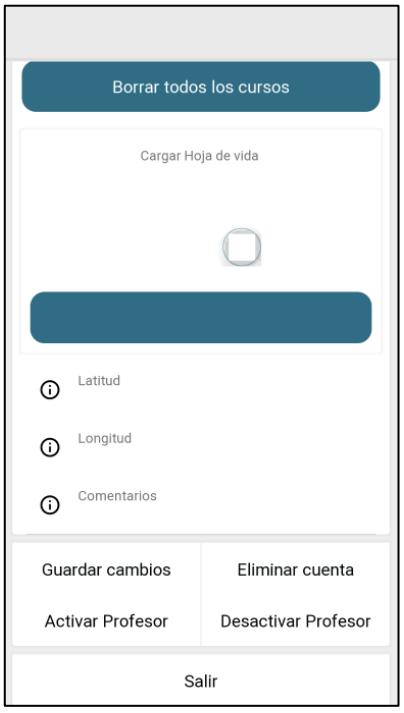

Figura 2.22. Botones de administración del usuario Tutor

2.3.3. ITERACIÓN 2

En esta iteración se llevó a cabo el registro de usuarios estudiante y tutor. Ambos registros cuentan con una *Activity* de registro común, es necesario elegir el tipo rol que se desea tener dentro del sistema. Una vez pulsado el botón de registrar, el usuario es redirigido a las diferentes *Activities* para continuar con su registro.

2.3.3.1. Registro de Usuario

Una vez se despliega la actividad, se carga el objeto visual *spinner* con las opciones de “Profesor” y “Estudiante”. El usuario llena los campos de la *Activity* y el botón “Registrarse”, valida la información mediante la llamada del método `validateForm()`. Si la validación de los campos es correcta ejecuta la función getToken(), que revisa si el *token* es correcto, y si lo es, recupera los datos ingresados por el usuario y ejecuta `registerUser()` como se muestra en el Código 2.14. Este método hace uso del *endpoint “/*api/registrarcliente” y

ejecuta una petición al servidor para registrar al usuario en la tabla cliente de la base de datos, como se muestra en el Código 2.15.

Si el cliente recibe una respuesta exitosa del servicio, ejecuta `loginUser()`, y escribe en el archivo “login.data” los atributos del cliente “Id”, “Tipo” y “Nombre” para mantener la sesión activa antes de redirigirlo al *Activity* de “Búsqueda” si es un cliente estudiante o “ProfileProfesor” si es un profesor, como se muestra en el Código 2.16.

```
// Envia el Token
// Recupera los datos de los objetos visuales
// Ejecuta el metodo registerUser en caso de que la verificacion
// del token sea exitosa
public void getToken()
{
    ApiRestInterface apiService = ClientRest.getClient2().create(ApiRestInterface.class);
    Call<String> tokencall = apiService.login2( authorization: "Basic cm9vdDpwcnVlYmE=");
    tokencall.enqueue(new retrofit2.Callback<String>() {
        @Override
        public void onResponse(Call<String> call, retrofit2.Response<String> response) {
            try {
                // Verifica el cuerpo del mensaje de respuesta
                if (response.body() != null) {
                    // Obtiene los datos de los elementos visuales de la Activity
                    // y los envia como parametros de entrada al metodo registerUser
                    String  tokenHash = response.body().toString();
                    registerUser(tokenHash,edtName.getText()
                            .toString(),edtEmail.getText()
                            .toString(),edtPassword.getText()
                            .toString(),dropdown.getSelectedItem().toString());
                }
                // Captura le excepcion y ejecuta
                // hideProgressDialog
            } catch (Exception e) {
                hideProgressDialog();
                e.printStackTrace();
            }
        }
        // Si
        @Override
        public void onFailure(Call<String> call, Throwable t) {
            int a =3;
            hideProgressDialog();
        }
    });
}
```

Código 2.14. Método `getToken()`, para verificar que la petición hacia el servicio provenga de un usuario válido

```java
// Envia los parametros capturados en le layout para registrar al usuario
// Ejecuta e metodo loginUser() y como paametros de entrada envia
// token, email y password para una vez la respuesta sea exitosa
// mantener la autenticacion del usuario
public void registerUser(String token,String nombre,String email,String password,String tipo)
{
    ApiRestInterface apiService = ClientRest.getClient().create(ApiRestInterface.class);
    Call<String> tokencall = apiService.registrarCliente(token,nombre,email,password,tipo);
    tokencall.enqueue(new retrofit2.Callback<String>() {
        @Override
        public void onResponse(Call<String> call, retrofit2.Response<String> response) {
            try {
                if (response.body() != null) {
                    String listado = response.body();
                    if(listado.equalsIgnoreCase( anotherString: "1"))
                    {
                        loginUser(token,email,password);
                    }
                    // En caso error en el registro muetra un toast con el mensaje
                    //ejecuta el hideProgressDialog()
                    else{
                        hideProgressDialog();
                        Toast.makeText( context: SignUpActivity.this
                                , text: "Error Inesperado,Intente más tarde"
                                , Toast.LENGTH_SHORT).show();
                    }
                }
            } catch (Exception e) {
                hideProgressDialog();
                e.printStackTrace();
            }
        }
        @Override
        public void onFailure(Call<String> call, Throwable t) {
            int a =3;
            hideProgressDialog();
        }
    });
}
```

Código 2.15. Método `registerUsuario()`, para enviar la petición de registro del cliente al servidor

```java
public void loginUser(String token,String email,String password)
{
    ApiRestInterface apiService = ClientRest.getClient().create(ApiRestInterface.class);
    Call<ClienteDTO> tokencall = apiService.loginCliente(token,email, Hash.sha1(password));
    tokencall.enqueue(new retrofit2.Callback<ClienteDTO>() {
        @Override
        public void onResponse(Call<ClienteDTO> call, retrofit2.Response<ClienteDTO> response) {
            try {
                if (response.body() != null) {
                    ClienteDTO listado = response.body();
                    if(listado.getRespuesta().equalsIgnoreCase( anotherString: "1"))
                    {
                        try
                        {
                            // Perite hacer uso del archivo login.data
                            // Mediante la creacion del objeto tipo OutPutStreamWriter
                            OutputStreamWriter fout=
                                    new OutputStreamWriter(
                                            openFileOutput( name: "login.data"
                                                    , Context.MODE_PRIVATE));
                            // Escribe los datos de autenticacion
                            //Id
                            //Tipo
                            //Nombre
                            fout.write( str: listado.getId()+"|"+listado.getTipo()
                                    + "|"+listado.getNombre());
                            fout.close();
                            // Ejecuta la funcion irMain
                            irMain(listado.getId(),listado.getTipo(),listado.getNombre());
```

Código 2.16. Método `loginUser()`, para mantener la sesión activa del usuario

En el lado del servidor al realizar la petición por medio del *endoint* "/api/registrarCliente" y registra al cliente en la base de datos y envía la respuesta de "ok" si el registro fue exitoso, como se muestra en el **Código 2.*17***.

```
@PostMapping("/registrarCliente")
public ResponseEntity<String> registrarCliente(
        @RequestParam("tokenID") String  tokenID,
        @RequestParam("nombre") String  nombre,
        @RequestParam("email") String  email,
        @RequestParam("passw") String  passw,
        @RequestParam("tipo") String  tipo) {

    String respuesta = "0";

    try {
            boolean log = false;

            try {
                Jwts.parser().setSigningKey(DatatypeConverter.
                        parseBase64Binary(Constantes.getSecretPassword()))
                        .parseClaimsJws(tokenID).getBody();
                log = true;
            } catch ( Exception e) {
                log = false;
            }

            if(log){

                Usuario usuario = new Usuario();
                usuario.setNombre_usuario(email);
                usuario.setFecha_creacion(new Date());
                // Si es un usuario profesor
                // se le da el estado de 2 (DESACTIVADO)
                if(tipo.equalsIgnoreCase("PROFESOR"))
                usuario.setEstado(2);
                else
                usuario.setEstado(1);
                usuario.setContrasena(Hash.sha1(passw));
                Usuario id =usuarioRepository.save(usuario);
                Cliente cliente = new Cliente();
                cliente.setNombre(nombre);
                cliente.setApellido(".");
                cliente.setCoordx(0.0);
                cliente.setCoordy(0.0);
                cliente.setTelefono("09");
                cliente.setUsuario_id(id);
                cliente.setFecha("01/01/1900");
                cliente.setPerfil("1");
                List<Tipo_Cliente> tipo_Cliente = tipoClienteRepository.findAll();
                // Se verifica el tipo cliente para ser guardado
                for(int i =0 ; i<tipo_Cliente.size();i++ )
                {
                    if(tipo_Cliente.get(i).getTipo().equalsIgnoreCase(tipo))
                    cliente.setTipocliente_id(tipo_Cliente.get(i));


                }
                clienteRepository.save(cliente);
                if(tipo.equalsIgnoreCase("PROFESOR")) {
                    Profesor profesor = new Profesor();
                    profesor.setDescripcion("Descripción profecional");
                    profesor.setIdcliente(Integer.parseInt(cliente.getId()+""));
                    profesor.setFotohojavida("null");
                    profesor.setHojavida("-1");

                    profesorRepository.save(profesor);

                }
                sendSimpleMail(email,nombre);
                respuesta="1";
            }

        }catch (Exception e) {
         respuesta="0";
        }
    return ResponseEntity.ok(respuesta);
}
```

Código 2.17. Envío de la información hacia la base de datos para realizar el registro de un usuario

La interfaz utilizada para el proceso se muestra en la Figura 2.23.

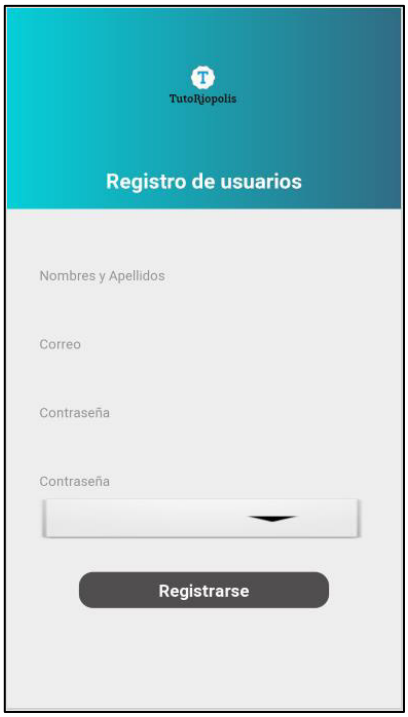

Figura 2.23. Activity para registro común de los usuarios Estudiante y Tutor

El usuario registrado como estudiante finaliza el proceso una vez pulsado el botón "Registrarse", en caso de que el rol elegido sea del tipo profesor, deberá completar el registro mediante la redirección hacia una nueva *Activity.*

2.3.4. ITERACIÓN 3

La iteración tres corresponde a las búsquedas de tutores, basada en los filtros y la ubicación. Las búsquedas las pueden realizar tanto usuarios estudiantes registrados, como no registrados. Si el usuario no se encuentra registrado, no podrá visualizar los resultados hasta que realice el proceso de registro y en el caso de que intente acceder a un perfil de un usuario tutor, será redirigido al *Activity* de Login.

2.3.4.1. Búsqueda de Tutores

La búsqueda de tutores está basada en el nivel de estudio de una materia en concreto y se la realiza mediante el uso de dos *spinner*. El primer *spinner*, permite elegir el nivel académico de la materia que se desea buscar y el segundo *spinner* en base a la elección del primero, muestra las opciones de materias disponibles para dicho nivel académico. Para esta actividad se realizan dos consultas a la base de datos, mediante la función `getToken()`, que se encuentra dentro del método `onCreate()`, que se ejecuta al iniciar la actividad para que los objetos visuales muestren la información en pantalla, como se

muestra en el Código 2.18. Para obtener la información de Niveles y Categorías, es necesario realizar una consulta en la base de datos por medio del servicio. Los *endpoint* para realizar estas solicitudes son "/api/listarNiveles" y "/api/listarCategorias", como se muestra en el Código 2.19.

```
public String getToken()
{
    //Inicia la llamda al servicio
    ApiRestInterfaceMicroServer2 apiService = ClientRestMicroServer2.getClient()
            .create(ApiRestInterfaceMicroServer2.class);
    tokenHash="";
    Call<String> tokencall = apiService.login2( authorization: "Basic cm9vdDpwcnVlYmE=");
    tokencall.enqueue(new retrofit2.Callback<String>() {
        @Override
        public void onResponse(Call<String> call, retrofit2.Response<String> response) {
            try {
                // Llamada de metodos para obtener contenido de spinners
                if (response.body() != null) {
                    tokenHash = response.body().toString();
                    cargarNiveles(tokenHash);
                    cargarCategorias(tokenHash);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        // En caso de falla
        @Override
        public void onFailure(Call<String> call, Throwable t) { int a =3; }
    });
    return token;
}
```

Código 2.18. Llamada a las funciones `cargarNiveles()` y `cargarCategorias()` para obtener contenido de *spinners*

```
@FormUrlEncoded
// Ejecutara una peticion HTTP tipo POST, hacia la URL "/api/listarnivelescolar"
@POST("/api/listarnivelescolar")
//Field: Añade contenido al Body del metodo HTTP
Call<List<NivelesDTO>> listarnivelescolar(
        @Field("tokenID") String tokenID);

@FormUrlEncoded
// Ejecutara una peticion HTTP tipo POST, hacia la URL"/api/listarnivelescolar"
@POST("/api/listarCategorias")
        //Field: Añade contenido al Body del metodo HTTP
Call<List<CategoriasDTO>> listarCategorias(
        @Field("tokenID") String tokenID);
```

Código 2.19. *Endpoint* para enviar la petición al servicio y obtener los datos de niveles y categorías disponibles en el sistema

Al recibir las peticiones mediante los *endpoint* antes mencionados, el servidor realiza las consultas del tipo `findAll()`, a las tablas “Categorias” y “NivelEscolar”, si las consultas se realizan con éxito, la respuesta es enviada al cliente como se muestra en el **Código 2.*20*** y **Código 2.*21***.

```
@PostMapping("/listarnivelescolar")
public ResponseEntity<Object> listarnivelescolar(
        @RequestParam("tokenID") String  tokenID) {
    List<Nivel_Escolar> niveles =new ArrayList<Nivel_Escolar>();
     try {
            boolean log = false;

            try {
                // Revisa el token
                Jwts.parser().setSigningKey(DatatypeConverter
                        .parseBase64Binary(Constantes.getSecretPassword()))
                        .parseClaimsJws(tokenID).getBody();
                log = true;
            }
            catch ( Exception e)
            {
                log = false;
            }
            // Consulta de niveles a la base de datos
            //Me diante la consulta findAll()
            //que guarda todo el contenido de la tabla en la variable
            //la lista "niveles"

            if(log){
                // Consulta hacia la tabla Nivel Escolar
                  niveles = nivel_EscolarRepository.findAll();
            }
     }catch (Exception e) { }
     // Envio la respuesta al cliente
    return ResponseEntity.ok(niveles);
}
```

Código 2.20. Consulta hacia la tabla Nivel Escolar

```
@PostMapping("/listarCategorias")
public ResponseEntity<Object> listarcategorias(
        @RequestParam("tokenID") String  tokenID) {

     Map<String, Object> respuesta = new ConcurrentHashMap<>();
     List<Categorias> listaCategorias = new ArrayList<>();

     try {

            boolean log = false;
            //Revision de token
            try {
                Jwts.parser().setSigningKey(DatatypeConverter
                        .parseBase64Binary(Constantes.getSecretPassword()))
                        .parseClaimsJws(tokenID).getBody();
                log = true;
            } catch ( Exception e) {
                log = false;
            }

            if(log){
                // Consulta a la base de datos
                listaCategorias = categoriasRepository.findAll();
            }

     }catch (Exception e) {  }
    // Respuesta enviada al cliente, en caso de consulta exitosa
    return ResponseEntity.ok(listaCategorias);
}
```

Código 2.21. Consulta en la tabla Categoría

Debido a que el contenido del *spinner* de categorías depende del *spinner* de nivel escolar, es necesario extraer aquellas categorías que coinciden con el nivel escolar, para realizar este procedimiento se hace uso de un bucle `for`, que compara el identificador único del nivel escolar con el de la categoría y lo almacena, como se muestra en el Código 2.22.

```
for(int i=0 ;i<listadoCaracteristicas.size();i++)
{
    if(listadoCaracteristicas.get(i).getIdCategoriaPadre()==nivel.getId())
    {ITEMS2.add(listadoCaracteristicas.get(i));}
}
```

Código 2.22. Bucle de tipo `for` para obtener los valores del spinner Categoría, en base al spinner Niveles Escolar por medio de la comparación del identificador único

Una vez elegidas las opciones en los *spinner.* El botón “Iniciar búsqueda”, ejecutará el método `launchActivityWithSpinnerInformation()`. Esta función recupera los valores de los *spinner* y los guarda en los extras del *intent* para enviarlos a la siguiente actividad, realiza el llamado a la *activity “*MapsActivity” e inicia la misma, como se muestra en el Código 2.23.

```
public void launchMapActivitywithSpinnerInformation(View view) {
    // Recupero Valores de los spinner
    NivelesDTO ni = (NivelesDTO)dropdown.getSelectedItem();
    CategoriasDTO cate = (CategoriasDTO)dropdown2.getSelectedItem();
    // Se envia los identificadores de los spinner y se despliega MapActivity
    Intent i = new Intent(getApplicationContext(), MapsActivity.class);
    i.putExtra( name: "type",  value: "default");
    i.putExtra( name: "ni",  value: ni.id+"");
    i.putExtra( name: "cate",  value: cate.id+"");
    // Indicadores para ejecutar una nueva tarea segun el codigo
    i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
    startActivity(i);
    finish();

}
```

Código 2.23. Método para desplegar el Mapa y recuperar valores de los spinner

La interfaz para realizar la búsqueda, se muestra en la Figura 2.24.

Figura 2.24. Activity de búsqueda

Al iniciar la *Activity* "MapsActivity" y por medio del método `onCreate()`, se ejecuta el método `getToken()`, que valida el *token* y ejecuta el método `cargarClientes()`. Este método hace una petición al servicio mediante el *endpoint* "/api/listarProfesoresMap" y envía los parámetros *token*, "nivelEscolar" y "categoría", con la finalidad de obtener un listado de profesores filtrado, como se muestra en el Código 2.24.

```
@FormUrlEncoded
@POST("/api/listarprofesoresmap")
Call<List<ProfesorMapDTO>> listarprofesoresmap(
        @Field("tokenID") String tokenID,
        @Field("nivelEscolar") String nivelEscolar,
        @Field("categoria") String categoria);
```

Código 2.24. Endpoint realizar la petición de una lista de profesores al servicio

Al llegar la petición al servicio, revisa el *token*, se hace una consulta del tipo `findAll()` a las tablas "profesor" y "curso", con el resultado se realiza un filtrado para obtener una lista de profesores que cumpla con la solicitud de la búsqueda, como se muestra en el Código 2.25.

```
if(!nivelEscolar.equalsIgnoreCase("-1"))
{
    // Filtrado por nivel escolar y categorias obtengo los identificadores
    for(int j=0;j<listaCursos.size() ;j++)
    {
        if(!categoria.equalsIgnoreCase("-1"))
        {
            if(listaCursos.get(j)
                    .getIdcategoria().intValue()==Integer.parseInt(categoria))
            {
                listaIdClientes.add(listaCursos.get(j).getIdcliente());
            }
        }
        else
        {
            if(listaCursos.get(j).getNivelescolar()
                    .intValue()==Integer.parseInt(nivelEscolar))
            {
                listaIdClientes.add(listaCursos.get(j).getIdcliente());
            }
        }
    }

}

// Obtiene todos los clientes, objetos clientes
listaClientes = clienteRepository.findAll();
// Elimina todos los regostros diferentes del tipo 2, que son profesores
for(int i = 0 ;i<listaClientes.size();i++ )
{
    // Obtiene solo los clientes del tipo profesor
    if(listaClientes.get(i).getTipocliente_id().getId()!=2)
    {   listaClientes.remove(i); }
}
// Recorre la lista de clientes en funcion de la de los identificaores
// Si el id coincide
for(int i = 0 ;i<listaClientes.size();i++ )
{
    int buscar = 0;
    for(int a =0; a<listaIdClientes.size();a++)
    {
        Long id=  Long.parseLong(listaIdClientes.get(a)+"");
        if(listaClientes.get(i).getId()==id)
        {buscar = 1;}
    }
    // Si no exite coincidencia es removido
    if( (buscar==0 && !nivelEscolar.equalsIgnoreCase("-1") ) )
        {
            listaClientes.remove(i);
        }
}
```

Código 2.25. Filtrado para obtener objetos de tipo “Profesor” y posicionarlos en el Mapa

Una vez se ha recibido la respuesta, la aplicación ejecuta el método `onMapReady()`, que permite desplegar un fragmento de mapa que se basa en la posición, que obtiene del dispositivo, agrega la etiqueta “Posición Actual” y posiciona la cámara en el marcador del estudiante, como se muestra en el Código 2.26. Para dibujar los clientes profesor en el mapa, se obtiene los elementos de la lista y para cada uno de ellos se crea un objeto del tipo marcador, en el cual se envía la latitud y longitud, se añade el diseño y la etiqueta “Click para más detalles”, como se muestra en el Código 2.27.

```java
// Ejecuta el fragmento de mapa con configuracion
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    // Creo el objeto del tipo LatLng para
    // //para manejar la longitud y latitud
    LatLng sydney = new LatLng(latitude, longitude);
    // Añade escuchadores de Click al mapa
    mMap.setOnInfoWindowClickListener(this);
    // Creo el onjeto de tipo marcador
    Marker posicion;
    posicion =mMap.addMarker(new MarkerOptions()
            .position(sydney)
            // Asigna el icono de la posicion del estudiante
            // //y la etiqueta  "Pisicion Actual" donde se realiza la busqueda
            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN))
            .title("Posición actual"));
            posicion.setTag(0);
    // Se mueve al marcador la camara del mapa
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(sydney, v: 19.0f));
}
```

Código 2.26. Despliega el fragmento de mapa y posicionamiento de la cámara en la ubicación que se obtiene del dispositivo

```java
//Se creo el objeto LatLng para almanecenal las coordenadas del
//Profesor
LatLng sydney = new LatLng(profesorDTO.coordx, profesorDTO.coordy);
// Se crea el objeto del tipo "marcador"
Marker posicion;
// Se envian los datos de longitud y latitud
// El tipo de diseño del marcador
// y el titulo de la etiqueta
posicion = mMap.addMarker(new MarkerOptions()
        .position(sydney)
        .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED))
        .snippet("Click para más detalles")
        .title(profesorDTO.nombre));
posicion.setTag(profesorDTO.id);
```

Código 2.27. Marcadores para clientes Tutor

Finalmente, si se desea ver los resultados en el mapa, la aplicación detectará si el usuario estudiante se encuentra autenticado o no en el sistema. Si se encuentra autenticado, será dirigido al perfil del tutor seleccionado, si no, será dirigido a la *Activity* login, como se muestra en el Código 2.28.

```java
@Override
public void onInfoWindowClick(Marker marker) {
    int a = (Integer) marker.getTag();
    String texto = "";
    try
    {
        //Crea el objeto para leer el archivo
        BufferedReader fin =
                new BufferedReader(
                        new InputStreamReader(
                                openFileInput( name: "login.data")));
        //Lee los datos
        texto = fin.readLine();
        //Cerrar
        fin.close();
    }
    catch (Exception ex)
    {
        Log.e( tag: "Ficheros",  msg: "Error al leer fichero desde memoria interna");
    }
    // Si el texto es una cadena vacia o igual a 0
    if(texto.equalsIgnoreCase( anotherString: "") || texto.equalsIgnoreCase( anotherString: "0"))
    {
        // Si no se encuentra autenticado
        Intent intent = new Intent( packageContext: MapsActivity.this, Login2Activity.class);
        intent.putExtra( name: "register",  value: "0");
        // Inicio la actividad
        startActivity(intent);
        finish();
    }
    // Permite ver la activity del tutor seleccionado
    else
    if(a!=0) {
        Intent intent = new Intent( packageContext: MapsActivity.this
                , InformacionProfesionalActivity.class);
        intent.putExtra( name: "register",  value: "0");
        intent.putExtra( name: "id",  value: marker.getTag()+"");
        startActivity(intent);
        finish();
    }
}
```

Código 2.28. Método de verificación de autenticación del usuario estudiante y redirección hacia las *Activities* del perfil Tutor y login

En la Figura 2.25 se muestra la interfaz gráfica destinada a la visualización en el mapa de los resultados de los tutores filtrados.

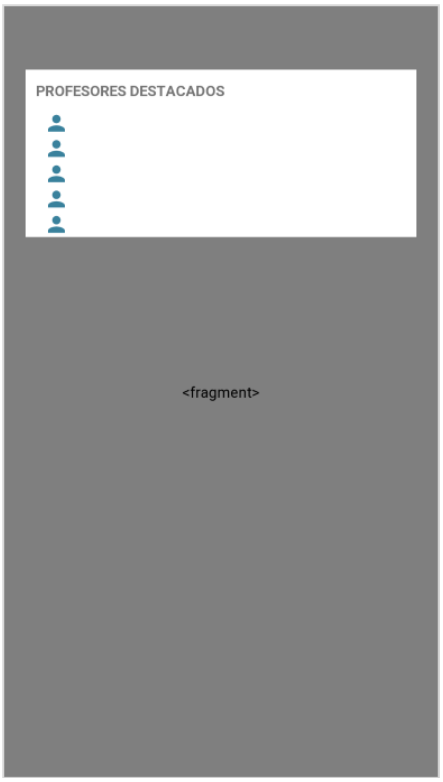

Figura 2.25. *Activity* para desplegar el mapa y mostrar los tutores cercanos a la ubicación de la búsqueda

2.3.5. ITERACIÓN 4

Al ingresar al perfil de un tutor por parte de un usuario estudiante, es posible asignarle al tutor una calificación y un comentario.

2.3.5.1. Calificaciones y Comentarios del Usuario Tutor

Es necesario recuperar el perfil del tutor realizando una petición al servicio, para lo cual se utiliza la función `getToken()`, que revisará si el token es válido. Si la verificación es exitosa, ejecutará los métodos `mostrarProfesor()` y `cargarCategorias()`. El servidor mediante los *endpoint* "/api/getPorfesor" y "/api/listarCategorias", realiza las consultas, obtiene los atributos del tutor seleccionado y envía la respuesta al cliente, como se muestra en el Código 2.29 y Código 2.30.

```
if(log){
    Usuario user = usuarioRepository.getByID(Integer.parseInt(id)).get(0);
    Cliente client = clienteRepository.findByUsuario(user).get(0);
    listaCursos=cursoRepository.findAll();
    lprofesor =profesorRepository.findAll();
    listaCursos =cursoRepository.findAll();
    // Filtrado para obtener los cursos del Porfesor por id
    for(int i=0;i<listaCursos.size();i++)
    {
        if(listaCursos.get(i).getIdcliente()!=Integer.parseInt(id))
            listaCursos.remove(i);
    }
    //Obtengo los atributos
    respuesta.put("respuesta", 1);
    respuesta.put("contrasena", user.getContrasena());
    respuesta.put("nombre_usuario", user.getNombre_usuario());
    respuesta.put("nombre", client.getNombre());
    respuesta.put("apellido", client.getApellido());
    respuesta.put("telefono", client.getTelefono());
    respuesta.put("fecha", client.getFecha());
    respuesta.put("perfil", client.getPerfil());
    // Obtengo los atrobutos unicos de profesor mediante id
    for(int i=0;i<lprofesor.size();i++) {
        if(lprofesor.get(i).getIdcliente()==Integer.parseInt(id)) {
                respuesta.put("descripcion", lprofesor.get(i).getDescripcion());
                respuesta.put("fotohojavida", lprofesor.get(i).getFotohojavida());
                respuesta.put("hojavida", lprofesor.get(i).getHojavida());
        }
    }

    respuesta.put("cursos", listaCursos);
    listaComentario =comentarioRepository.findAll();
    // Filtrado para obtener los comentarios del profesor
    int n1=0;
    for(int i=0;i<listaComentario.size();i++)
    {
        if(listaComentario.get(i).getIdcliente()!=Integer.parseInt(id))
        {   listaComentario.remove(i);}
        else
        {n1++;}
    }

    if(n1==0)
    {listaComentario=new ArrayList<>();}

    respuesta.put("comentarios", listaComentario);
}
else
```

Código 2.29. Consulta a la base de datos para obtener un usuario Profesor

```
@PostMapping("/listarCategorias")
public ResponseEntity<Object> listarcategorias(
        @RequestParam("tokenID") String  tokenID) {

    Map<String, Object> respuesta = new ConcurrentHashMap<>();
    List<Categorias> listaCategorias = new ArrayList<>();

    try {

            boolean log = false;
            //Revision de token
            try {
                Jwts.parser().setSigningKey(DatatypeConverter
                        .parseBase64Binary(Constantes.getSecretPassword()))
                        .parseClaimsJws(tokenID).getBody();
                log = true;
            } catch ( Exception e) {
                log = false;
            }

            if(log){
                // Consulta a la base de datos
                listaCategorias = categoriasRepository.findAll();
            }

    }catch (Exception e) {  }
    // Respuesta enviada al cliente, en caso de consulta exitosa
    return ResponseEntity.ok(listaCategorias);
}
```

Código 2.30. Consulta la base de datos para obtener todas las Categorías

En el **Código 2.*31***, se muestra el cómo el método `mostrarProfesor()`, despliega la información obtenida en las diferentes herramientas visuales del Activity.

```
// Muestra la info en los las herramientas visuales
edtName.setText(listado.nombre);
edtMobile.setText(listado.telefono);
edtCorreo.setText(listado.nombre_usuario);
etPlannedDate.setText(listado.fecha);
String foto = listado.perfil;
fotoCode = listado.perfil;
etDescripcionProfecional.setText(listado.descripcion);
ITEMS3.clear();
// Filtrado de cursos por profesor
for(int i= 0 ;i<listado.cursos.size() ;i++ )
{
    CursoDTO curso= listado.cursos.get(i);
    //Correccion de cursos
    for(int a =0 ;a<listadoCaracteristicas.size();a++) {
        if(id.equalsIgnoreCase( anotherString: curso.idcliente+"") &&
                listadoCaracteristicas.get(a).getId()==curso.idcategoria)
            curso.descripcion = listadoCaracteristicas.get(a).tipo;
    }
    ITEMS3.add(curso);
}
// Cargar spinner de cursos
((BaseAdapter) dropdown3.getAdapter()).notifyDataSetChanged();
dropdown3.invalidate();
dropdown3.setSelection(0);
String text ="";
// Mostrar comentarios
for(int i= 0 ;i<listado.comentarios.size() ;i++ )
{
    ComentarioDTO comentario= listado.comentarios.get(i);
    if(comentario.idcliente==Integer.parseInt(id))
    text+= "<p><font color='#39819c'>"+comentario.nombre+":</font>"+comentario.comentario+"</p>";
}

//Mostrar descripcion profesional
etDescripcionProfecional2.setText(Html.fromHtml(text), TextView.BufferType.SPANNABLE);
mRatingBar.setRating(Float.parseFloat(listado.fotohojavida));
hojaDeVida =listado.hojavida;
edtMisMaterias4.setText("Hoja de vida PDF");
// Cargar PDF en la direccion de memoria del telefono predeterminada
try {
    FileOutputStream fos =
            new FileOutputStream( name: "/data/user/0/ec.seb.tutoriopolis/pdftemp.pdf");
    fos.write(Base64.decode(hojaDeVida, Base64.NO_WRAP));
    fos.close();
}catch (Exception e){
    int a=3;
}
// Decodifiacion de foto de perfil
byte[] decodedString = Base64.decode(foto, Base64.DEFAULT);
Bitmap decodedByte = BitmapFactory.decodeByteArray(decodedString,  offset: 0, decodedString.length);
imgProfile.setImageBitmap(decodedByte);
```

Código 2.31. Método para mostrar los atributos de un Tutor, en los objetos visuales del *Activity*

Cuando se registra un comentario o una calificación en el perfil del tutor, el botón “Guardar Comentario” desencadena el método `saveComentario()`, que recupera la calificación del spinner, el cual tiene las opciones de calificación de una a cinco estrellas y ejecuta el método `registrarComentario()`. Mediante el *endpoint* “/api/registrarComentario” se envía al servidor la información de calificación y comentario para ser almacenada en la base de datos y actualiza en el *Activity* la información, como se muestra en el **Código 2.*32***.

```
public void registrarComentario(String token,String nombre,String comentario,int idProfesor,int evaluacion)
{
    ApiRestInterface apiService = ClientRest.getClient().create(ApiRestInterface.class);
    Call<String> tokencall = apiService.registrarComentario(token,nombre,comentario,idProfesor,evaluacion);
    tokencall.enqueue(new retrofit2.Callback<String>() {
        @Override
        public void onResponse(Call<String> call, retrofit2.Response<String> response) {
            try {
                // Revisa el contenido del body del mensaje
                if (response.body() != null) {
                    String listado = response.body();
                    if(listado.equalsIgnoreCase( anotherString: "1"))
                    {
                        // Obtiene el comentario del editText y lo muestro en pantalla
                        String text=etDescripcionProfecional2.getText().toString();
                        text+="<p><font color='#39819c'>"+texto+":</font>"+etcomentario
                                .getText().toString()+"</p>";
                        etDescripcionProfecional2.setText
                                (Html.fromHtml(text), TextView.BufferType.SPANNABLE);
                        Toast.makeText
                                ( context: InformacionProfesionalActivity.this
                                        , text: "Comentario guardado correctamente"
                                        , Toast.LENGTH_SHORT).show();
                    }
                }
            } catch (Exception e) {
                {
                    Toast.makeText( context: InformacionProfesionalActivity
                            .this, text: "Error al guardar,inntente mas tarde", Toast.LENGTH_SHORT).show();
                }
            }
        }

        @Override
        public void onFailure(Call<String> call, Throwable t) {
            {
                Toast.makeText( context: InformacionProfesionalActivity
                        .this, text: "Error al guardar,inntente mas tarde", Toast.LENGTH_SHORT).show();
            }
        }
    });
}
```

Código 2.32. Método para realizar la petición al servicio de registrar un comentario

Finalmente, en el servidor se calcula la calificación promedio y se guarda junto al comentario por medio del identificador único del usuario Tutor, como se muestra en el **Código 2.*33***.

```
if(log){

    lprofesor =profesorRepository.findAll();

    Comentarios comentario1 = new Comentarios();
    comentario1.setIdcliente(idProfesor);
    comentario1.setComentario(comentario);
    comentario1.setNombre(nombre);
    comentarioRepository.save(comentario1);

    Profesor profesor =  null;

    for(int i=0;i<lprofesor.size();i++) {
        if(lprofesor.get(i).getIdcliente()==idProfesor) {
            profesor =  lprofesor.get(i);
        }
    }

    if(profesor!=null)
    {
        // Realiza el promedio de calificaciones
        double eva = Double.parseDouble(profesor.getCalificacion());
            eva = (eva+evaluacion)/2;
        profesor.setCalificacion(eva+"");
        profesorRepository.save(profesor);
    }
    respuesta="1";
}


catch (Exception e)
{
    respuesta="0";
}
```

Código 2.33. *Endpoint* para realizar la petición al servidor de actualizar la calificación, junto con el comentario y almacenarla en la base de datos

En la **Figura 2.*26*** se muestra la interfaz de perfil del usuario Tutor para registrar el comentario y calificación.

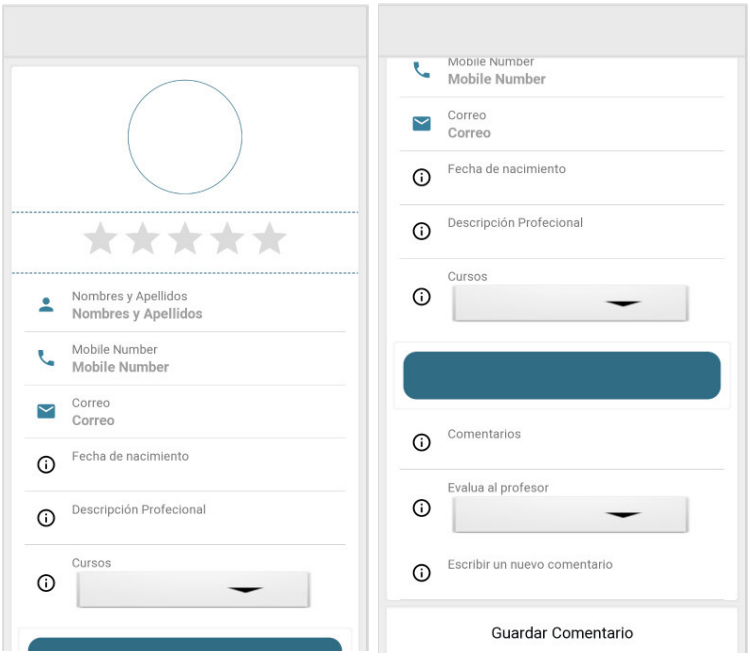

Figura 2.26. Activity para registrar comentarios y calificaciones en el perfil Tutor

3. METODOLOGÍA

El uso de la metodología XP, requiere implementar pruebas de aceptación por cada módulo implementado en el sistema, con la finalidad de comprobar su correcto funcionamiento. En esta sección, se especifican los resultados obtenidos en las pruebas de aceptación realizadas, en cada una de las iteraciones.

3.1. PRUEBAS DE ACEPTACIÓN

Una vez finalizado la fase de desarrollo e implementación del sistema prototipo, se procedió a realizar las pruebas de aceptación de cada una de las iteraciones, con la finalidad de validar su funcionamiento.

3.1.1. PRUEBAS DE ACEPTACIÓN – ITERACIÓN 1

3.1.1.1. Login del Sistema

Procedimiento: El primer escenario describe el acceso de los usuarios registrados en el sistema, la autenticación se la realiza mediante el correo y la contraseña. En caso de que los datos utilizados sean incorrectos o el usuario no se encuentre registrado, se desplegará un mensaje indicando que los campos no son válidos.

Resultados: Existen tres tipos de ingreso, para estudiantes, tutores y administrador. En la **Figura 3.*1*** se muestra el caso en el cual un estudiante registrado ha podido ingresar al sistema exitosamente. En la **Figura 3.*2*** se puede observar la autenticación de un tutor registrado de manera correcta y que es redirigido al *Activity* de perfil del tutor. En la **Figura 3.*3*** se muestra el *Activity* de administración de usuarios, que corresponde al ingreso exitoso del administrador. Finalmente, en la **Figura 3.*4*** se muestra el caso, en el cual un usuario no ha podido llevar a cabo la autenticación con éxito, independientemente del rol que tenga en el sistema.

Figura 3.1. Ingreso exitoso de un Estudiante

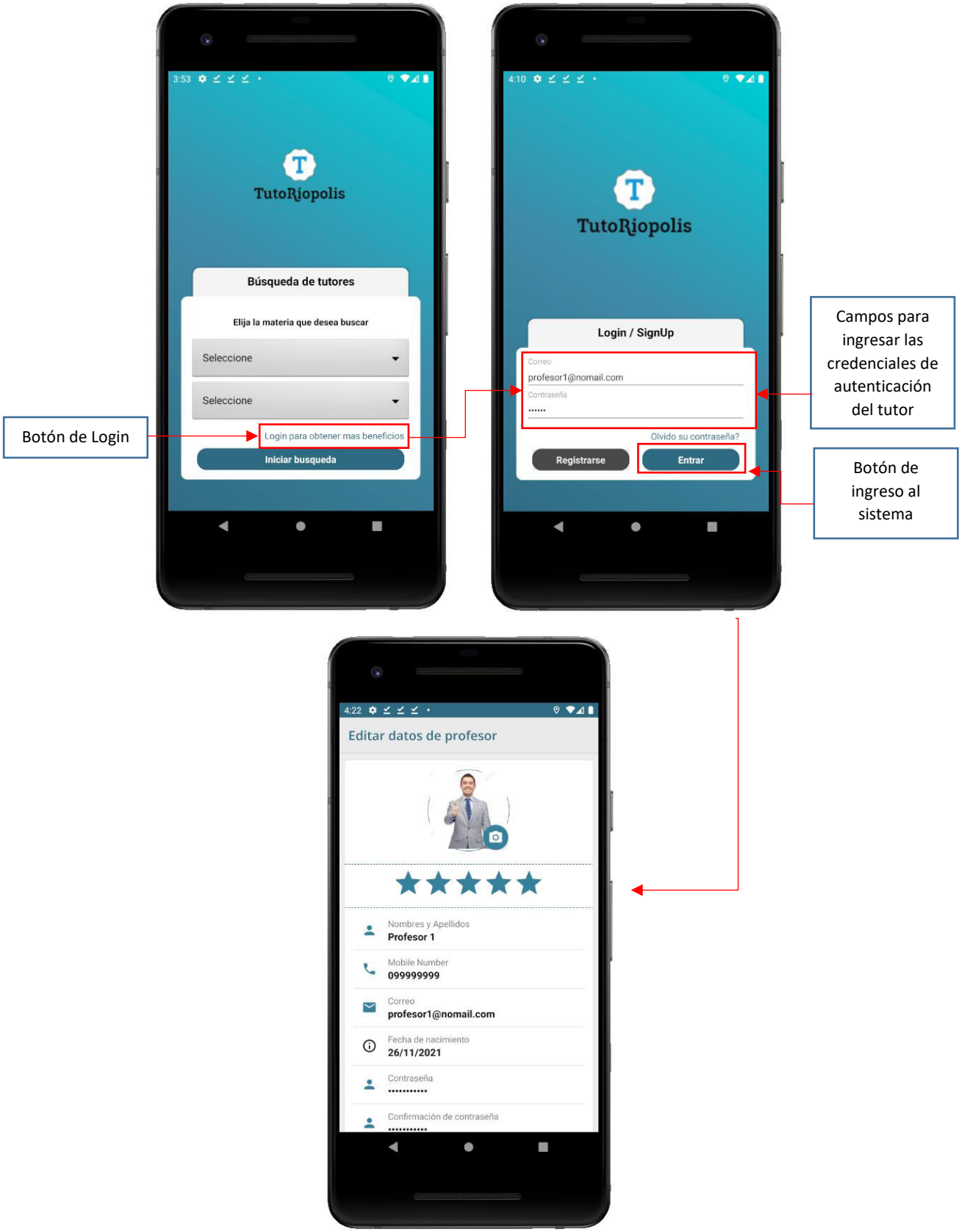

Figura 3.2. Ingreso exitoso de un Tutor

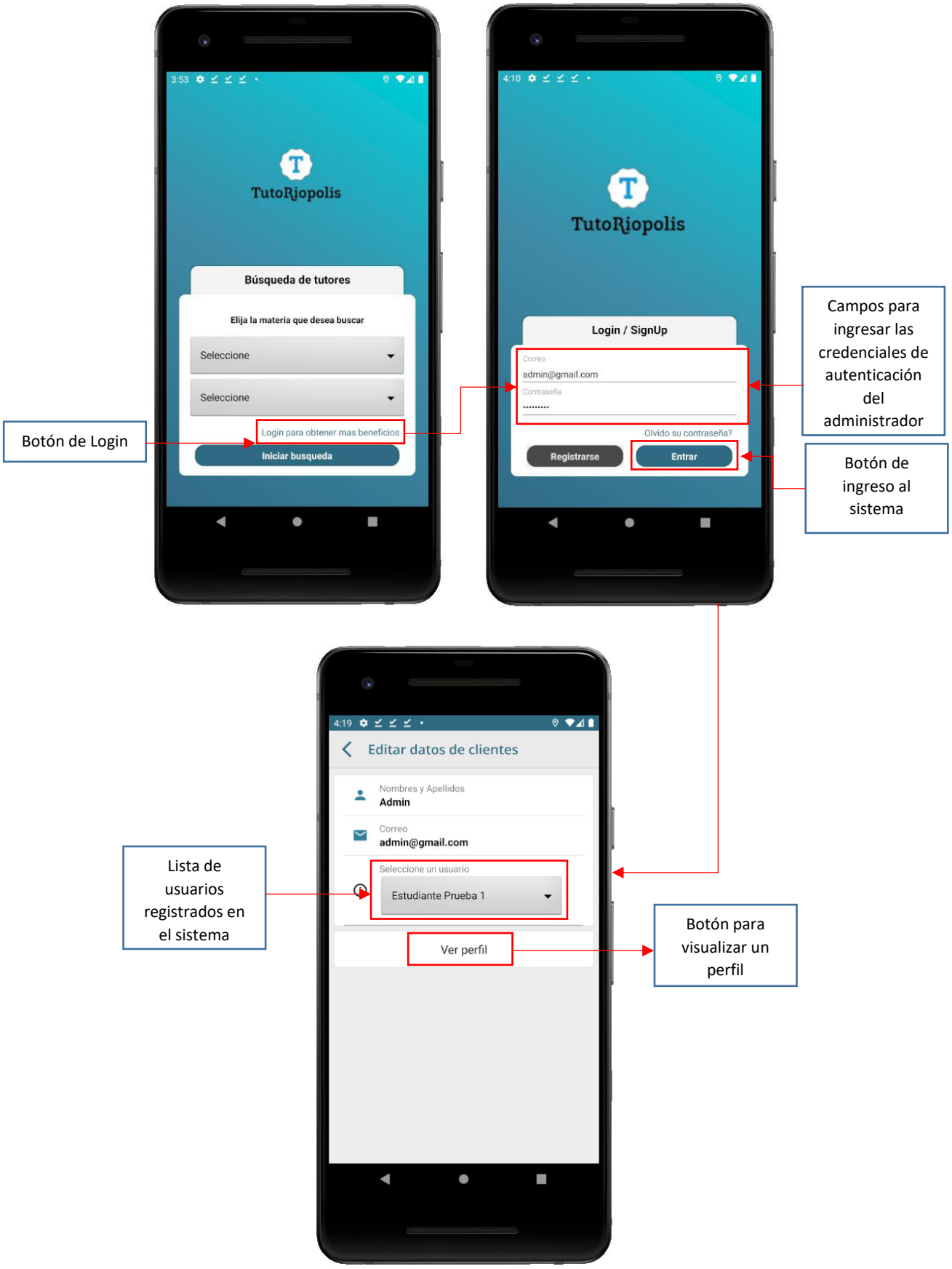

Figura 3.3. Ingreso exitoso del Administrador

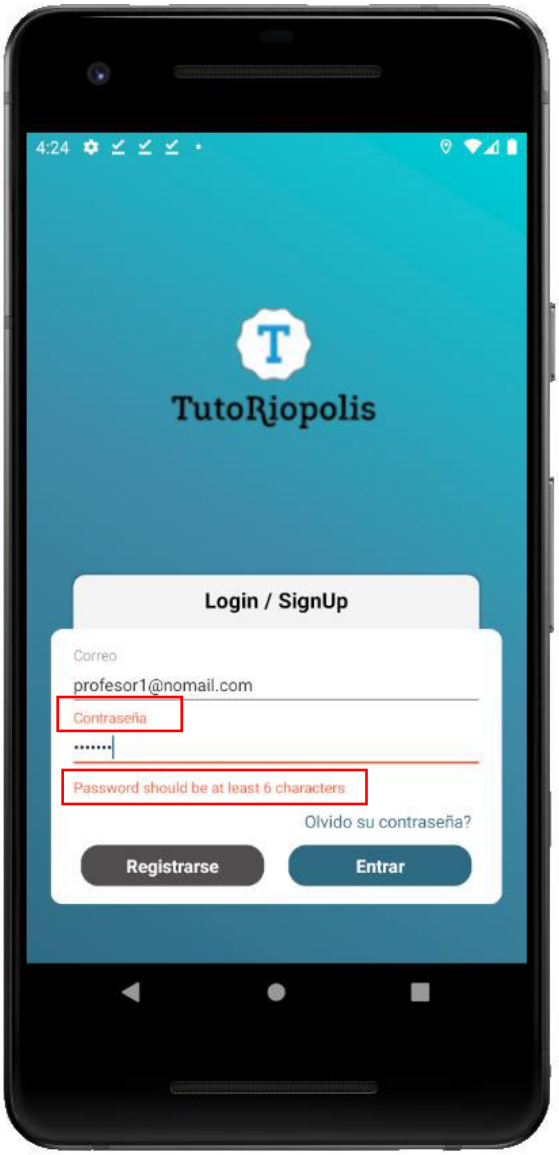

Figura 3.4. Autenticación que no pudo ser completada con éxito

3.1.1.2. Validar el Perfil del Usuario Tutor

Procedimiento: En este escenario un tutor registrado previo a aparecer como resultado en las búsquedas, debe ser aprobado por el administrados del sistema, con la finalidad de que la información sea verificada y así, aportar un nivel de seguridad al sistema. El administrador puede modificar, eliminar, activar o desactivar la cuenta.

Resultados: En la Figura 3.5 se muestra la interfaz de administrador, en la cual se puede acceder a una lista de profesores, seleccionarlos e ingresar al perfil de cada uno de ellos, de esta manera se puede revisar la información y usar las opciones de modificar, eliminar, activar o desactivas cuenta. En la Figura 3.6 se muestra cómo elegir un perfil Tutor para visualizarlo. En la Figura 3.7 se observa el perfil del tutor seleccionado junto con los botones

de activar y desactivar, que permitirán al tutor aparecer o no en las búsquedas, el botón modificar, que permitirá al administrador editar la información del perfil de un tutor y por último, el botón eliminar, que borrará de la base de datos la información de un determinado tutor.

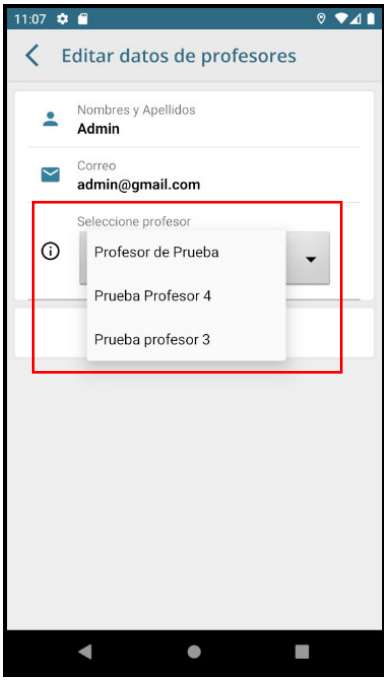

Figura 3.5. Interfaz de administración de usuarios

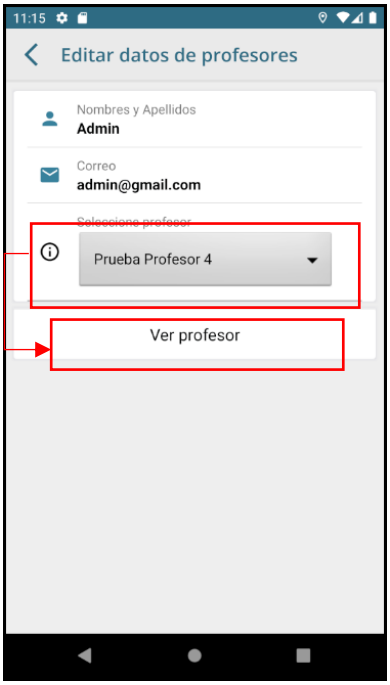

Figura 3.6. Elección del perfil tutor a revisar la información

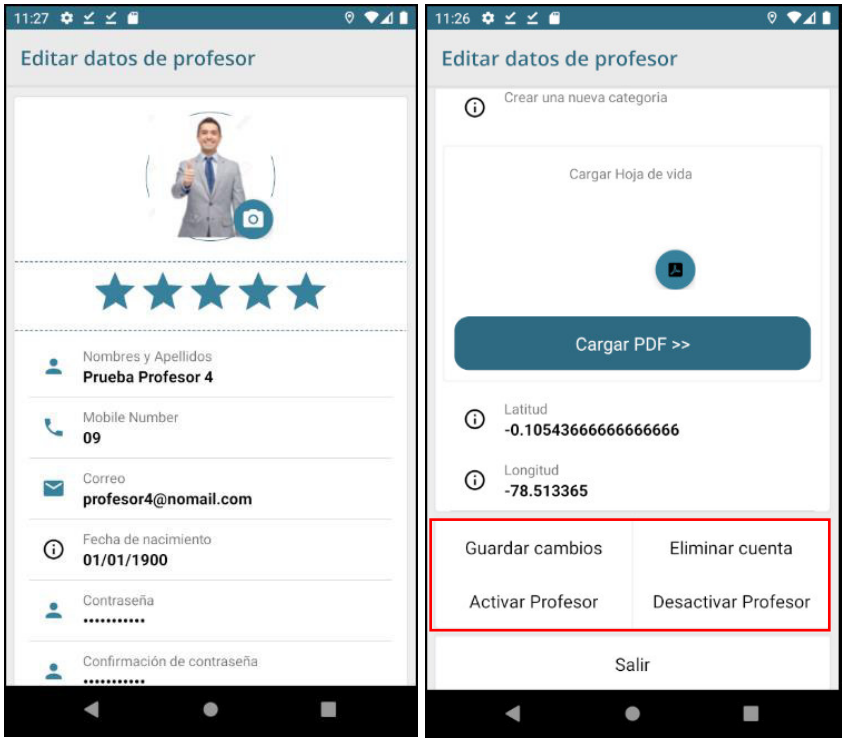

Figura 3.7. Vista de un perfil Tutor con los botones de administración

3.1.1.3. Modificar, Activar, Desactivar y Eliminar Cuenta de un Usuario Tutor

Procedimiento: En este escenario se han utilizado cuatro botones para la administración de tutores, los mismos que permitirán al administrador, modificar la información en caso de que sea necesario, activar o desactivar la cuenta con la finalidad de mostrarlo o no en las búsquedas y finalmente eliminar la cuenta si el uso ha sido indebido dentro del sistema.

Resultados: Todo tutor que se registre en la aplicación mantiene un estado de “No Activado” hasta que el administrador revise su información y sea aprobado, por lo tanto, mientras ésta no sea revisada el tutor no podrá aparecer dentro de las búsquedas que realice un usuario estudiante. Para realizar las pruebas de activación de usuarios del tipo tutor, se usarán tres tutores. El usuario “Profesor 1” se encuentra activo, por lo que se podrá visualizar en las búsquedas, los tutores “Profesor 2” y “Profesor 3” se encuentran desactivados, a espera de la activación, por lo cual, no es posible encontrarlos en las búsquedas, como se muestra en la Figura 3.8.

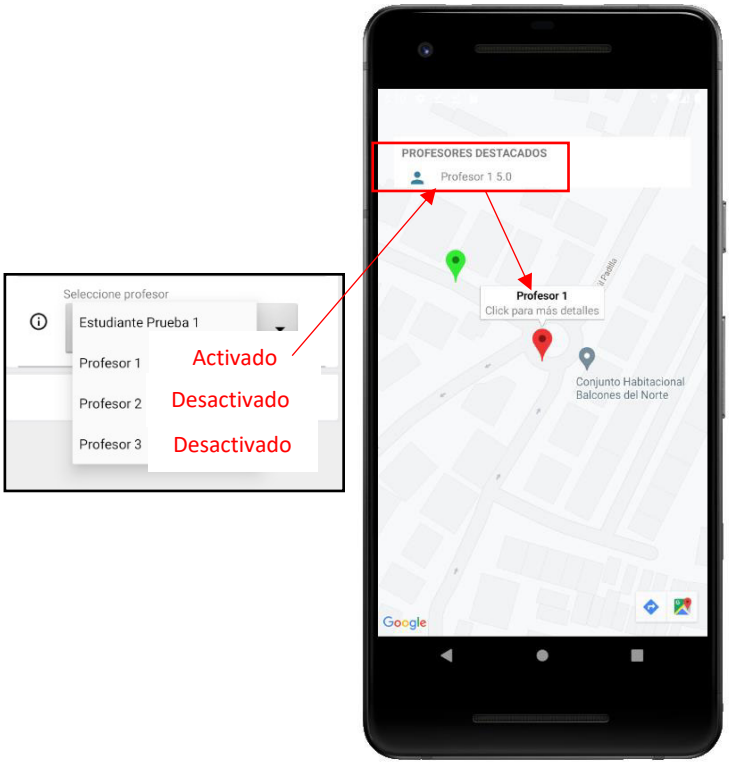

Figura 3.8. Los usuarios activados son visibles en las búsquedas

Una vez que un tutor sea activado, éste aparecerá en las búsquedas como se muestra en la Figura 3.9. El tutor “Profesor 3” mantiene su estado de “Desactivado” por lo que no es posible visualizarlo en el mapa.

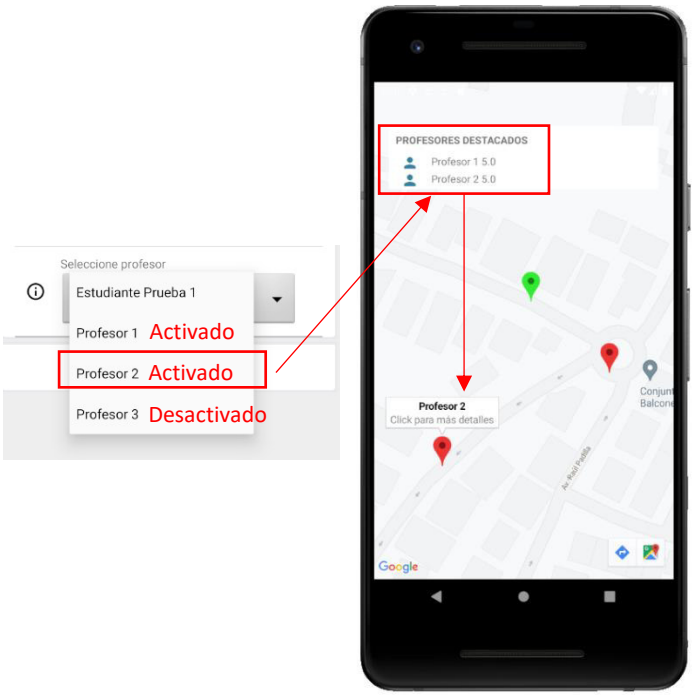

Figura 3.9. Una vez Activado un tutor, aparecerá en las búsquedas

La opción de desactivar, retirará al tutor de los resultados de las búsquedas que se realicen, pero no se lo eliminará del sistema, como se muestra en la Figura 3.10.

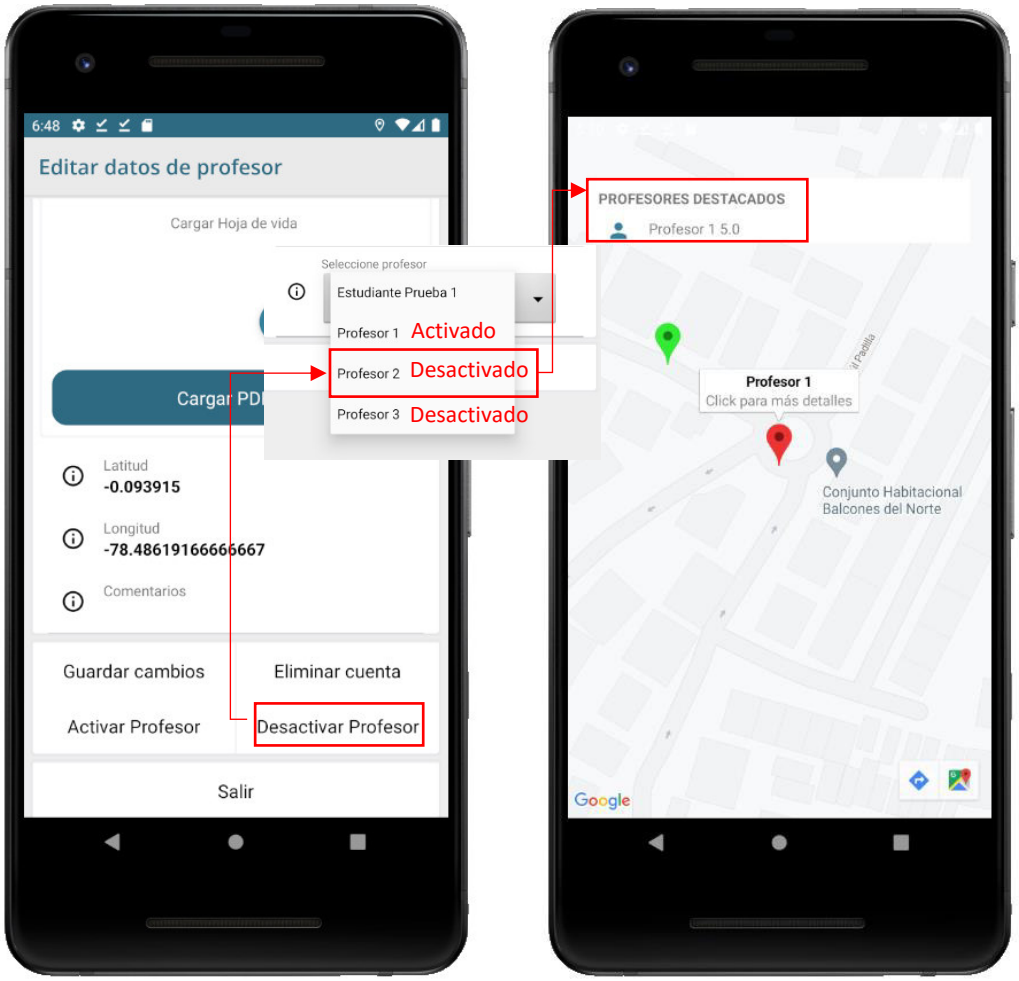

Figura 3.10. Desactivar un perfil tutor de las búsquedas

La opción “Eliminar cuenta” en el perfil de un usuario tutor, lo elimina permanentemente del sistema, por lo cual desaparecerá de la lista de profesores del *Activity* de administración, como se muestra en la Figura 3.11.

El usuario tutor que sea eliminado no podrá autenticarse en el sistema, ya que ha sido removido del sistema en su totalidad, como se muestra en la Figura 3.12.

Una vez la cuenta ha sido eliminada, no es posible volver a recuperarla y un nuevo registro es necesario.

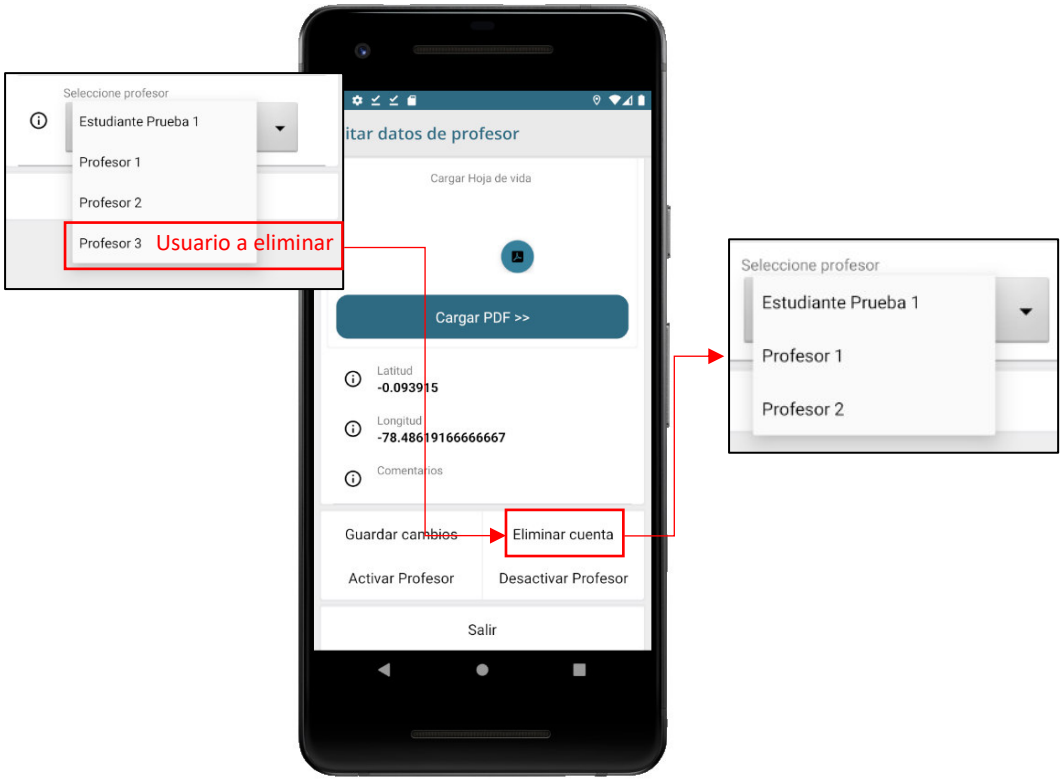

Figura 3.11. Usuario Eliminado desde la interfaz de Administrador

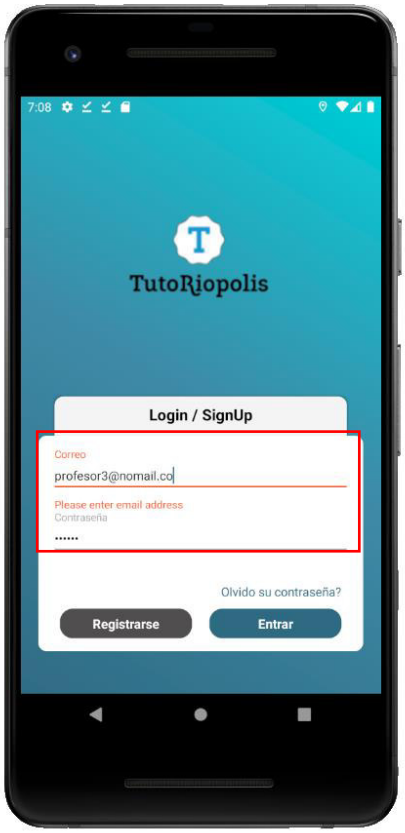

Figura 3.12. Una vez eliminado un usuario, la autenticación en el sistema no es posible

Finalmente, para modificar la información del perfil tutor, se debe pulsar sobre el campo que se desea editar y éste activará para realizar la edición, como se muestra en la Figura 3.13.

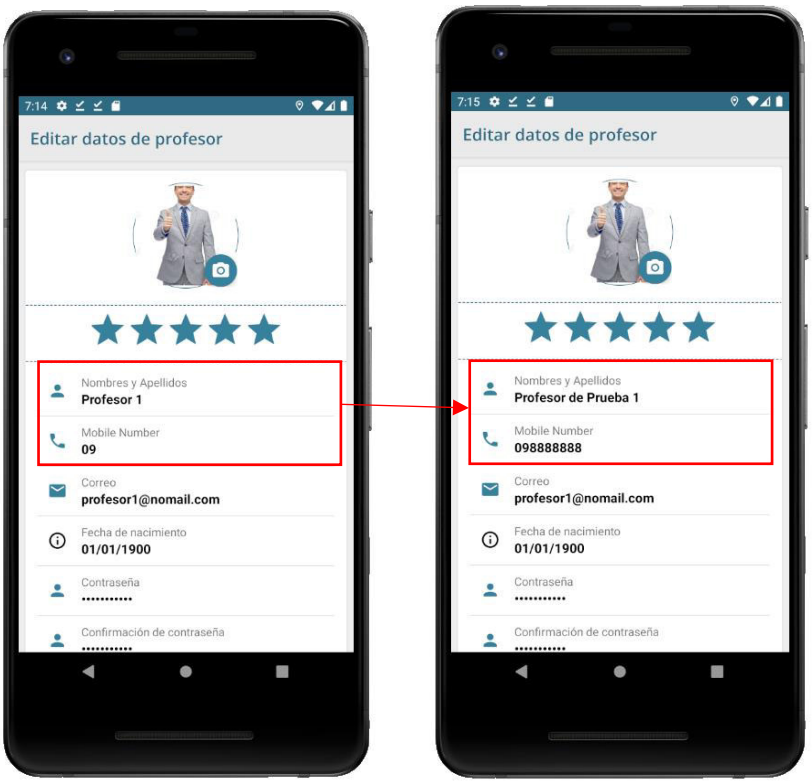

Figura 3.13. Modificar información del perfil tutor

3.1.2. PRUEBAS DE ACEPTACIÓN – ITERACIÓN 2

3.1.2.1. Registro Usuario

Procedimiento: Los usuarios para hacer uso de la aplicación deben registrarse en el sistema, para esto, existen dos tipos de registro, el registro de usuario tutor y el registro de usuario estudiante. Para ambos usuarios se despliega una *Activity*, en la cual se solicita información básica de registro, como nombre, correo, contraseña y el tipo de registro que se desea llevar a cabo, ya sea como profesor o estudiante. Una vez que se elige el tipo de registro, la aplicación redirecciona a la *Activity* para continuar el registro en el caso del usuario tutor o la redirige a la *Activity* de búsqueda en el caso del usuario estudiante.

Resultados: En la **Figura 3.*14*** se muestra la manera de hacer el registro en la aplicación, iniciando desde el *Activity* de inicio de la aplicación hasta la *Activity* de registro común para ambos usuarios.

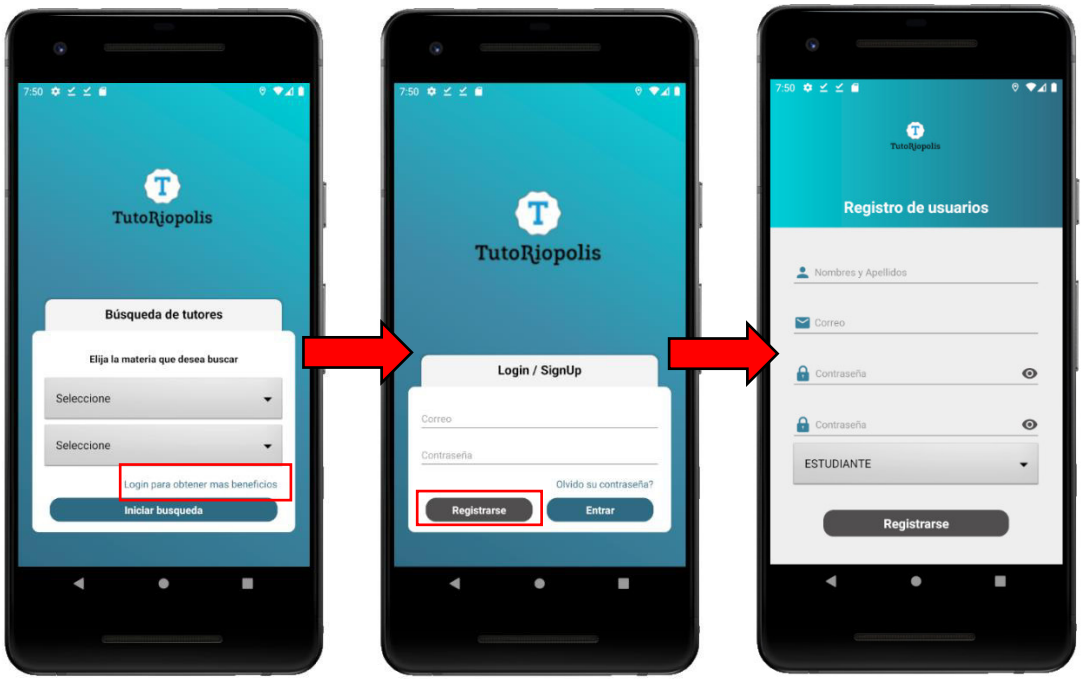

Figura 3.14. Procedimiento para realizar el registro en el sistema

Una vez en la *Activity* de registro común, es necesario llenar los campos con la información necesaria y elegir el rol que se desea tener dentro del sistema como se muestra en la **Figura 3.*15***.

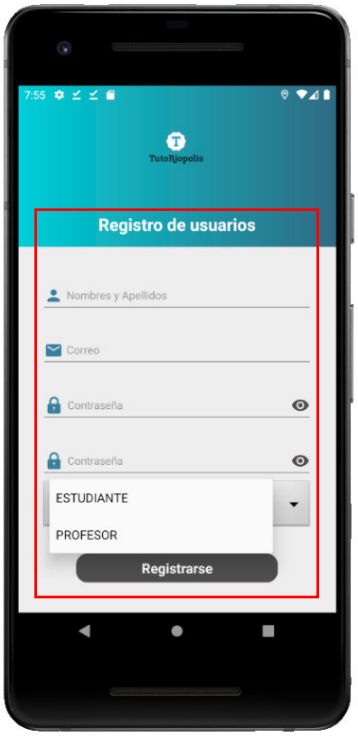

Figura 3.15. Activity común de registro para los usuarios Tutor y Estudiante

En el caso de que uno de los campos se encuentre vacío, no es posible continuar con el registro, ya que es obligatorio llenar toda la información, como se muestra en la **Figura 3.*16***. Para el caso del campo “correo” detecta si el correo ingresado posee el formato adecuado, si no cumple el formato, no es posible continuar, como se muestra en la **Figura 3.*17***.

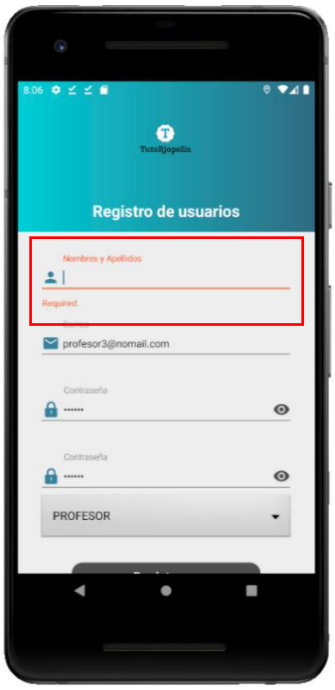

Figura 3.16. Si uno de los campos se encuentra vacío, no se puede realizar el registro

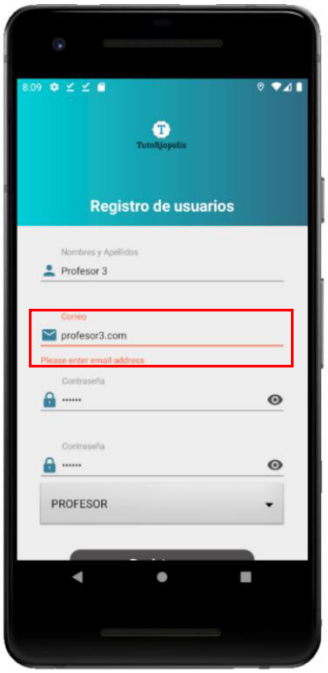

Figura 3.17. Control del formato del correo

En la **Figura 3.*18***, se presenta el control mínimo de 6 caracteres en el campo contraseña, para poder realizar el registro.

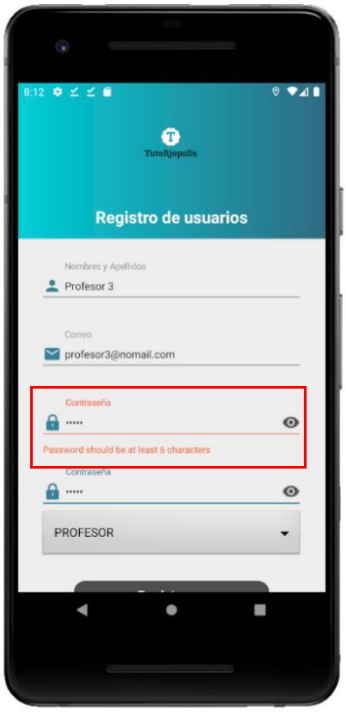

Figura 3.18. Control mínimo de 6 caracteres en el campo contraseña

En la **Figura 3.*19***, se realiza el control del campo contraseña, al ingresar nuevamente la contraseña, el sistema se asegura que la contraseña ingresada sea correcta.

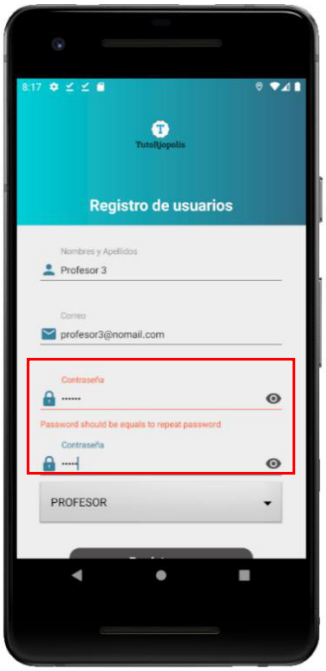

Figura 3.19. Control para comprobar que la contraseña ingresada, es correcta

En la **Figura 3.*20***, muestra que es posible visualizar la contraseña, con la finalidad de poder repetirla en el campo de confirmación de contraseña.

Figura 3.20. Control para visualizar la contraseña

En la **Figura 3.*21*** se muestran los roles que posee el sistema de registro, una vez que se llenan todos los datos y se elige el rol, se puede completar con el registro.

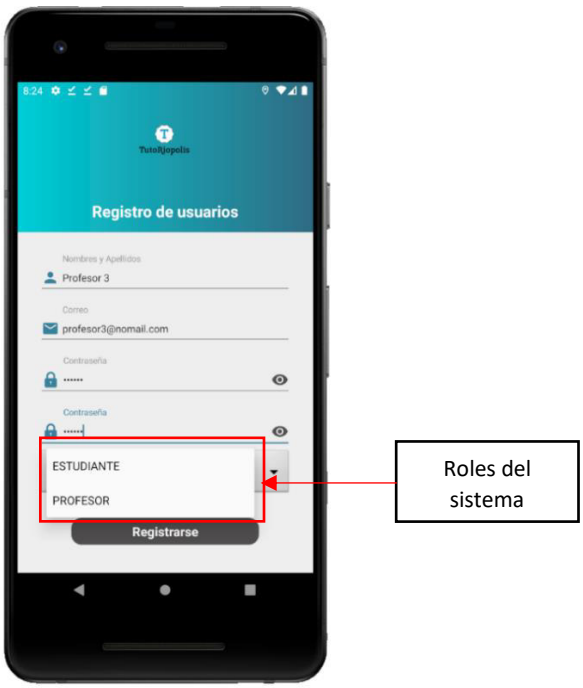

Figura 3.21. Roles disponibles en el sistema para realizar el registro

Si el rol elegido en el sistema es el de profesor, se despliega una *Activity* para continuar con el registro, en el cual es necesario llenar más campos de información, como: Foto, número de celular, fecha de nacimiento, descripción personal, cursos en base al nivel de educación, hoja de vida y automáticamente el sistema tomará los datos de longitud y latitud, como se muestra en la **Figura 3.*22***.

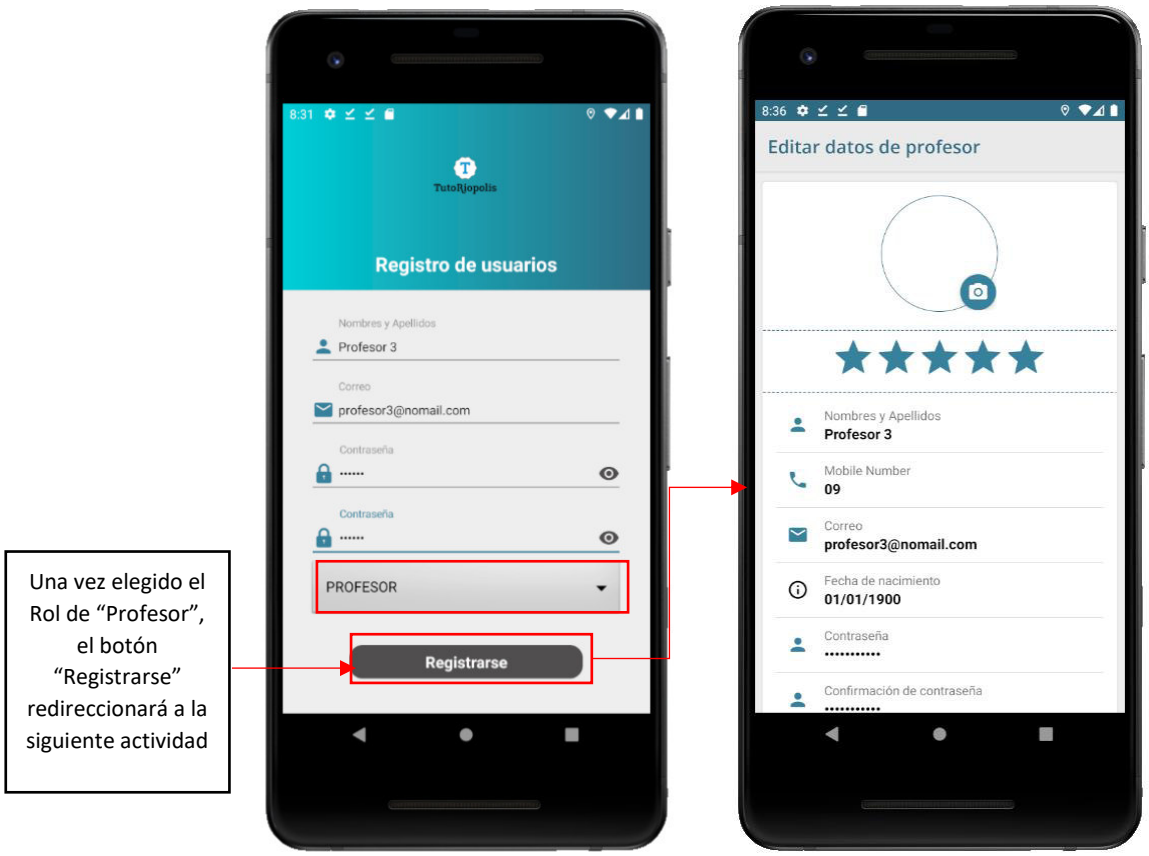

Figura 3.22. Registro usuario Tutor

Una vez en la *Activity* para continuar con el registro del tutor es posible ingresar una fotografía como se muestra en la **Figura 3.*23***. El campo destinado a recolectar el número de teléfono posee control de numeración por lo que el ingreso de un carácter alfabético no es posible.

En la **Figura 3.*24*** se muestra el ingreso de la fecha de nacimiento, para llenar este campo se desplegará el calendario que puede ser configurado en base al año, mes y día. De esta manera se facilita el ingreso y se evita problemas de formato de fecha, como se muestra en la **Figura 3.*25***.

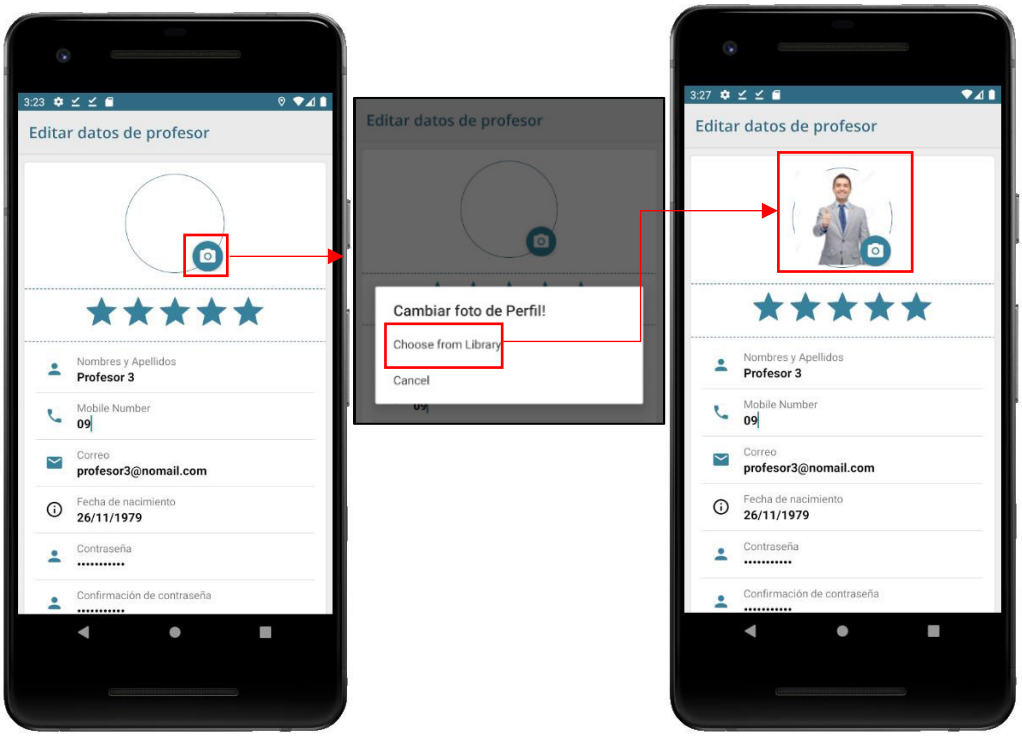

Figura 3.23. Cargar una foto en el perfil de un Tutor

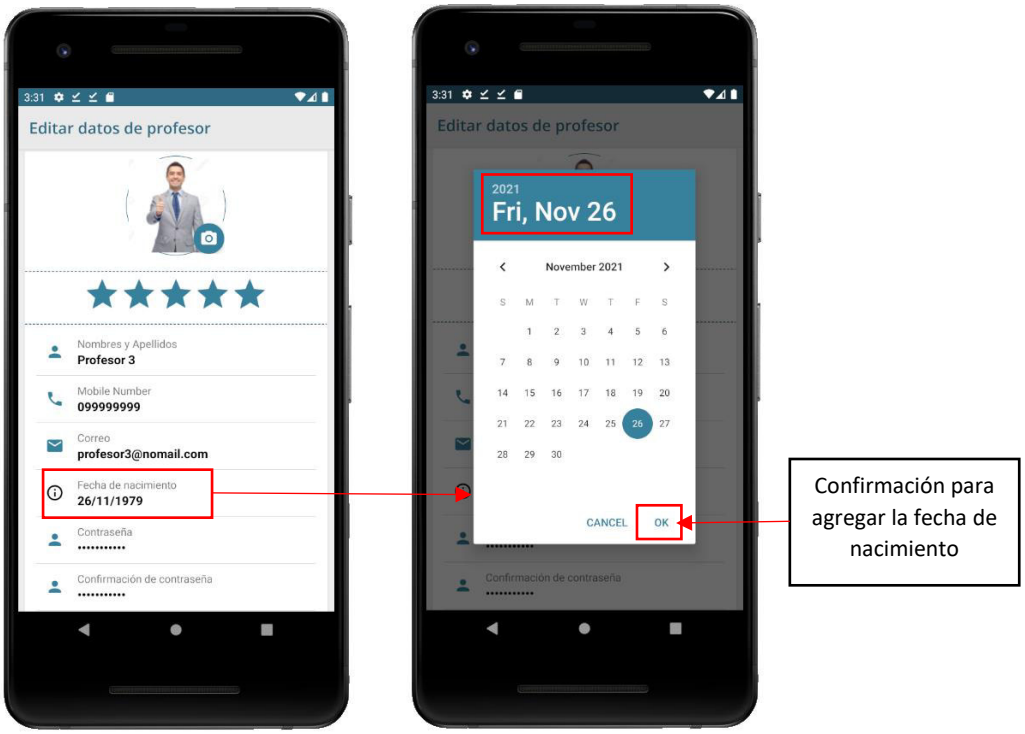

Figura 3.24. Ingreso de la fecha de nacimiento en el registro de un usuario Tutor

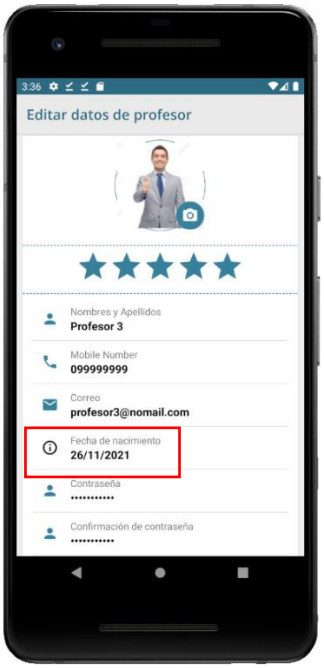

Figura 3.25. Fecha de nacimiento en formato dd/mm/aa

Los campos de contraseña y confirmación de contraseña son recuperados de la *Activity* de registro común, por lo que no es necesario ingresarlas nuevamente. El campo descripción profesional, permite ingresar una breve reseña profesional, como se muestra en la Figura 3.26.

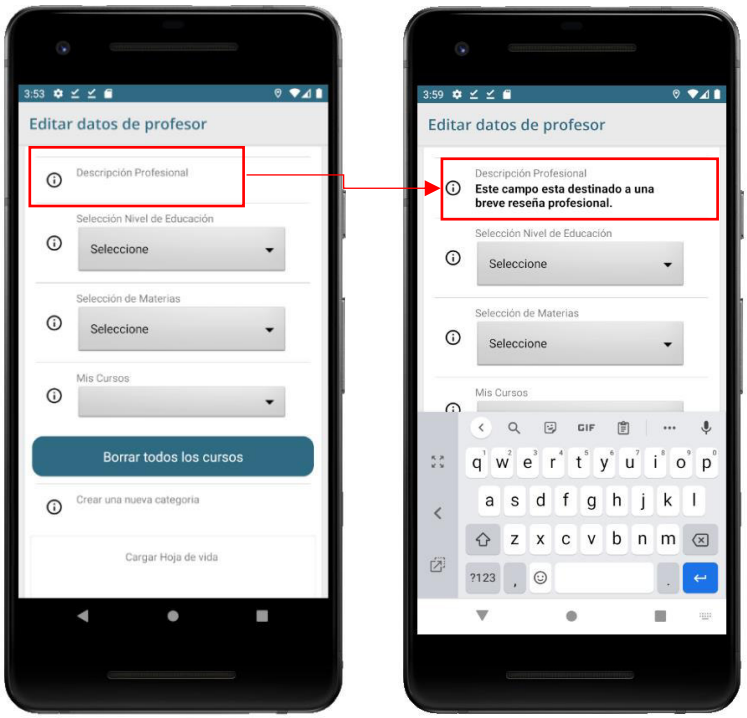

Figura 3.26. Agregar una descripción profesional en el perfil de un Tutor

El ingreso de materias se encuentra dividido en dos niveles, el primer nivel permite elegir el “Nivel de Educación” de la materia que se va a ingresar, una vez seleccionada esta opción, en el nivel dos, se desplegarán las materias que actualmente se encuentran en el sistema, una vez elegida la materia es necesario guardarla y se visualizará en el campo “Mis cursos” como se muestra en la **Figura 3.*27***.

Figura 3.27. Agregar una materia existente en el sistema al perfil de un tutor

En el caso de que una materia no se encuentre en el sistema, es posible agregarla, ya que existe un campo destinado para esta acción, el procedimiento es el mismo, en el primer nivel se elige el “Nivel de Educación” y se ingresa la nueva materia manualmente por medio del campo “Ingresar una nueva Categoría”, después de guardar los cambios es posible visualizarla en el campo “Mis cursos”, como se muestra en la **Figura 3.*28***.

Figura 3.28. Agregar una materia que no se encuentra en el sistema

En la Figura 3.29, se muestra cómo eliminar todos los cursos del perfil tutor, para volver a ingresarlos. Esta acción se lleva a cabo mediante el botón “Borrar todos mis cursos”, que elimina los registros de cursos en la base de datos y limpia el contenido del *spinner* “Mis cursos”.

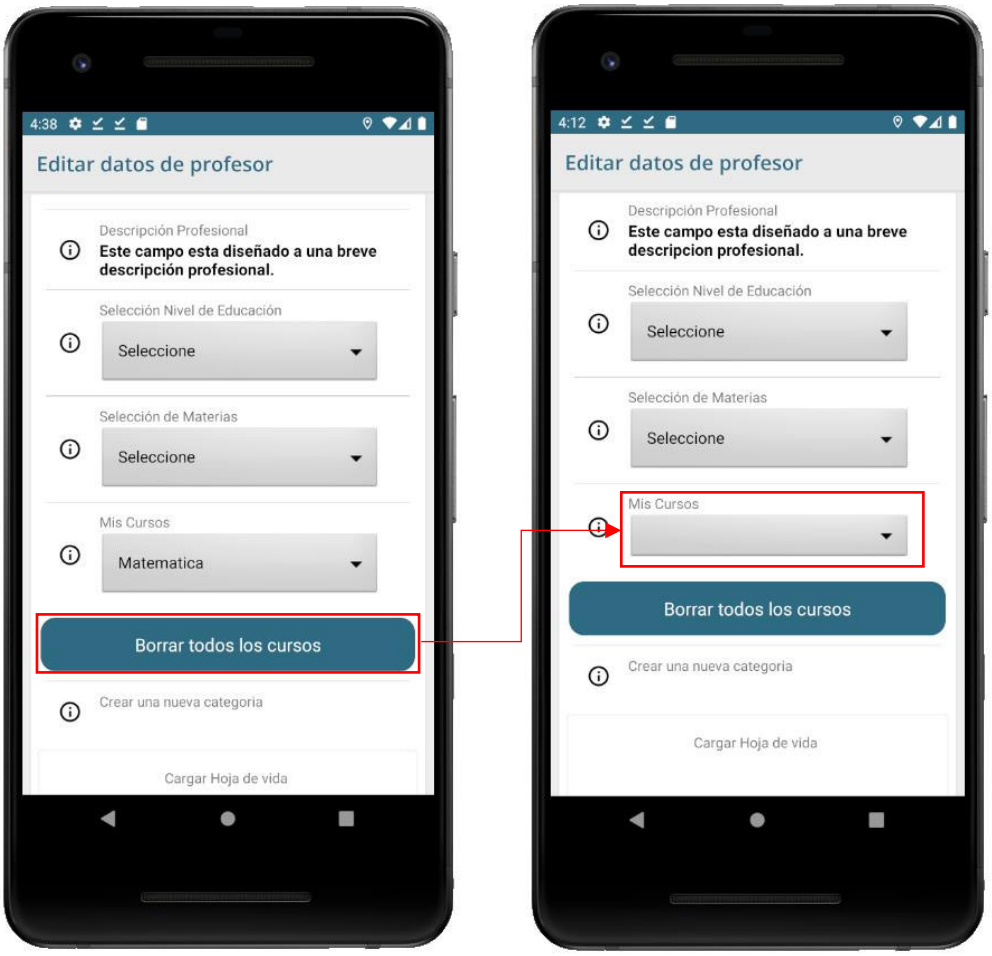

Figura 3.29. Eliminar cursos agregados en el perfil Tutor

En la Figura 3.30, se muestra el proceso de como cargar la hoja de vida en el perfil del tutor. Para realizar esta actividad se ha asignado un botón, el mismo que al presionarlo desplegará el buscador de documentos, que permitirá navegar por los archivos almacenados en el dispositivo, para posteriormente cargarlo en el perfil. Una vez que la hoja de vida se encuentre disponible en el perfil, se activará un botón con la etiqueta “Hoja de vida PDF”, que al pulsarlo permitirá visualizar el contenido del archivo PDF.

Figura 3.30. Carga y visualización del archivo PDF

En la **Figura 3.*31***, se muestra que de manera automática el sistema actualiza la posición del dispositivo mediante los campos longitud y latitud.

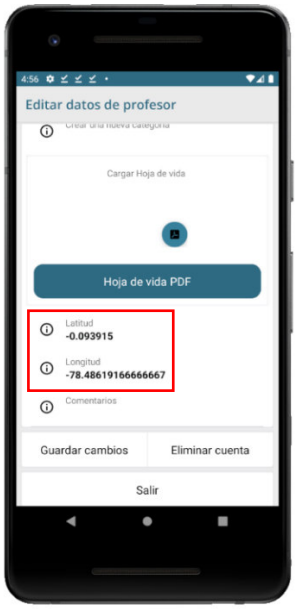

Figura 3.31. Actualización de manera automática de la posición mediante los campos "Longitud" y "Latitud"

El campo "Comentarios" se visualizará y actualizará de manera automática, siempre y cuando un estudiante agregue un comentario para un determinado tutor y no es posible editarlo.

La **Figura 3.*32*** se muestra el botón "Guardar cambios", el mismo que guarda todos los campos llenados anteriormente en la base de datos y el botón "Eliminar" que elimina el registro del perfil del tutor de la base de datos permanentemente.

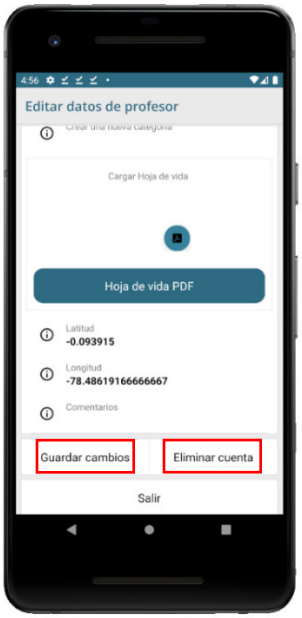

Figura 3.32. Botones "Guardar Cambio" y "Eliminar Cuenta"

La **Figura 3.*33***, muestra el registro de un usuario Estudiante, el mismo que al llenar la información básica y elegir el rol de estudiante se encontrará registrado, enviándolo automáticamente a la *Activity* de búsqueda de tutores.

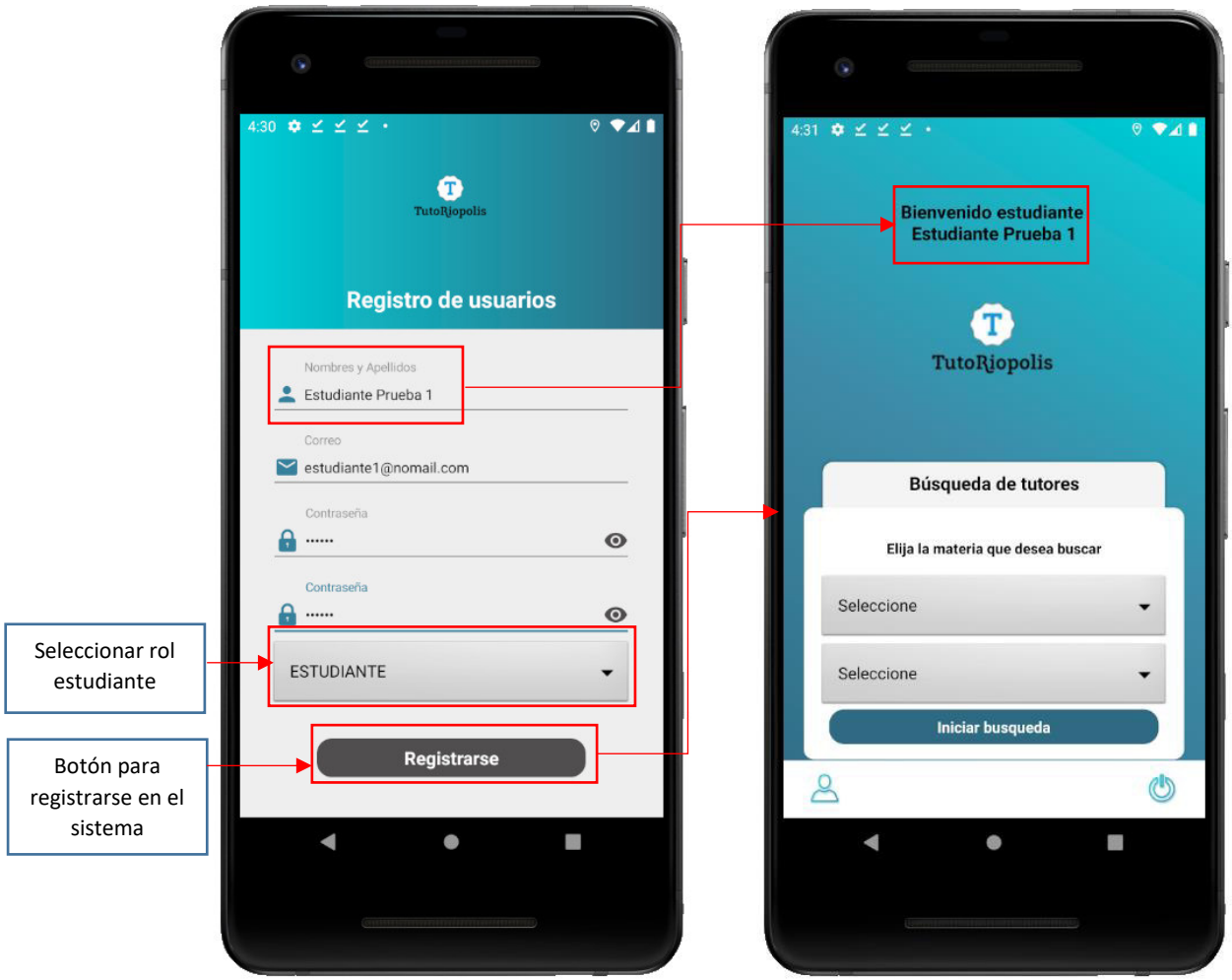

Figura 3.33. Registro de un usuario estudiante

Una vez el usuario estudiante se encuentra en la *Activity* de búsqueda de tutores podrá visualizar el mensaje de bienvenida del sistema y realizar las acciones de buscar un tutor, redirigirse a su perfil o cerrar sesión, como se muestra en la **Figura 3.*34***.

Figura 3.34. Activity de búsqueda del usuario Estudiante registrado

El botón de "Perfil", permitirá al usuario Estudiante dirigirse a su perfil, en el cual puede visualizar y actualizar sus datos si lo requiere. Los campos se mantienen activos para permitirle esta función, como se muestra en la **Figura 3.*35***.

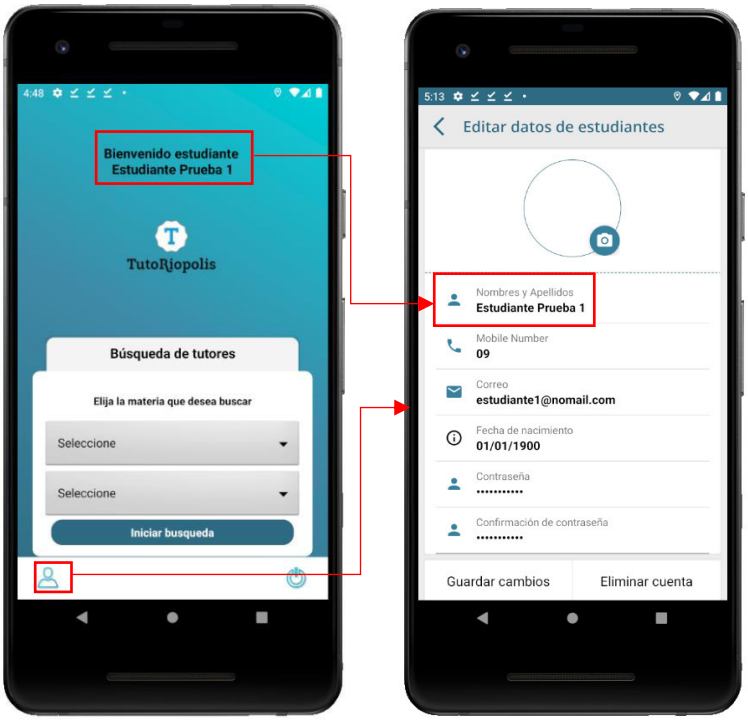

Figura 3.35. Activity perfil de usuario Estudiante

El botón “Guardar cambios” permite actualizar los datos que el usuario estudiante realizó en su perfil, ya sea cargar una fotografía, nombre, número de teléfono o contraseña, como se muestra en la Figura 3.36.

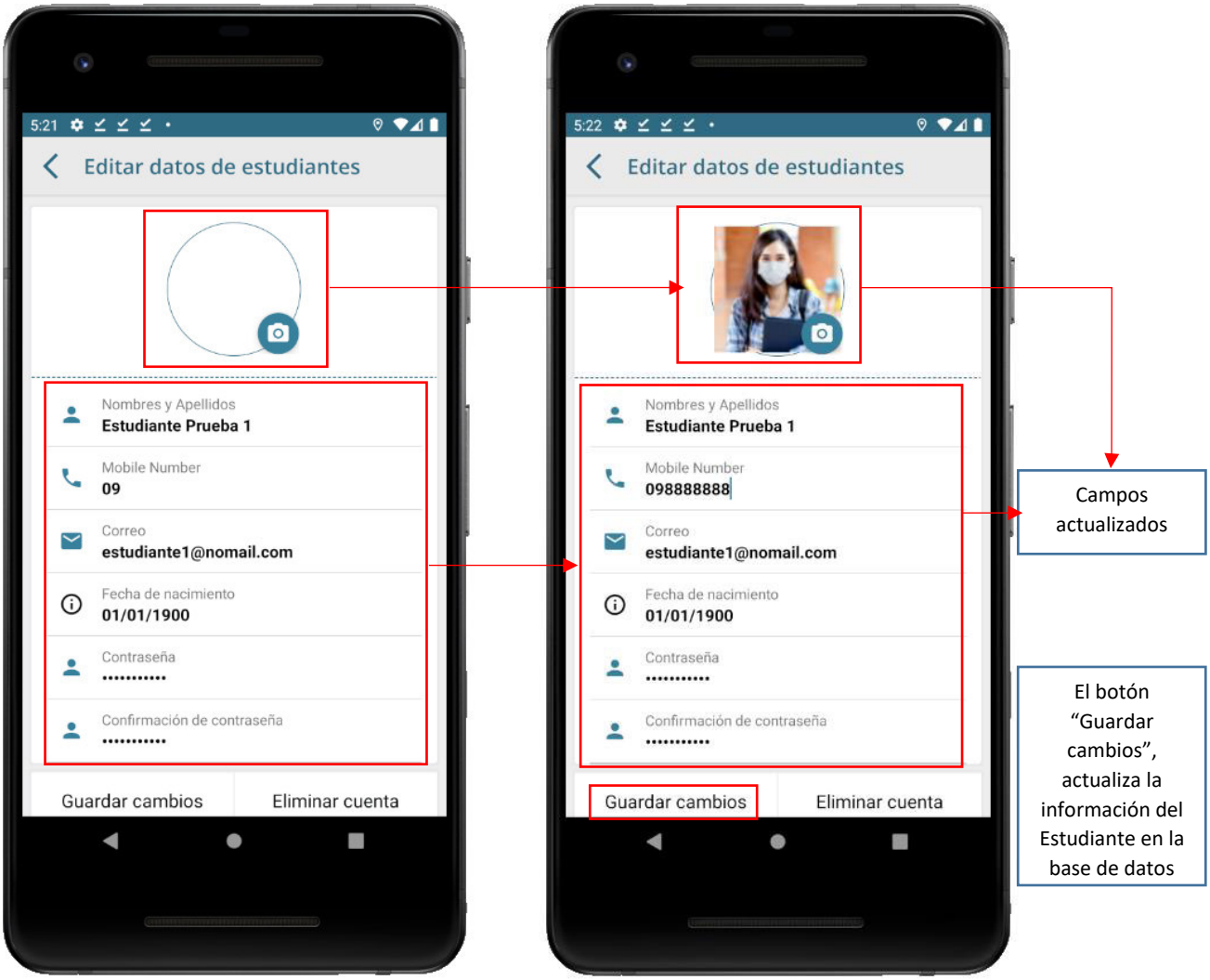

Figura 3.36. Editar información del perfil Estudiante

El botón “Eliminar cuenta”, borrará la información en la base de datos del Estudiante, sin la posibilidad de acceder al sistema y un nuevo registro será necesario para ingresar.

Finalmente, el botón “Cerrar sesión” le permitirá al usuario Estudiante salir del sistema, y para volver a ingresar será necesario la autenticación por medio de las credenciales correo y contraseña, como se muestra en la **Figura 3.*37***.

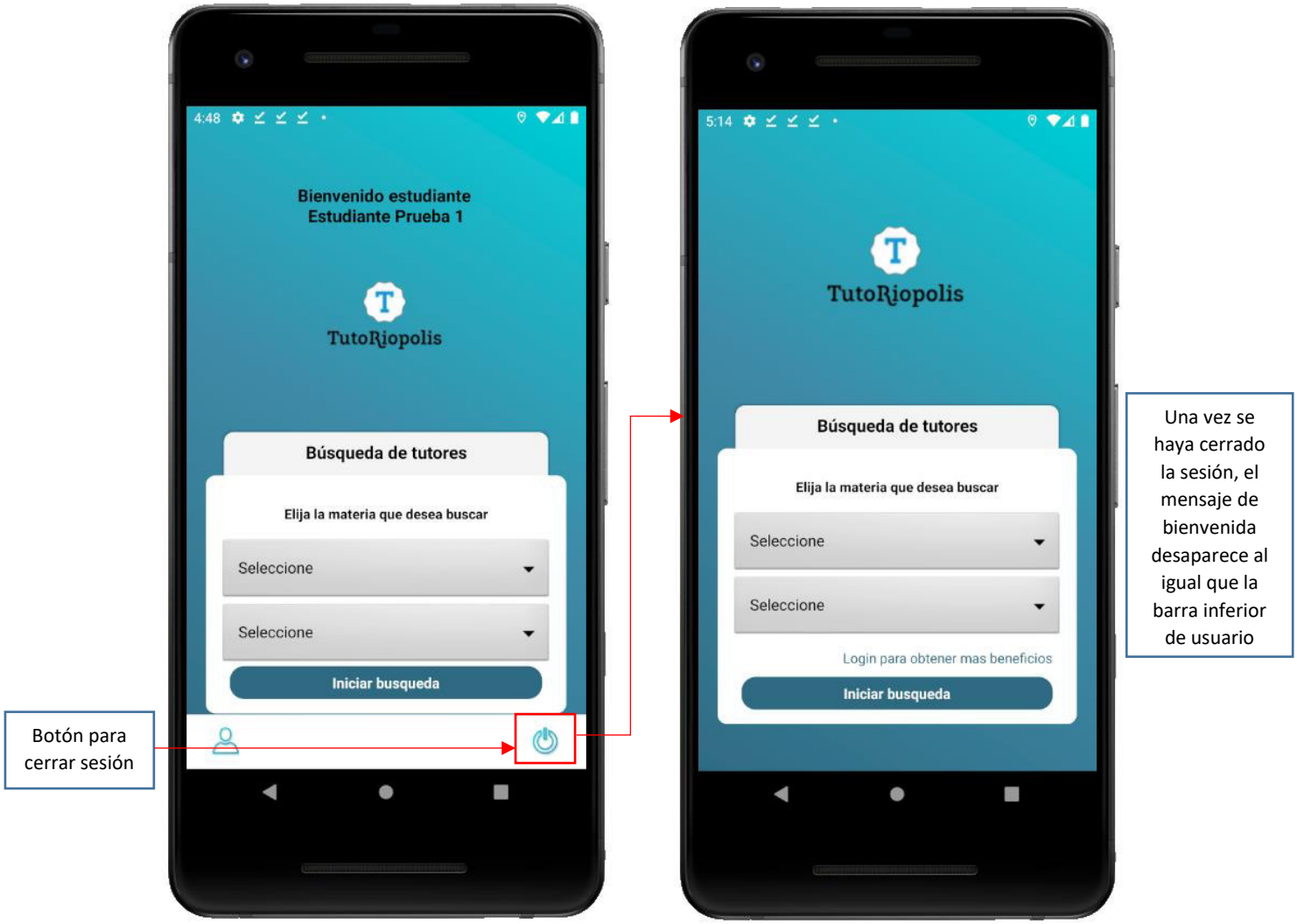

Figura 3.37. Cerrar sesión de una cuenta Estudiante

3.1.3. PRUEBAS DE ACEPTACIÓN – ITERACIÓN 3

3.1.3.1. Búsqueda de Tutores

Procedimiento: En este escenario se describe cómo realizar una búsqueda de tutores, mediante los filtros “Nivel de Educación”, “Materia” y la ubicación geográfica en donde se realiza la búsqueda. Todos estos factores afectarán los resultados, que se desplegarán en un fragmento de mapa, con la lista de los cinco mejores tutores calificados por los usuarios.

Resultados: Para realizar una búsqueda, es necesario hacer uso de los *spinner* que se encuentran en el *Activity.* El primer *spinner* permite elegir el “Nivel de Educación” de una materia, que puede ser: “Bachillerato” o “Educación Superior”. Dependiendo de la opción seleccionada en el primer *spinner*, se desplegarán en el segundo *spinner*, las materias que se encuentran disponibles para ese “Nivel de Educación”. El botón “Iniciar búsqueda”, permitirá visualizar en el mapa los resultados basados en la posición geográfica.

En la **Figura 3.*38***, se muestra cómo realizar búsquedas desde un usuario Estudiante no registrado.

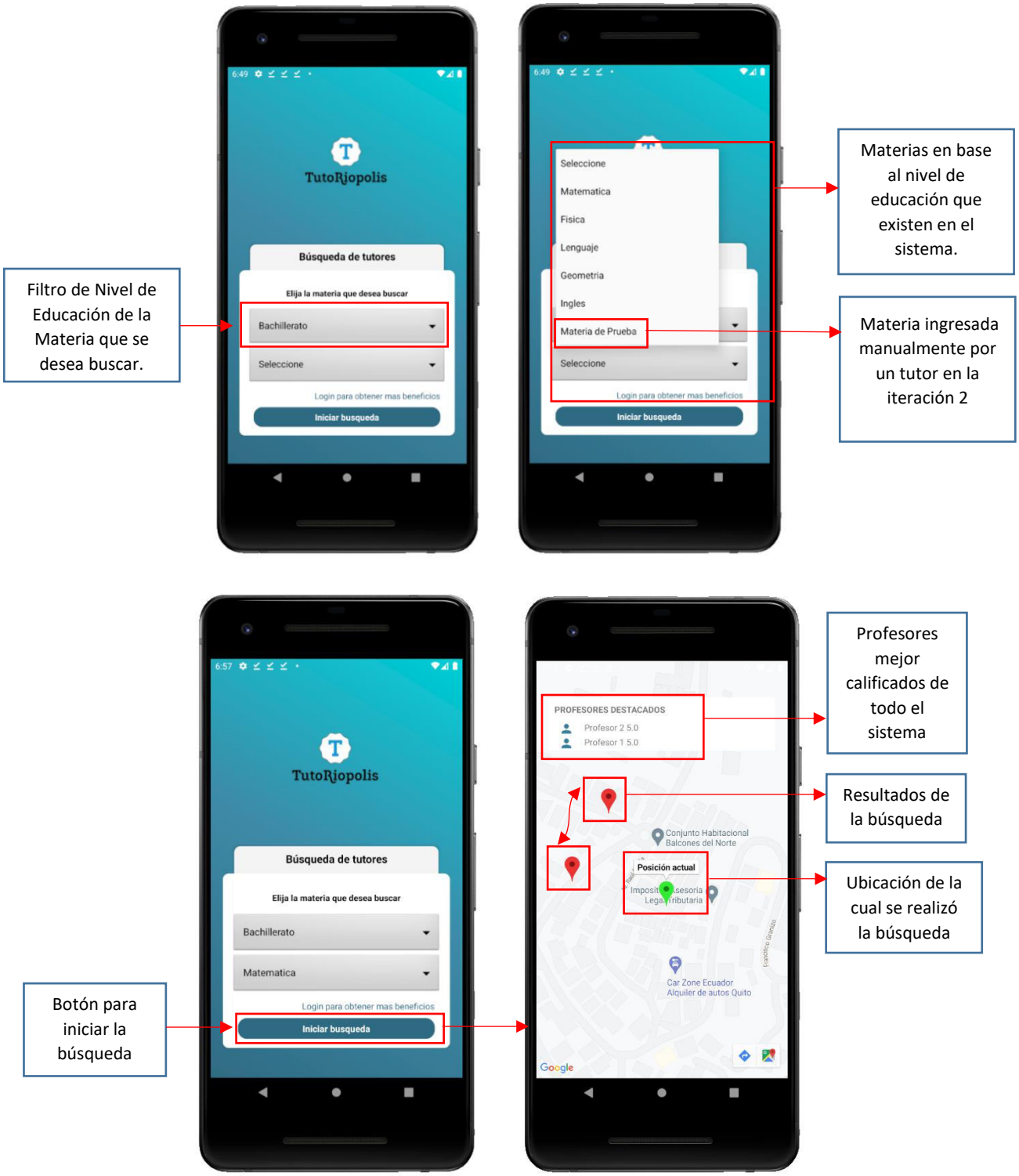

Figura 3.38. Búsqueda de tutores por parte de un usuario Estudiante no registrado

El usuario Estudiante no registrado, no podrá visualizar los perfiles que arrojo la búsqueda de tutores, y al intentar ingresar en uno de ellos, será dirigido a la *Activity* de búsqueda, hasta que realice su registro en el sistema, como se muestra en el **Figura 3.*39***.

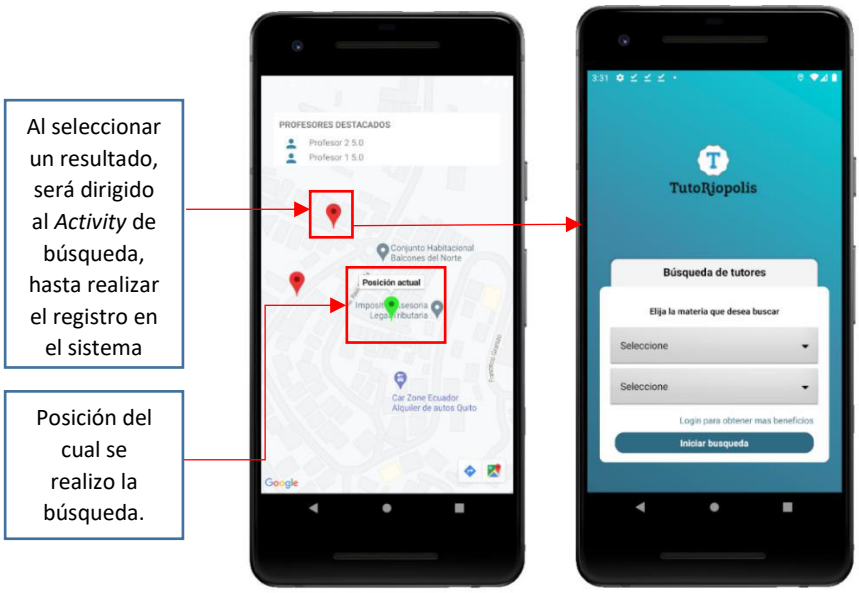

Figura 3.39. Visualización de un perfil tutor, por parte de un usuario Estudiante no registrado

En la **Figura 3.*40***, se presenta la búsqueda por parte de un usuario Estudiante registrado, el mismo que podrá visualizar la información disponible en el perfil del tutor, como se muestra en la **Figura 3.*41***.

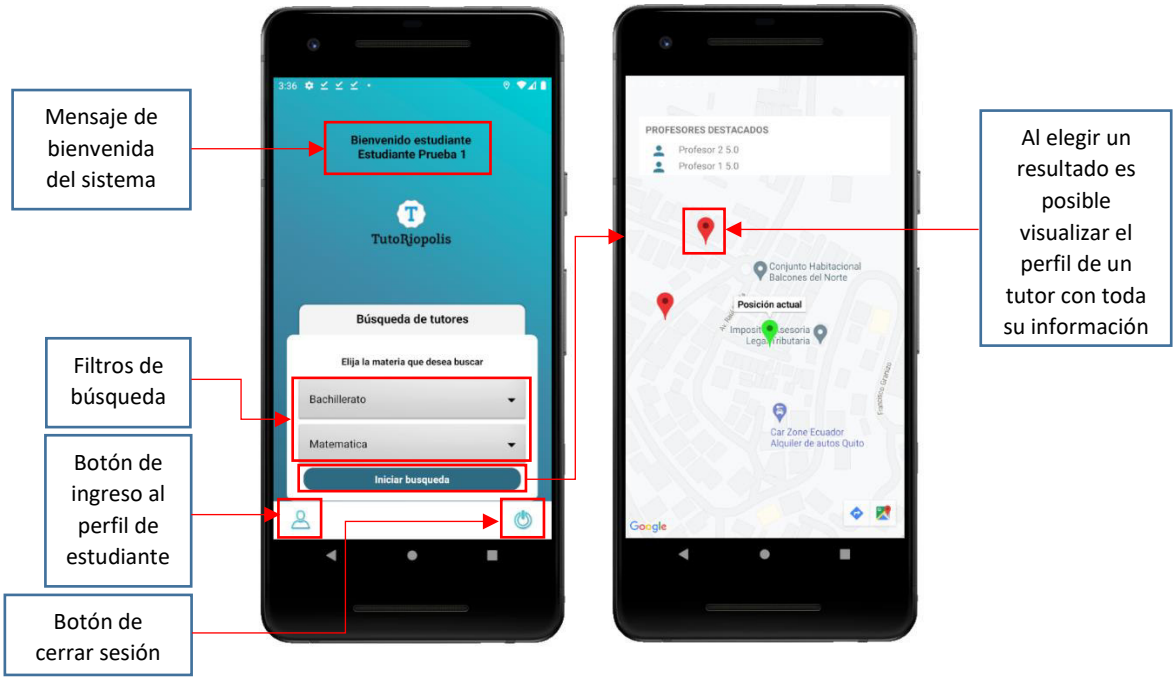

Figura 3.40. Búsqueda de tutores por parte de un usuario Estudiante registrado

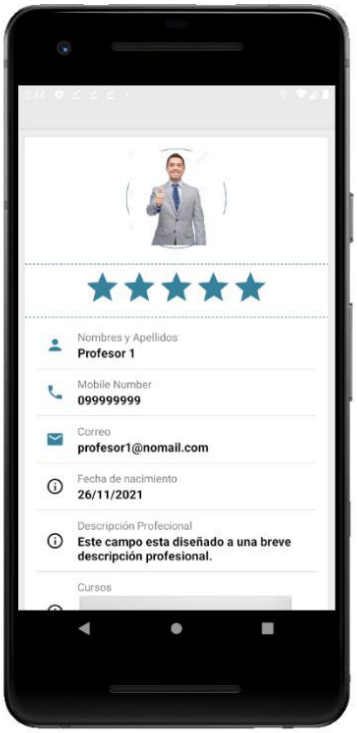

Figura 3.41. Visualización del perfil Tutor por parte del usuario Estudiante registrado

3.1.4. PRUEBAS DE ACEPTACIÓN – ITERACIÓN 4

3.1.4.1. Calificaciones para el Usuario Tutor

Procedimiento: Un usuario estudiante registrado puede o no asignarle una calificación al tutor, la misma que representará la calidad del servicio de la tutoría, se puede calificar a un tutor asignándole una estrella, siendo esta la más baja o cinco estrellas, siendo la calificación más alta. Todo tutor, inicia con una calificación de cinco estrellas y el promedio de todas las calificaciones obtenidas se mostrará en el perfil.

Resultados: En la **Figura 3.*42***, se muestra como asignar una calificación a un usuario tutor. Las calificaciones solo pueden ser otorgadas por los usuarios Estudiantes registrados y cada calificación afecta en su promedio general dentro del sistema. Un promedio alto, ayuda al tutor a ser parte de la lista de los cinco tutores mejor puntuados al realizar una búsqueda.

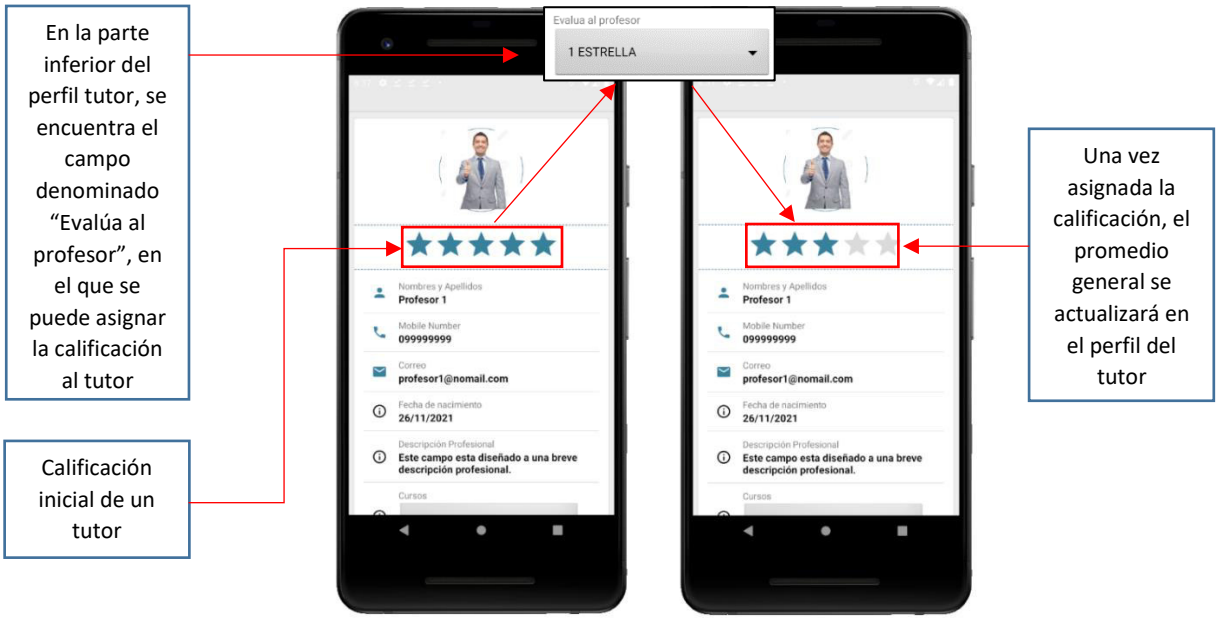

Figura 3.42. Asignación y cálculo promedio de la calificación general de un Tutor

3.1.4.2. Comentarios para Tutores

Es posible agregar un comentario en el perfil de un tutor, el comentario podrá ser visualizado, por el tutor dueño del perfil y por los usuarios estudiantes que ingresen a visualizar dicho perfil, como se muestra en la Figura 3.43.

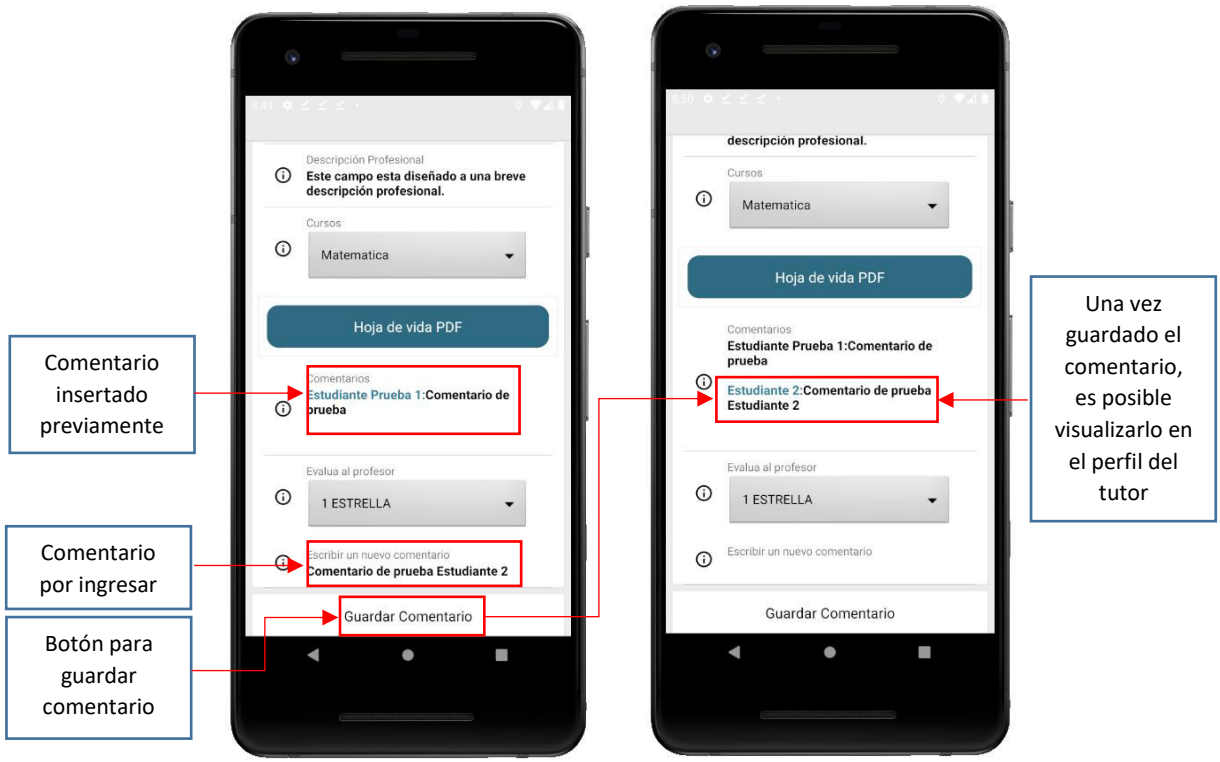

Figura 3.43. Asignar comentario a un usuario Tutor

3.2. FALLOS ENCONTRADOS EN EL SISTEMA

La mayor cantidad de fallos se encontraron en la ejecución de la aplicación móvil. Los elementos gráficos como *spinners*, deben actualizarse en función de la selección de otro *spinner* o al momento de guardar cambios. Por lo que se debió implementar el método `notifyDataSetChanged()`, y de esta manera se logró actualizarlos instantáneamente.

3.2.1. LISTADO DE FALLOS Y CORRECCIÓNES

En la Tabla 3.1, se presenta la lista de fallos más importantes encontrados en el desarrollo del sistema prototipo.

Tabla 3.1. Listado de Fallos y Correcciones

APLICACIÓN	FALLA	SOLUCIÓN
Android	Error en el perfil de Tutores al momento de recuperar los cursos registrados. Ya que se visualizaban todos los cursos.	Se realizó un filtro por identificador único de profesor, y de esta manera se visualizan los cursos, de cada Tutor en específico.
Android	Error al momento de recuperar la fecha de nacimiento, debido al formato.	Se implementó un dateTimePicker(), de esta manera se corrigió el problema de formato de fecha.
Android	La carga de la imagen tomaba demasiado tiempo, por lo que no era posible visualizarla al entrar a un perfil de forma inmediata.	Se lo implementó como tarea asíncrona, de esta manera la carga de la imagen se la realiza por *background*.
Servicio	Al filtrar los usuarios Tutores por nivel escolar y curso, se obtenía el identificador único, mas no los atributos.	Con la lista de identificadores, se hizo uso de un bucle `for`, para comparar los identificadores y guardar el objeto profesor con sus atributos.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Una vez terminado el desarrollo del sistema prototipo y realizadas las pruebas de aceptación de la aplicación móvil, se concluye que el sistema puede agilizar de manera considerable la búsqueda de tutores, de tal manera que permite encontrar un tutor cerca de cualquier ubicación que se requiera, siempre y cuando existan tutores registrados cerca de dicha ubicación.

- El sistema al estar basado en microservicios permite menorar riesgos de fallos, ya que, si un microservicio no se encuentra funcionando correctamente, no afectará a toda la aplicación y es posible que el sistema siga trabajando, además, detectar el fallo se vuelve más sencillo y facilita considerablemente volver a poner en funcionamiento un microservicio dado de baja.

- El sistema es altamente escalable, los módulos aplicados en cada microservicio lo hacen fácil de replicarse permitiendo que la aplicación crezca de manera más ágil y rápida en base a la necesidad del negocio.

- El sistema al estar alojado en una plataforma Cloud como lo es Amazon permite un escalamiento en procesamiento y memoria mucho más sencillo ya que no es necesaria una implementación física, basta con mejorar el plan de la instancia EC2 en los servicios AWS. Menorando costos de implementación, mantenimiento y administración de manera considerable.

- El hacer uso de pruebas de aceptación en cada módulo implementado, permitió identificar errores y a su vez corregirlos de manera anticipada en el desarrollo del sistema prototipo.

4.2. RECOMENDACIONES

- Actualmente todas las funciones se realizan desde la aplicación móvil, ya sean administración del sistema y de contraseñas de usuario. La implementación de estas funciones en un servidor web podría agilizar este proceso.

- La aplicación está diseñada para ser descargada y utilizada exclusivamente por los usuarios del sistema operativo Android, la misma aplicación se podría replicar para usuarios de IOS, de esta manera se ampliaría el alcance de usuarios.

- El servicio se encuentra alojado en dos microservicios, separando las funciones de autenticación de las de registro y búsqueda. La implementación de un tercer

microservicio para separar la búsqueda del registro de usuarios, permitiría una independencia superior, minimizando los riesgos de fallo. Debido a que, si un microservicio falla, los demás seguirán funcionando con normalidad.

- La implementación de un canal de comunicación directo entre usuarios estudiantes y tutores, como un módulo de intercambio de mensajes, permitiría una comunicación más fluida, evitado la publicación de información.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] J. Fernández, "Introducción a las Metodologías Ágiles" [Online]. Disponible en: https://www.exabyteinformatica.com/uoc/Informatica/Tecnicas_avanzadas_de_ingenieria_de_software/Tecnicas_avanzadas_de_ingenieria_de_software_(Modulo_3).pdf [Accedido: 05-jun-2020].

[2] G. Fernández, "Introducción a *Extreme Programming*" [Online]. Disponible en: https://aalbertovargasc.files.wordpress.com/2011/07/presentacion-xp.pdf [Accedido: 1-jun-2020].

[3] R. Vargas and J. Maltes, "Programación en Capas" [Online]. Disponible en: http://www.di-mare.com/adolfo/cursos/2007-2/pp-3capas.pdf [Accedido: 1-jun-2020].

[4] Spring Boot, "*Using Spring Boot*" [Online]. Disponible en: https://docs.spring.io/spring-boot/docs/current/reference/html/ [Accedido: 13-oct-2020].

[5] Amazon EC2, "*Documentation of Amazon Elastic Compute Cloud*" [Online]. Disponible en: https://docs.aws.amazon.com/ec2/index.html [Accedido: 13-oct-2020].

[6] R. Hernández and M. Morales, "Dispositivos móviles en la educación" [Online]. Disponible en: https://www.galileo.edu/ivn/noticias/dispositivos-moviles-en-la-educacion/ [Accedido: 16-may-2020].

[7] G. Chanchi, W. Campo, J. Amaya and J. Arciniegas, "Esquema de Servicios par Televisión Digital Interactiva basados en el protocolo REST-JSON" [Online]. Disponible en: https://www.seer.ufrgs.br/cadernosdeinformatica/article/view/v6n1p233-240/11807] [2011] [Accedido: 01-Oct-2021].

[8] G. Duarte, "Arquitectura Propuesta para un Servicio Web Completo: Metodología de desarrollo e Implementación" [Online]. Disponible en: https://repository.udistrital.edu.co/bitstream/handle/11349/2809/DuarteVegaGabrielEduardo2016.pdf?sequence=1 [Accedido: 01-Oct-2021].

[9] E. Marini, "El modelo Cliente Servidor" [Online]. Disponible en: https://www.linuxito.com/docs/el-modelo-cliente-servidor.pdf [Accedido: 03-Oct-2021].

[10] X. Rea, T. Mancero, D. Rosero, and D. Imbaquingo, "Web Services REST: Una Revolución en el Acceso de Datos" [Online]. Disponible en: https://www.linuxito.com/docs/el-modelo-cliente-servidor.pdf [Accedido: 03-Oct-2021].

[11] M. Castro, D. Sánchez, J. Farfán, D. Castro, D. Cándido, and A. Vargas, "Aplicaciones de Servicios SOAP/REST para funcionalidades existentes en sistemas informáticos provinciales", 2013, [Online]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/94248/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y [Accedido: 03-Oct-2021].

[12] A. Monago, "Servicio de API REST sobre el Framework Spring, Hibernate, JSON Web Token y BBDD Oracle", [Online]. Disponible en: https://idus.us.es/bitstream/handle/11441/89487/TFG-2380-MONAGO.pdf?sequence=1&isAllowed=y [Accedido: 03-Oct-2021].

[13] Y. Fernández, Y. Díaz Y "Patrón Modelo-Vista-Controlador", [Online], Available: https://revistatelematica.cujae.edu.cu/index.php/tele/article/view/15/10 [Accedido: 03-Oct-2021]

[14] J. Sagredo, A. Truebe, M. Martinez, and M. López, "Automatización de la Codificación del Patrón Modelo Vista Controlador (MVC) en Proyectos Orientados a la Web", [Online], Available: https://www.redalyc.org/pdf/104/10423895005.pdf. [Accedido: 03-Oct-2021]

[15] R. Córdova, and B. Cuzco, "Análisis comparativo entre Bases de Datos Relacionales con Bases de Datos No Relacionales", [Online], Available: https://dspace.ups.edu.ec/bitstream/123456789/6977/1/UPS-CT003639.pdf [Accedido: 03-Oct-2021]

[16] The PostgresSQL Global Development Group, "*PostgresSQL Documentation*", 2021, [Online], Available: https://www.postgresql.org/docs/current/intro-whatis.html. [Accedido: 04-Oct-2021]

[17] A. Molina, "Implementación de un Recurso Docente sobre Android para el Autoaprendizaje de un Lenguaje de Programación", [Online], Available: https://idus.us.es/bitstream/handle/11441/28425/TFG-AlfonsoMolina.pdf?sequence=1&isAllowed=y. [Accedido: 04-Oct-2021]

[18] Spring, "*Spring Boot Documentation*", 2021, [Online], Available: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#executable-jar. [Accedido: 04-Oct-2021]

[19] S. Pérez, "Estudio del Framework Spring, Spring Boot y Microservicios", [Online], Available: https://ebuah.uah.es/xmlui/bitstream/handle/10017/45107/TFM_Ramirez_Perez_2020.pdf?sequence=1&isAllowed=y. [Accedido: 04-Oct-2021]

[20] Amazon Web Services, "*AWS Elastic Computing*", 2021, [Online], Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html. [Accedido: 04-Oct-2021]

[21] A. Paz, "Arquitectura de Amazon Elastic Compute Cloud (Amazon EC2) y la utilización del hipervisor XEN 1 Introducción", [Online], Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction. [Accedido: 04-Oct-2021]

[22] P. Letelier, C. Penadés, "Metodologías ágiles para el desarrollo de Software: eXXreme Programming (XP)" [Online], Available: https: https://www.cyta.com.ar/ta0502/v5n2a1.htm. [Accedido: 04-Oct-2021]

[23] M. Calero, "Un explicación de la programación extrema (XP)", [Online], Available: https://suriweb.com.ar/wp/tecnologia/wp-content/uploads/sites/18/2019/03/Una-explicacion-de-XP.pdf. [Accedido: 04-Oct-2021]

ANEXOS

ANEXO A. Procedimiento para instalar los ambientes de desarrollo

ANEXO B. Procedimiento para crear una Instancia EC2 en los servicios Web de Amazon

ANEXO C. Procedimiento para crear archivos APK y JAR

ANEXO A

A. PROCEDIMIENTO PARA INSTALAR LOS AMBIENTES DE DESARROLLO

A.1. INSTALACIÓN Y CONFIGURACIÓN DE ANDROID STUDIO

En primer lugar, es necesario ir a la página web oficial de Android para realizar la descarga del instalador como se muestra en la Figura A.1.

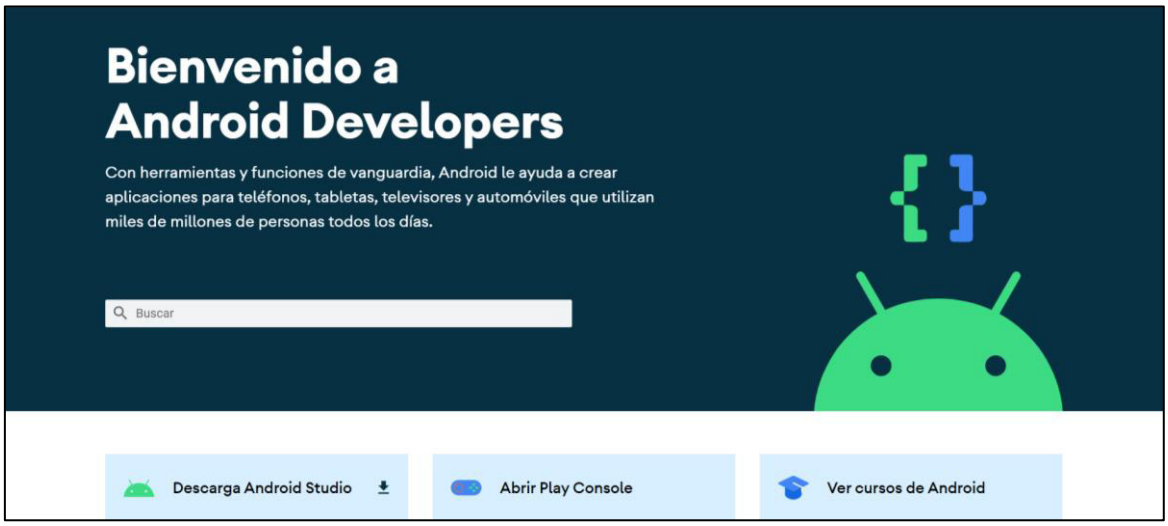

Figura A.1. Página Web oficial Android

Una vez se ha completado la descarga, es necesario ejecutar el instalador para iniciar con la configuración de instalación como se muestra en la **Figura A.*2***.

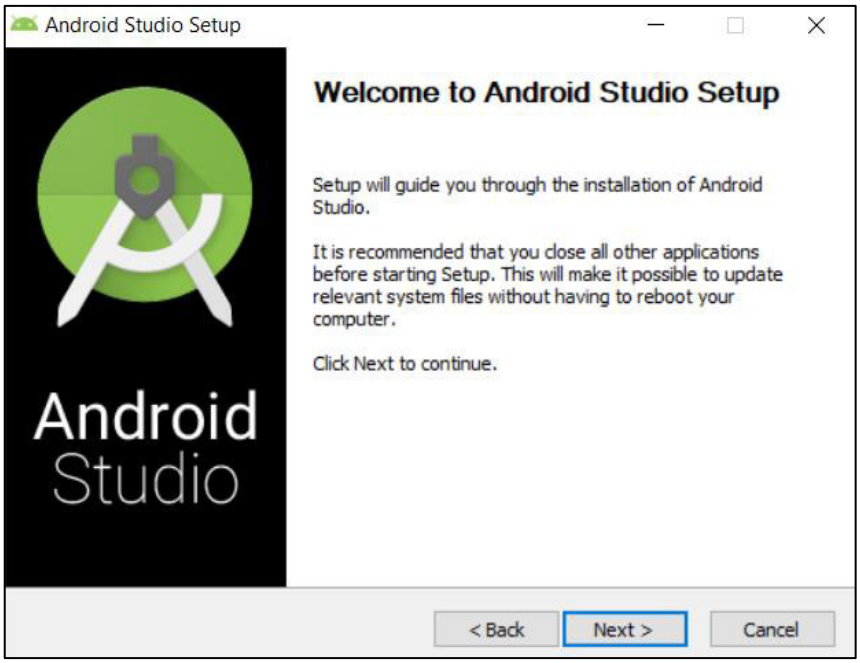

Figura A.2. Ventana de Bienvenida del instalador de Android Studio

Al presionar en el botón “Next”, se desplegará las opciones de instalación del IDE de Android Studio y del dispositivo virtual. En este caso se elegió ambas opciones, de esta manera se obtendrá el ambiente de desarrollo y el dispositivo virtual que permitirá probar la funcionalidad de la Aplicación y presionamos “Next” como se muestra en la **Figura A.*3***.

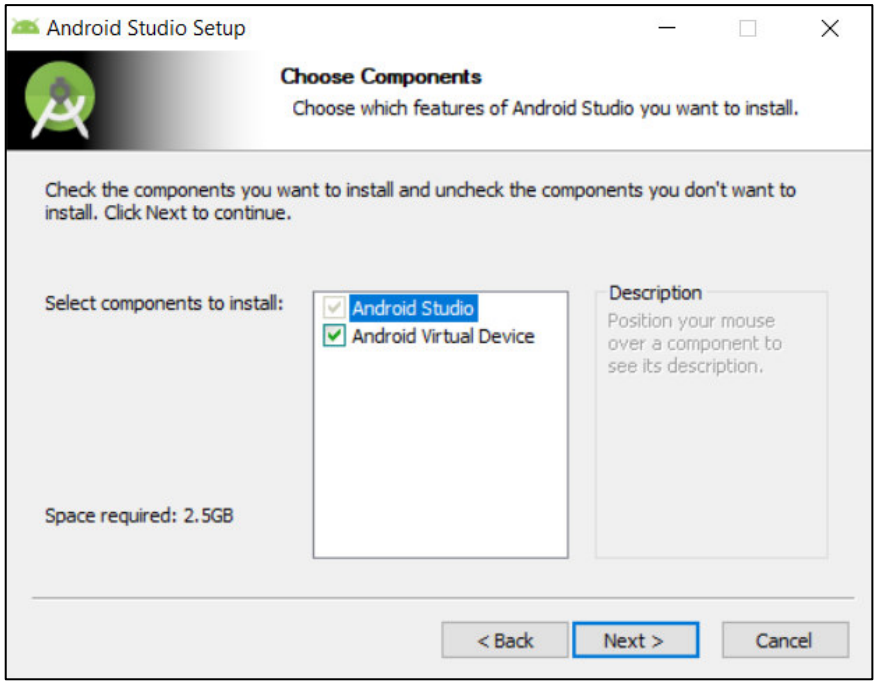

Figura A.3. Ventana de componentes a instalar

Es necesario contar con al menos 500 MB de espacio de almacenamiento, por lo que se debe especificar el *path* para la instalación y presionamos “Next”, como se muestra en la **Figura A.*4***.

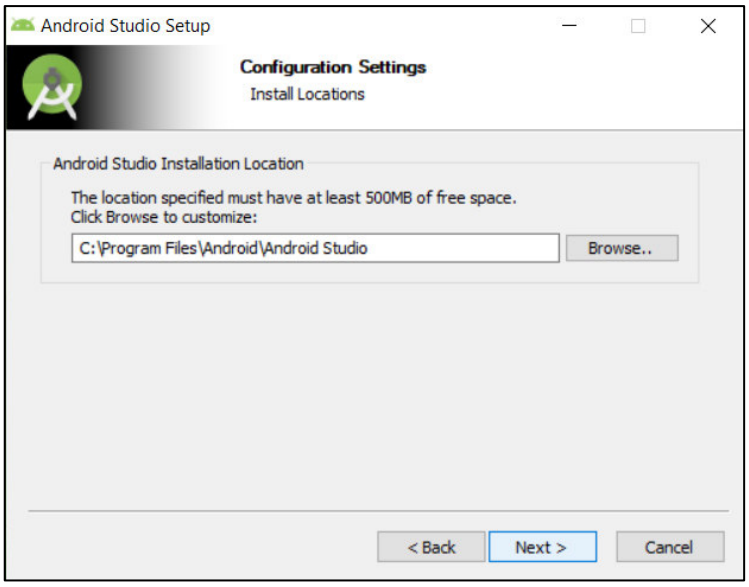

Figura A.4. *Path* de localización en memoria para la instalación de Android Studio

En este paso, se selecciona la carpeta del menú Inicio en la que se quiere crear y agregar un nombre al acceso directo, no se elegirá ningún menú y se mantendrá el nombre por defecto. Finalmente, se debe presionar el botón “Install” y esperar hasta que se complete la instalación, como se muestra en la Figura A.5. Al completar la instalación se mostrará la pantalla de la **Figura A.*6*** y se iniciará el entorno de desarrollo.

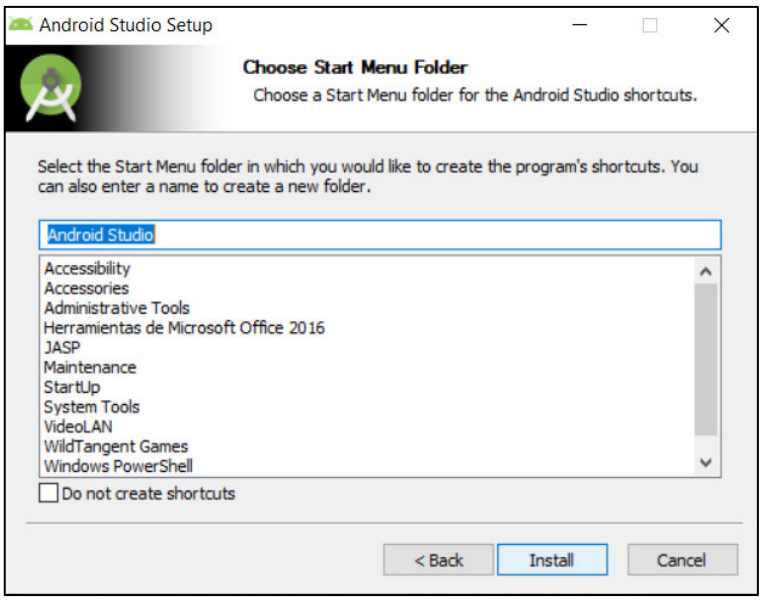

Figura A.5. Creación del Acceso Directo de Android Studio

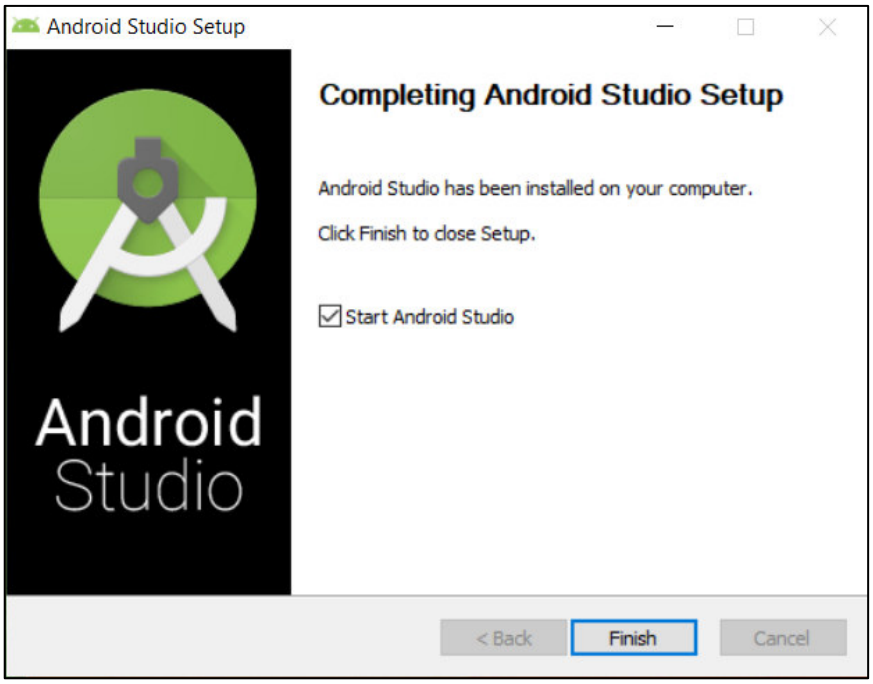

Figura A.6. Ventana para finalizar el proceso de instalación

En la **Figura A.*7*** se muestra la pantalla de "Bienvenida" de Android Studio, para continuar con la configuración es necesario presionar "Next".

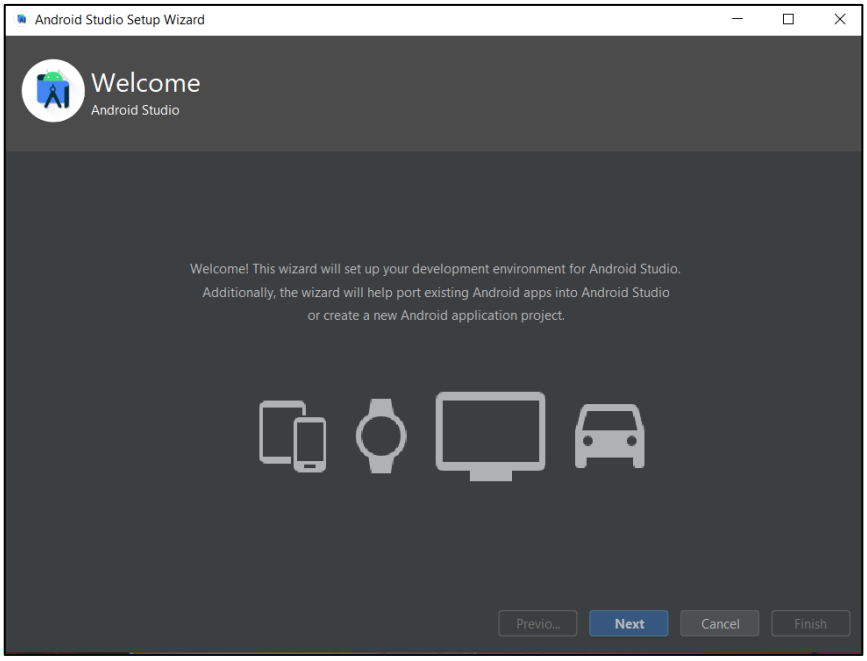

Figura A.7. Pantalla de "Bienvenida" del ambiente de desarrollo de Android Studio

En la **Figura A.*8*** se muestra la ventana en la cual se pide el tipo de *setup* que se desea instalar. Se elige la opción "*Standard*"

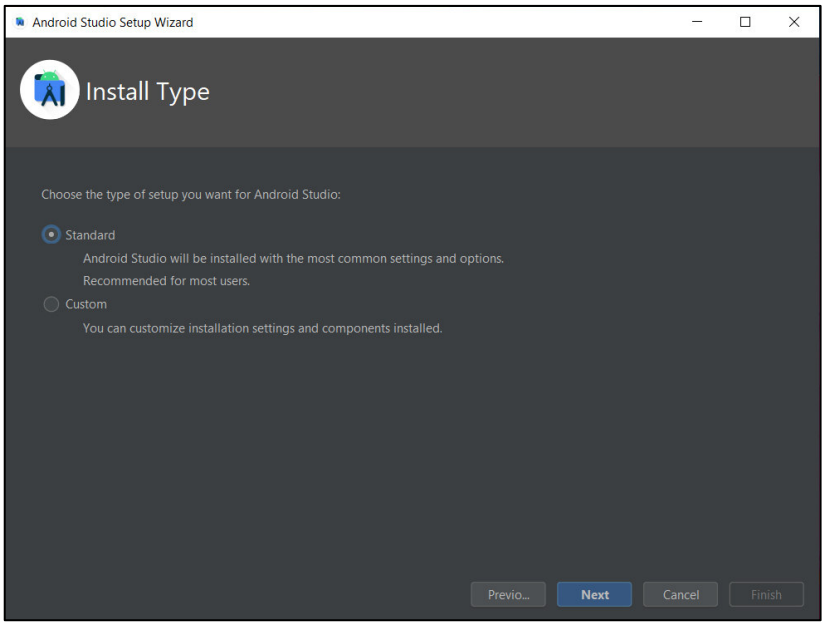

Figura A.8. Ventana de opciones del tipo de instalación

Al presionar “Next”, se permite seleccionar el tema que tendrá la IDE, es posible seleccionar el tema “Light” o “Darcula” como se muestra en la **Figura A.*9***.

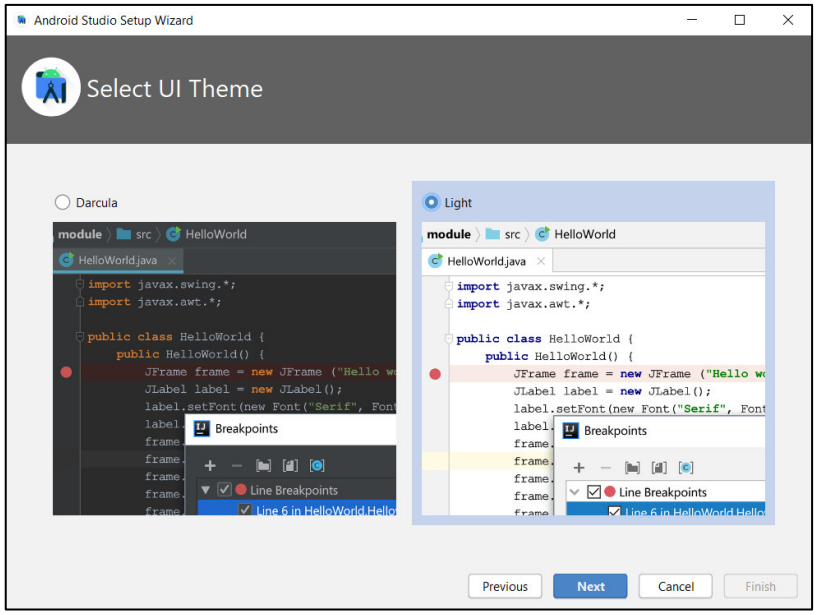

Figura A.9. Ventana de opciones de temas para el ambiente de desarrollo

En la **Figura A.*10***, se puede visualizar un resumen de las configuraciones elegidas previamente, si no se desean hacer cambios se presiona en el botón “Finish”.

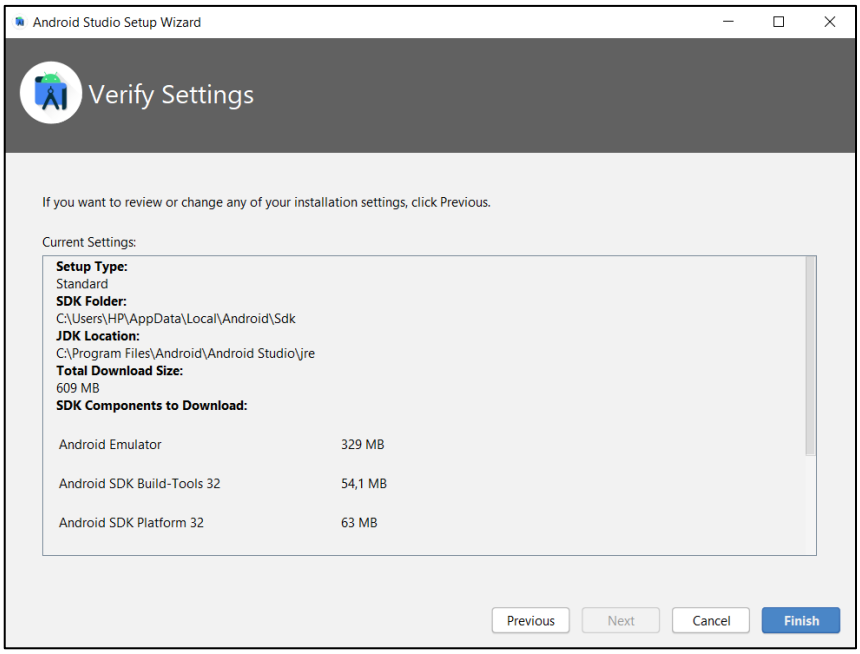

Figura A.10. Ventana de finalización de configuración del ambiente de desarrollo de Android Studio

A.2. INSTALACIÓN DEL FRAMEWORK SPRING

En primer lugar, es necesario ir a la página web oficial de Spring Tools para realizar la descarga del archivo Spring como se muestra en la **Figura A.*11***.

Figura A.11. Página Web Oficial del *Framework Spring Boot*

Una vez realizada la descarga, se obtendrá un archivo comprimido, que se muestra en la **Figura A.*12***.

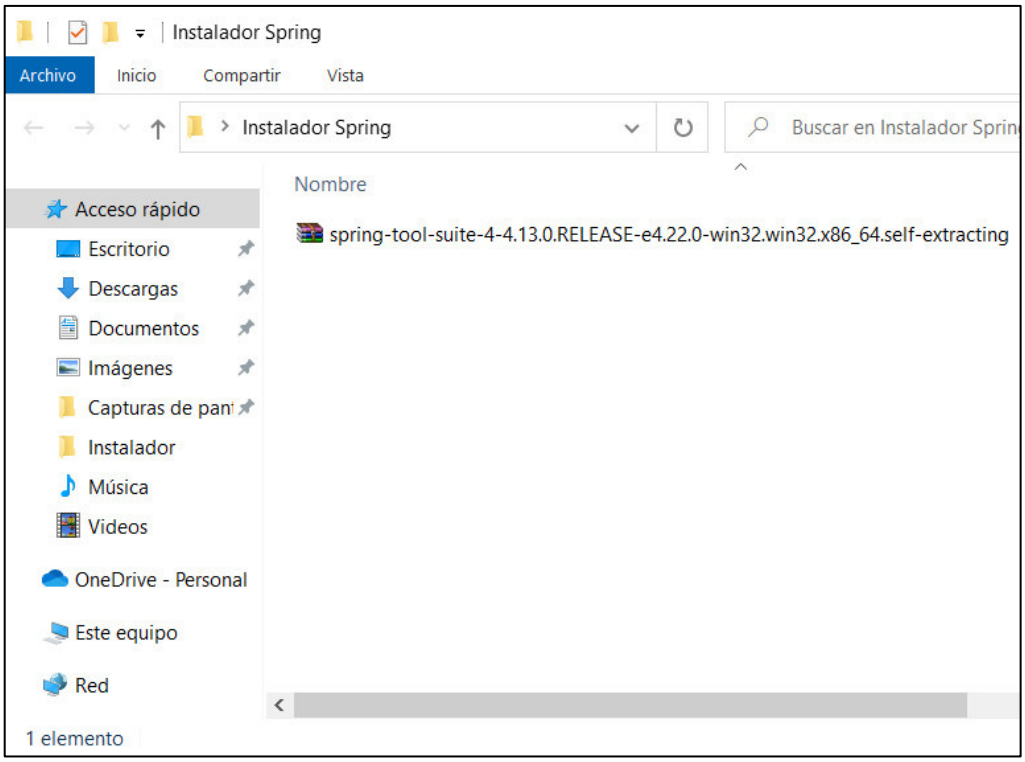

Figura A.12. Descarga del archivo comprimido desde la página web *Spring Tools*

En la **Figura A.*13***, se muestra el contenido del archivo descargado una vez descomprimido.

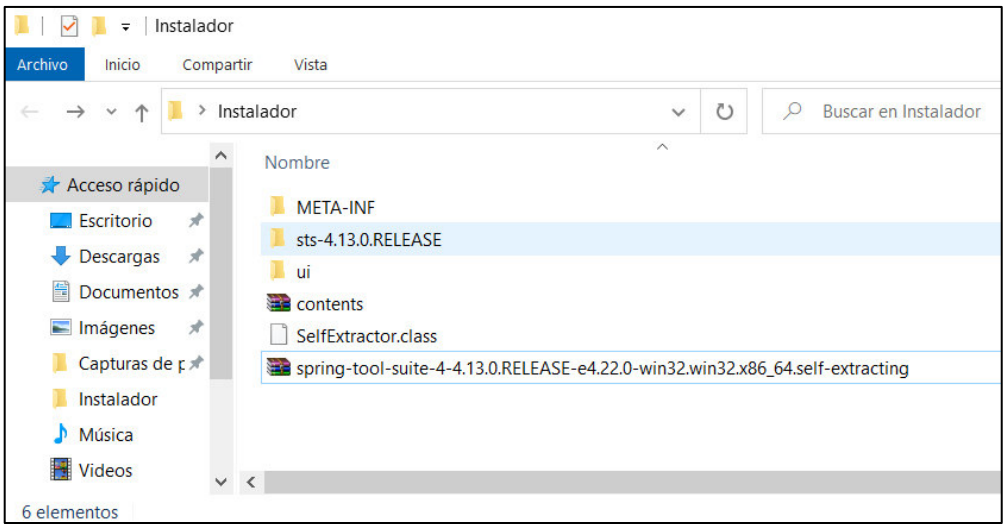

Figura A.13. Contenido del archivo descargado descomprimido

Es necesario descomprimir el archivo "spring-tool-suit-4-4.13.0.RELEASE-e4.22.0-win32.x86_64.self-extracting" como se muestra en la **Figura A.*14***.

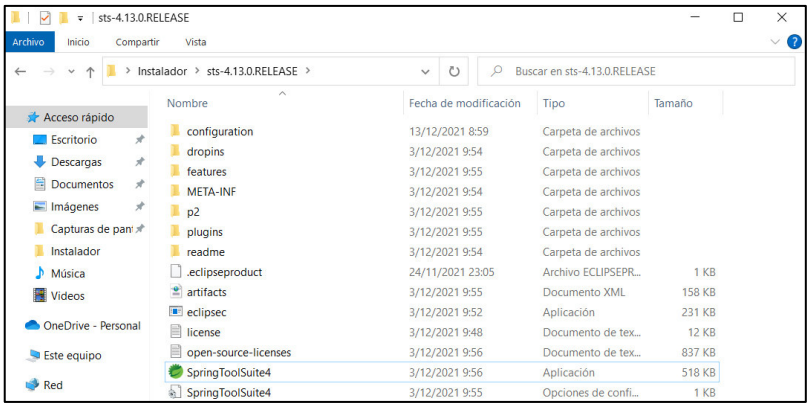

Figura A.14. Descompresión del archivo “spring-tool-suit-4-4.13.0.RELEASE-e4.22.0-win32.x86_64.self-extracting”

Una vez se visualiza el contenido del archivo, es necesario pulsar sobre “SpringToolSuit4”, que desplegará la ventana en donde se elegirá el directorio en el cual se ubicará el *workspace,* como se muestra en la **Figura A.*15***.

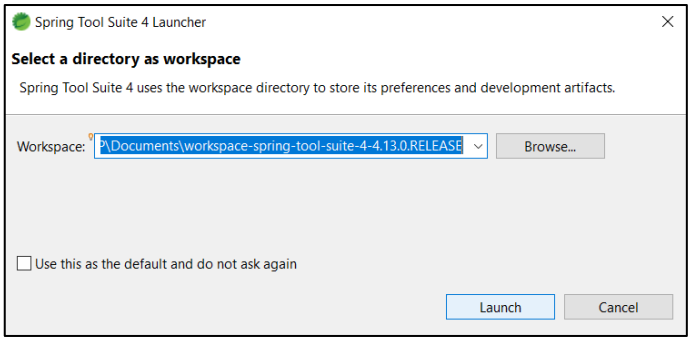

Figura A.15. Pantalla de ubicación del workspace

Finalmente, se debe pulsar “Launch” y se desplegara el software Spring, como se muestra en la **Figura A.*16***.

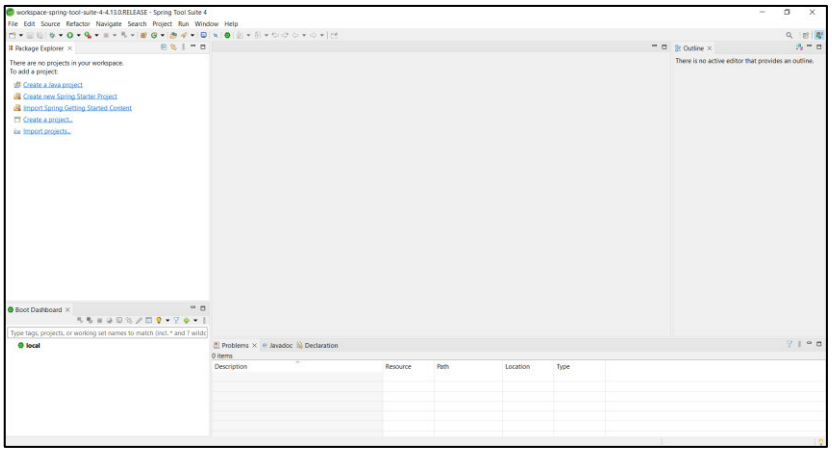

Figura A.16. Ventana del ambiente de desarrollo *Spring Boot*

A.3. INSTALACIÓN DEL MOTOR DE BASE DE DATOS POSTGRESQL

En primer lugar, es necesario ir a la página web oficial de Spring Tools para realizar la descarga del archivo Spring como se muestra en la Figura A.17.

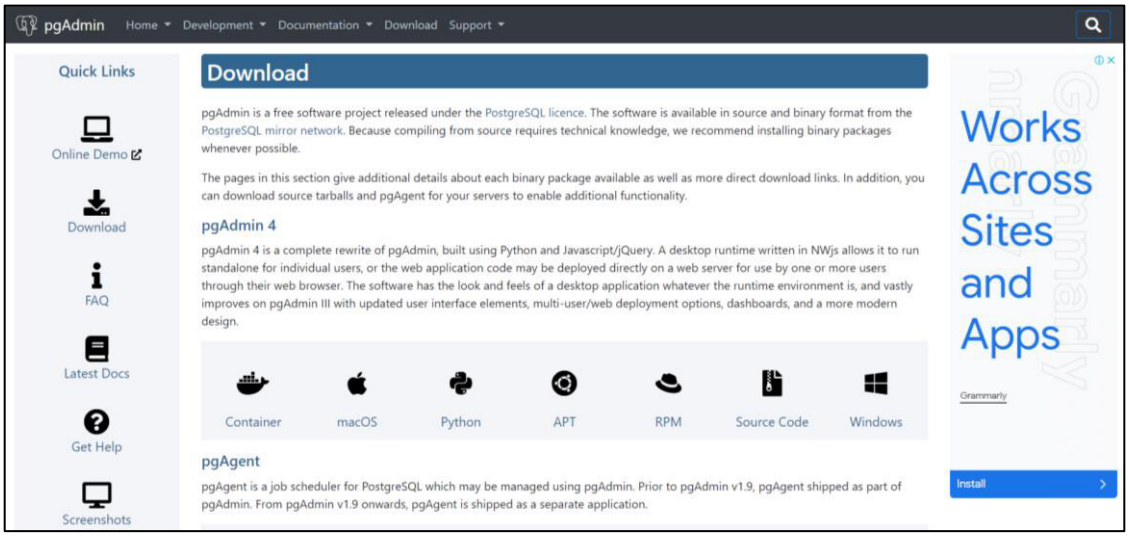

Figura A.17. Página Web Oficial del motor de base de datos PostgreSQL

Una vez descargado el instalador PostgreSQL, es necesario ejecutarlo. Al hacerlo se visualizará la ventana de “Bienvenida” del instalador, como se muestra en la Figura A.18.

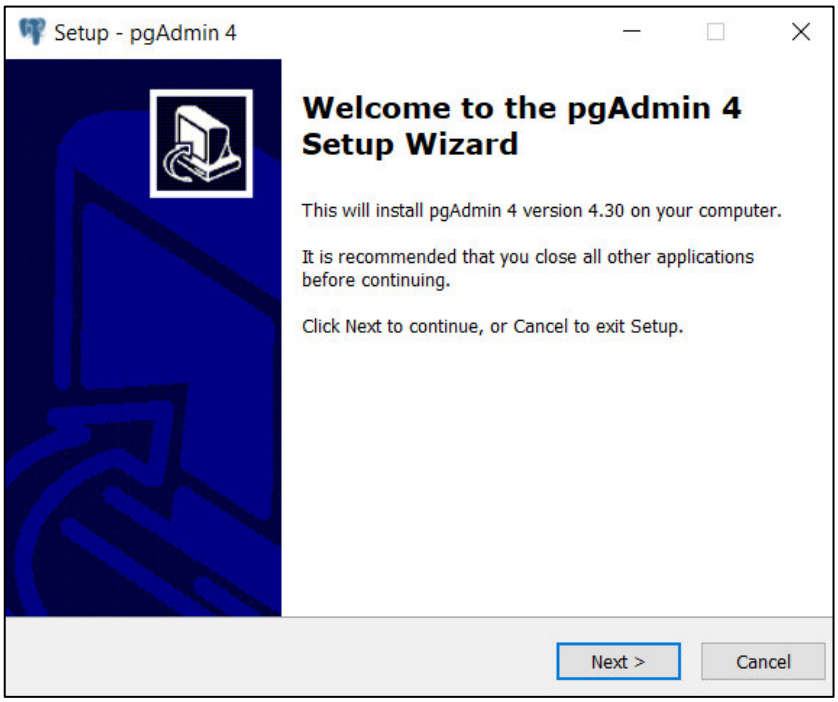

Figura A.18. Ventana de “Bienvenida” del Instalador del motor de base de datos PostgreSQL

Al pulsar el botón “Next”, se desplegará la ventana de “Términos y Condiciones”, en la cual se debe elegir la opción de “I accept the agreement”, como se muestra en la **Figura A.*19***.

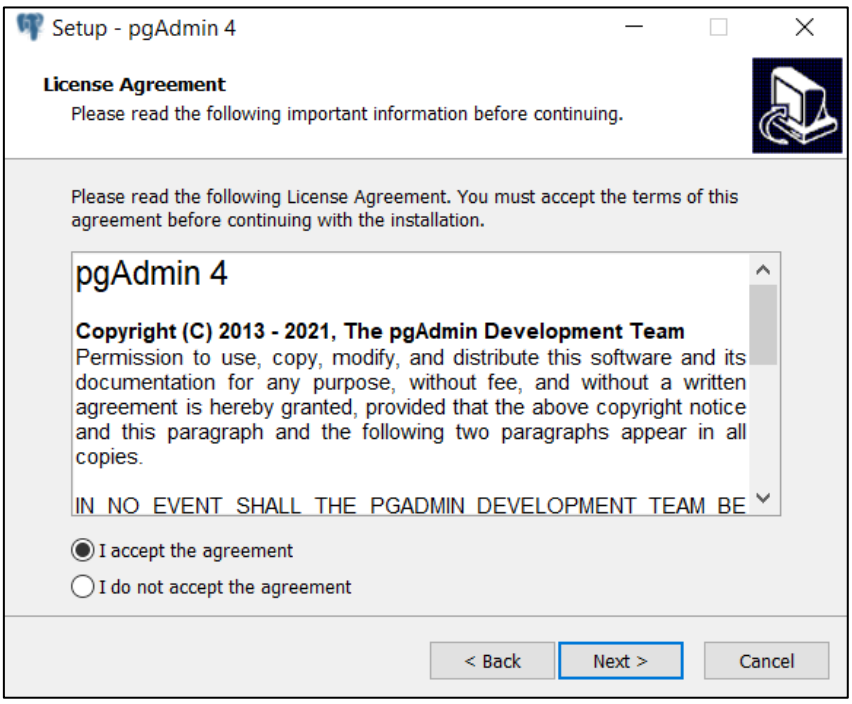

Figura A.19. Ventana de “Términos y Condiciones”

A continuación, se elige el directorio en el cual se instalará el software pgAdmin, como se muestra en la **Figura A.*20***.

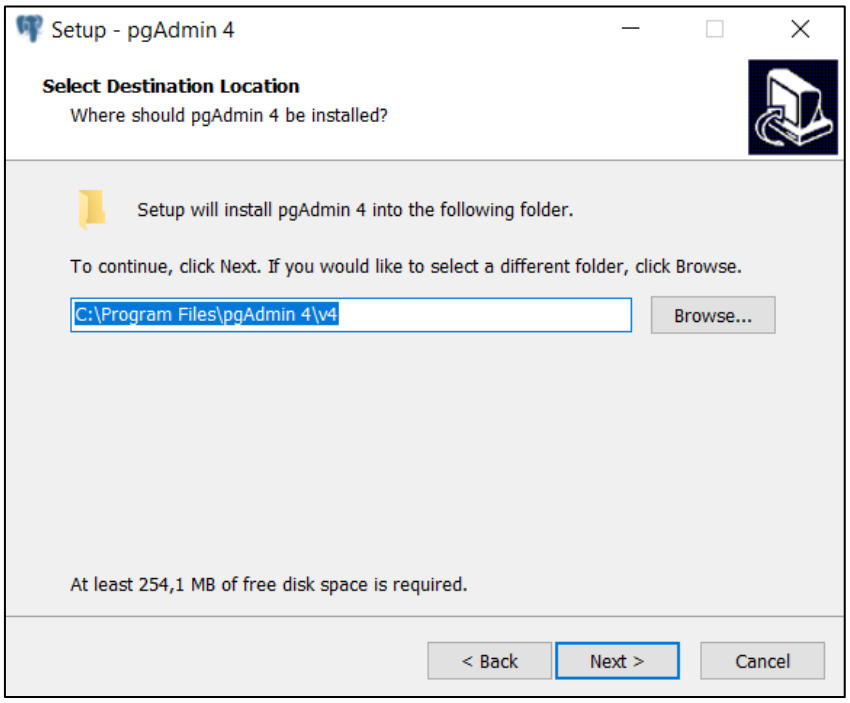

Figura A.20. Pantalla para elegir el directorio de almacenamiento del software pgAdmin

En la **Figura A.*21***, se configura la opción para crear un acceso directo.

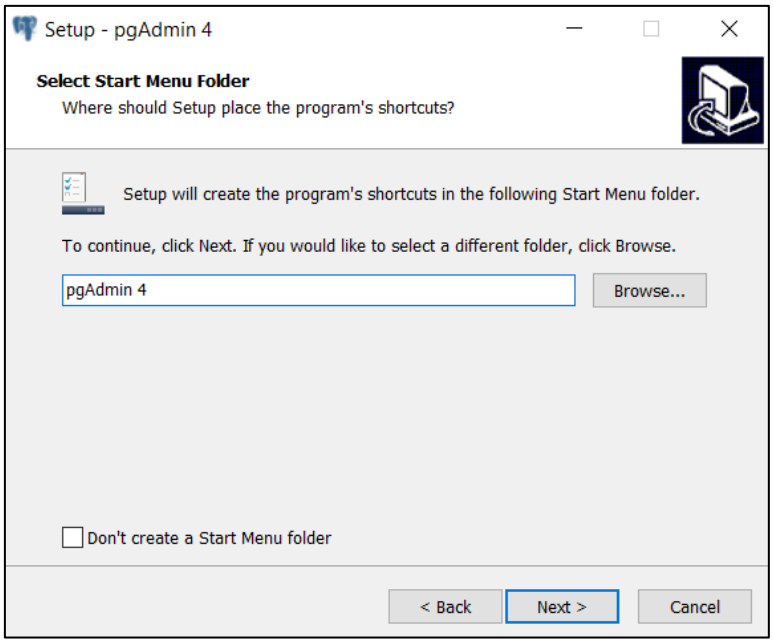

Figura A.21. Ventana para configurar la creación del acceso directo

Al presionar “Next”, se despliega la ventana para confirmar la instalación o revisar alguna configuración previa. Se debe presionar “Install” como se muestra en la **Figura A.*22***. Al finalizar la instalación se mostrará la **Figura A.*23***, que indicara que el motor de base de datos PostgreSQL, ha sido instalado con éxito.

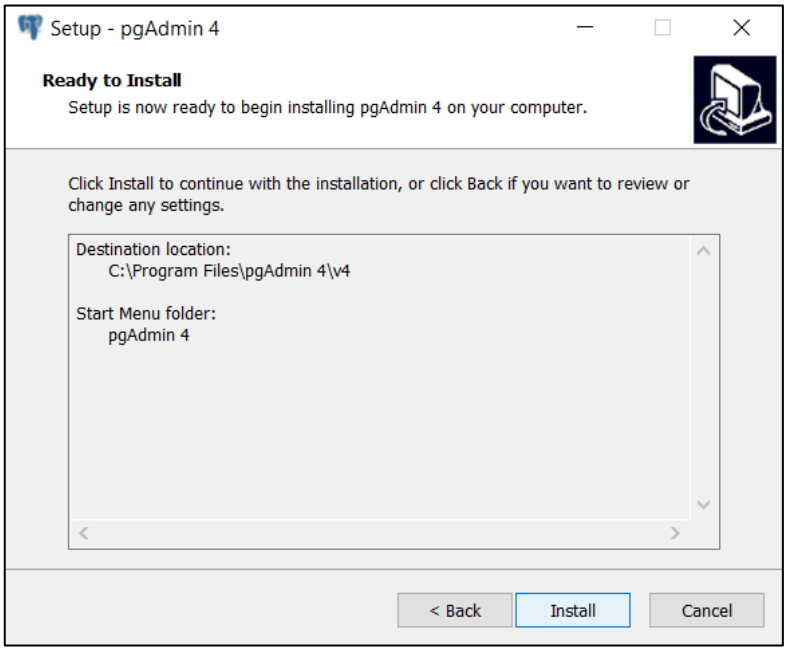

Figura A.22. Pantalla para confirmación de Instalación del motor de base de datos PostgreSQL

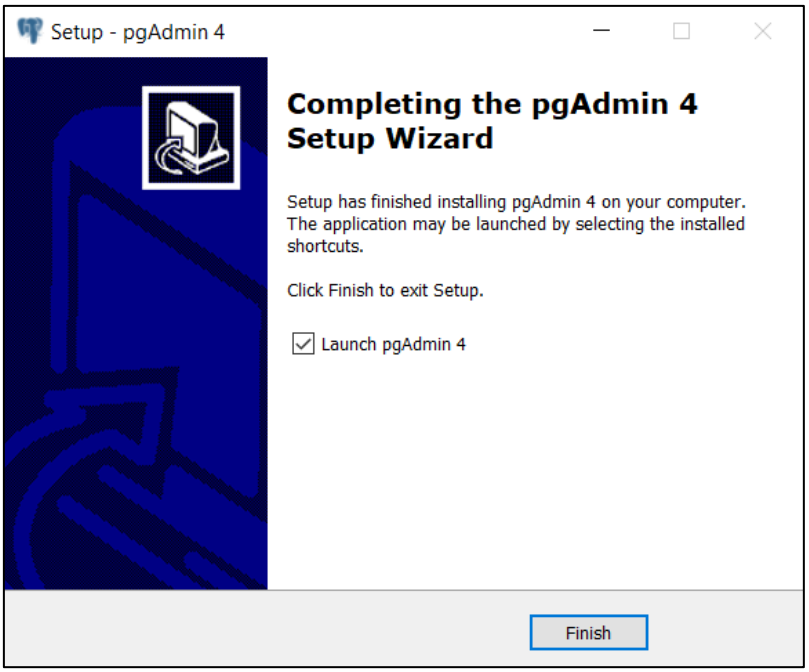

Figura A.23. Pantalla de finalización de Instalación

ANEXO B

B. PROCEDIMIENTO PARA DEPLEGAR LA INSTANCIA EC2

Para crear una instancia en EC2, es necesario tener una cuenta en Amazon. Una vez se realiza el registro, se debe ingresar a la dirección web de Amazon EC2 y autenticarse. Para crear la instancia se debe ir al apartado de “Crear una Solución” y elegir la opción de “Lance una máquina virtual con EC2”, como se muestra en la **Figura B.*1***.

Figura B.1. Pantalla de finalización de Instalación

Es necesario seleciconar el sistema operativo que tendrá la instancia EC2, en este caso se usará “Ubuntu Server 20.04 LTS (HVM), SSD Volume Type” de 64 bits (x86), como se muestra en la **Figura B.*2***.

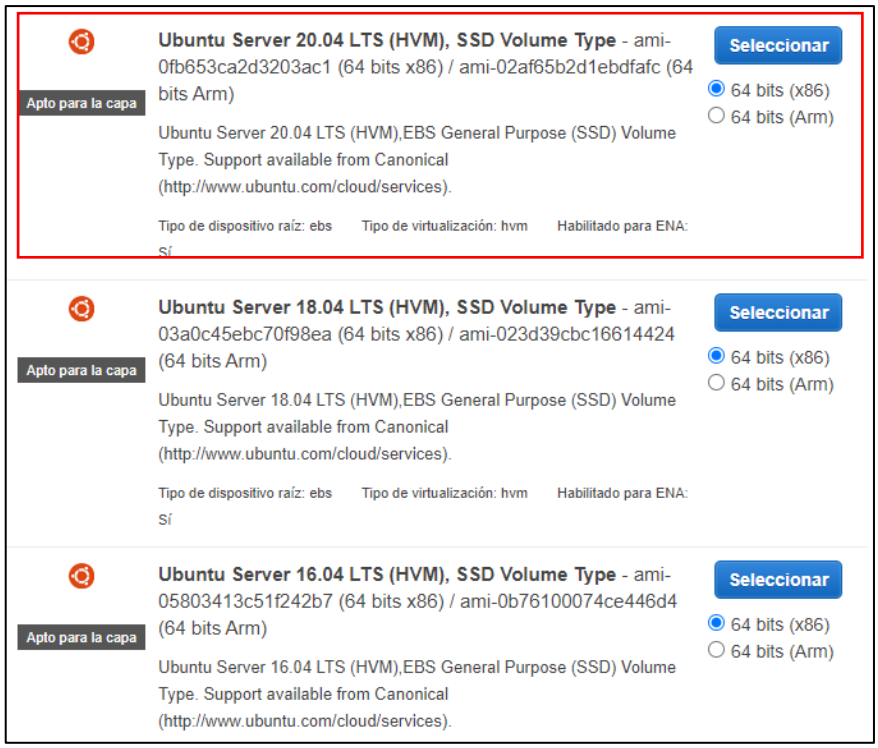

Figura B.2. Selección del Sistema Operativo

El siguiente paso, es seleccionar el “Tipo de Instancia”. En este apartado es posible seleccionar entre varias opciones de combinaciones de CPU, como memoria, almacenamiento y capacidad de red. De esta manera es posible asignar los recursos a la instancia. En este caso, se hará uso de la combinación “t2.micro”, como se muestra en la

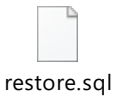

Figura B.*3*. Una instancia del tipo T2, son recomendables para el uso de microservicios, aplicaciones de baja latencia y bases de datos pequeñas.

Seleccionada actualmente: t2.micro (- ECU, 1 vCPU, 2.5 GHz, -, 1 GiB memoria, EBS solo)

	Familia	Tipo	vCPU	Memoria (GiB)	Almacenamiento de la instancia (GB)	Optimizado para EBS disponible	Desempeño de la red	Compatibilidad con IPv6
☐	t2	t2.nano	1	0.5	EBS solo	-	De bajo a moderado	Sí
☑	t2	t2.micro (Apto para la capa gratuita)	1	1	EBS solo	-	De bajo a moderado	Sí
☐	t2	t2.small	1	2	EBS solo	-	De bajo a moderado	Sí
☐	t2	t2.medium	2	4	EBS solo	-	De bajo a moderado	Sí
☐	t2	t2.large	2	8	EBS solo	-	De bajo a moderado	Sí
☐	t2	t2.xlarge	4	16	EBS solo	-	Moderada	Sí
☐	t2	t2.2xlarge	8	32	EBS solo	-	Moderada	Sí
☐	t3	t3.nano	2	0.5	EBS solo	Sí	Hasta 5 gigabits	Sí
☐	t3	t3.micro	2	1	EBS solo	Sí	Hasta 5 gigabits	Sí
☐	t3	t3.small	2	2	EBS solo	Sí	Hasta 5 gigabits	Sí
☐	t3	t3.medium	2	4	EBS solo	Sí	Hasta 5 gigabits	Sí
☐	t3	t3.large	2	8	EBS solo	Sí	Hasta 5 gigabits	Sí

restore.sql

Figura B.3. Tipo de Instancias EC2 disponibles en los servicios web de Amazon

Debido a que la instancia no está destinada a uso comercial, no es necesario agregar más configuraciones. Finalmente, se debe pulsar la opción de “Lanzar Instancia”, como se muestra en la **Figura B.*4***.

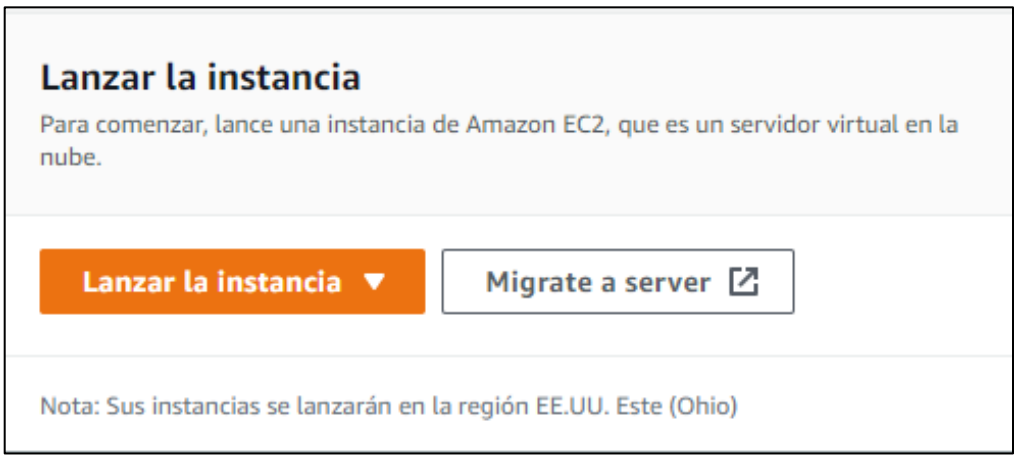

Figura B.4. Pantalla para iniciar la instancia EC2

ANEXO C

C. PROCEDIMIENTO PARA GENERAR LOS ARCHIVOS DE INSTALACIÓN DE MICROSERVICIOS Y APLICACIÓN MÓVIL

C.1. GENERACIÓN DEL ARCHIVO DE INSTALACIÓN DE LA APLICACIÓN MOVIL.

Para generar el archivo APK de instalación para dispositivos Android, es necesario dirigirse a la barra de herramientas de Android Studio y elegir la opción de "*Build*", como se muestra en la **Figura C.*1***.

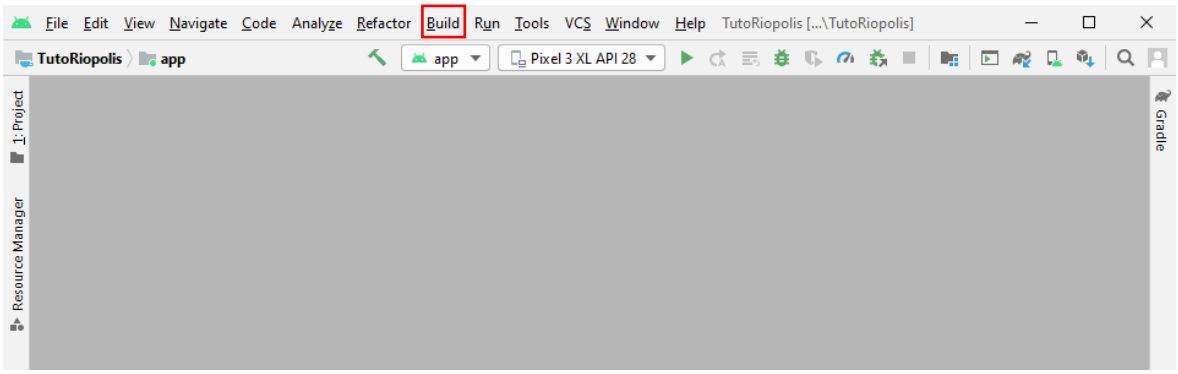

Figura C.1. Elegir opción "Build" en la barra de herramientas

Dentro de la opción "*Build*", se selecciona "*Build Bundles(s)*" y a su vez se elige la opción "*Build APK(S)*", como se muestra en la **Figura C.*2***.

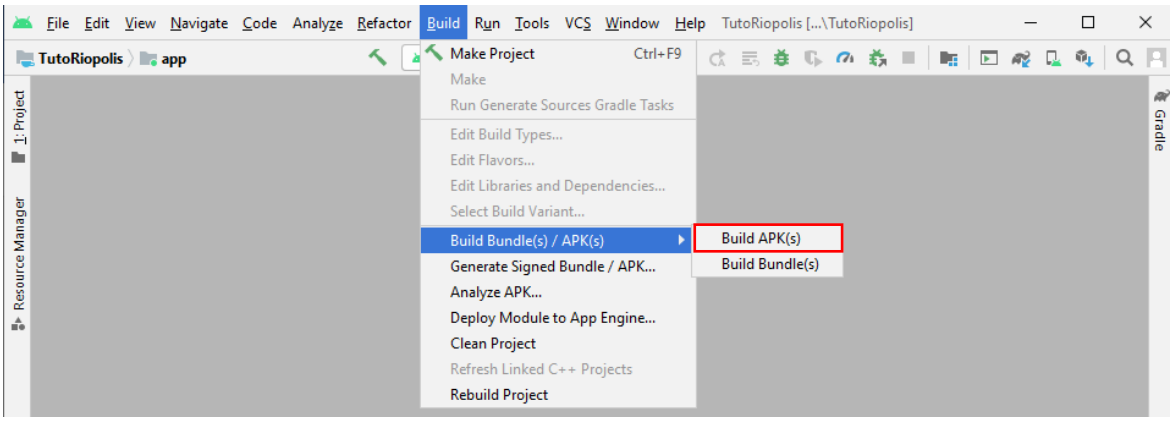

Figura C.2. Selección de "*Build APK(s)*" en la barra de herramientas "*Build*"

El APK generado, tendrá el nombre de "app-debug.apk" y se encontrará dentro del archivo del proyecto Android, en el directorio: app/build/outputs/apkdebug, como se muestra en la **Figura C.*3***.

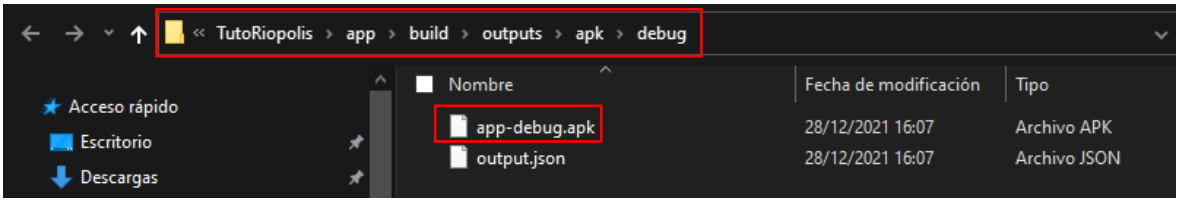

Figura C.3. Archivo APK generado

C.2. GENERACIÓN DEL ARCHIVO DE INSTALACIÓN DE LOS MICROSERVICIOS.

En el archivo pom.xml destinado para el manejo de dependencias, es necesario agregar la estructura de texto XML, que se muestra en el Código C.1. Este texto permite empaquetar archivos de tipo JAR ejecutables, ejecutar aplicaciones desarrolladas en el Framework de Spring Boot e iniciar una aplicación Spring.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

Código C.1. Estructura de texto XML para ejecutar aplicaciones desarrolladas en Spring

En el menú "*Run as*", se debe seleccionar la opción "Maven buil…", como se muestra en la **Figura C.*4***.

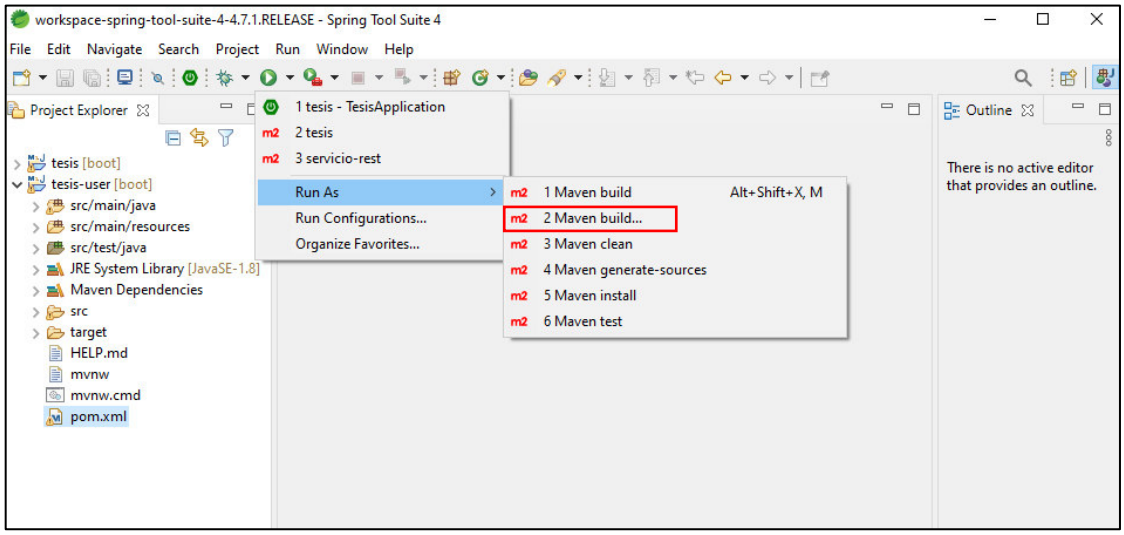

Figura C.4. Ingreso a la ventana para la configuración del ejecutable JAR del microservicio.

Al desplegarse la ventana "*Edit configuration and launch*", es necesario escribir la palabra "*package*" en el campo "*Goals*" y pulsar el botón "*Run*", como se muestra en la **Figura C.*5***.

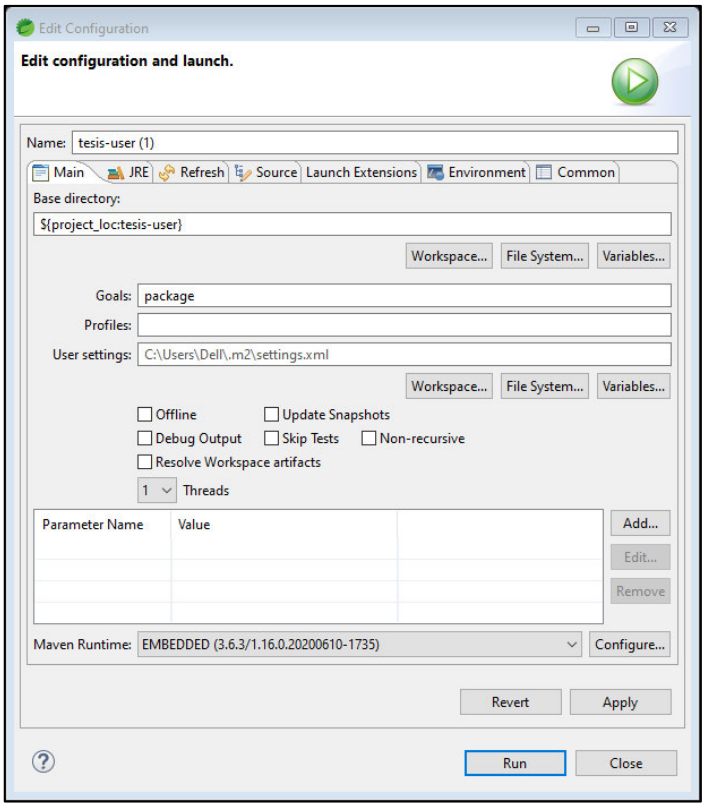

Figura C.5. Ventana de configuración para crear el ejecutable JAR del microservicio

En la ventana de la consola Spring Boot, es posible verificar si se ha creado el ejecutable del microservicio, como se muestra en la **Figura C.*6***.

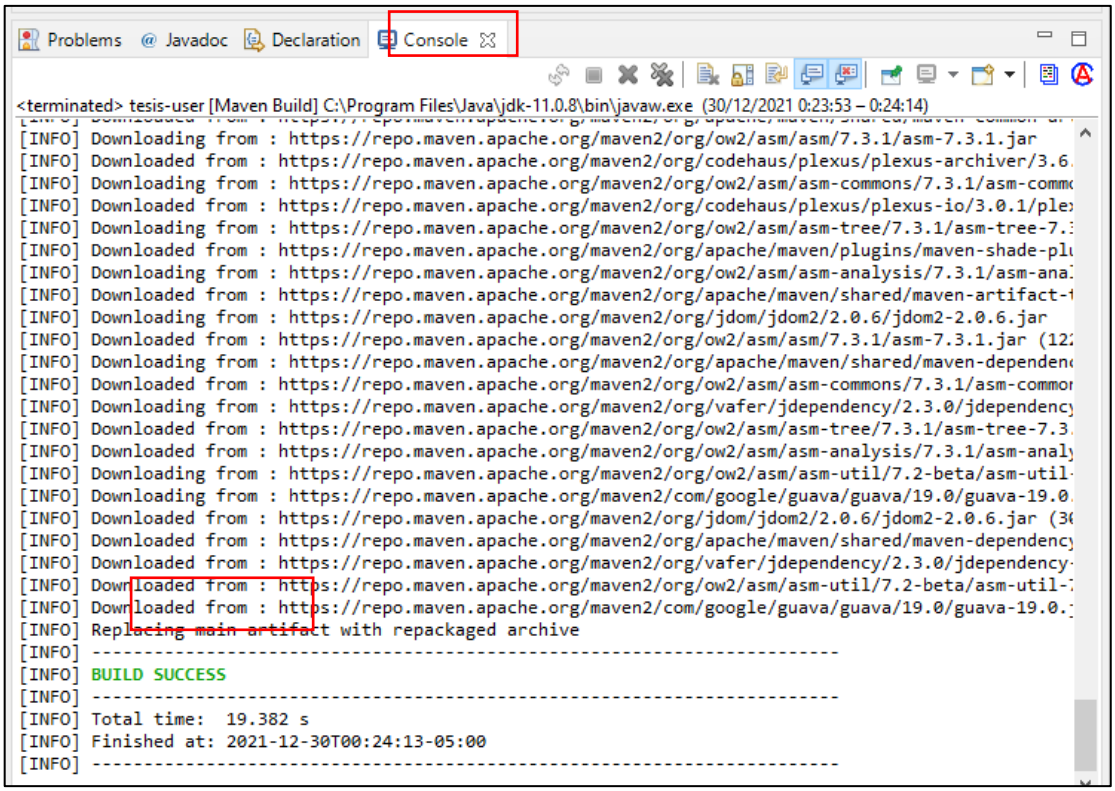

Figura C.6. Ventana de la consola Spring Boot, al crear el ejecutable JAR

Finalmente, se visualiza el archivo en el directorio en el cual se encuentra desarrollado el microservicio, como se muestra en la Figura C.7.

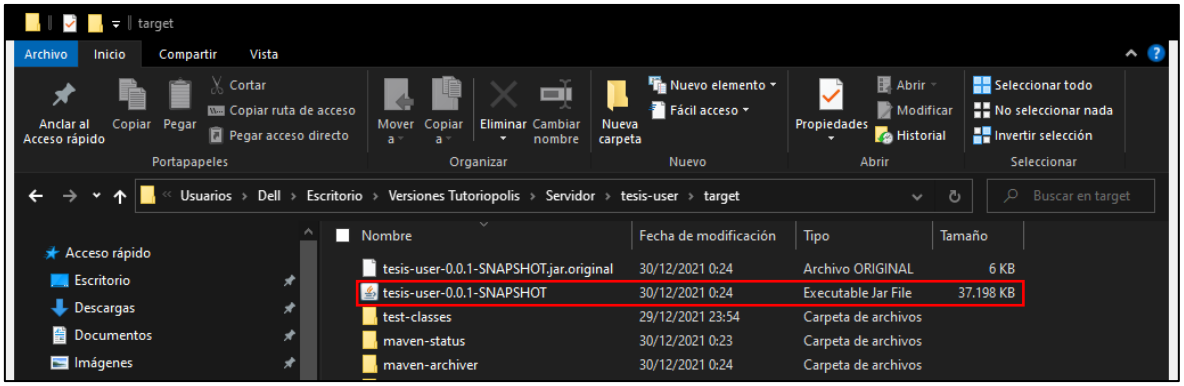

Figura C.7. Verificación de la creación del ejecutable