Universidad de Lima

Escuela de Ingeniería y Arquitectura

Ingeniería de Software II



**INFORME FINAL DEL PROYECTO: U-SELL**

**JUAN DIEGO CALDERON MOSQUEDA**

**30190171**

**BRUNO CÉSAR CÁRDENAS MARTÍNEZ**

**20161885**

**NATHAN ANDRÉS ESPINOZA YEP  
20160520**

**EDGARD ARON DECENO ROJAS**

**20122671**

**LEONARDO LUNA PALOMINO**

**20163263**

**Profesor: Hernán Quintana**

**Sección: 801**

Lima – Perú

Noviembre del 2019

Tabla de contenido

[Introducción](#_heading=h.30j0zll) **4**

[Marco Teórico](#_heading=h.1fob9te) **5**

[Ingeniería de Software](#_heading=h.8nci9atljxhv) 5

[Conceptos generales](#_heading=h.bb6v0psrrlev) 5

[Front-End](#_heading=h.4qtk2ddn9f2h) 5

[Back-End](#_heading=h.7zmqj9noqr8k) 5

[Model-vista-controlador(MVC)](#_heading=h.a7wd4mib4lpu) 6

[Metodologías Ágiles](#_heading=h.x3p755aujy26) 6

[SCRUM](#_heading=h.f8hh97eh08bt) 7

[Sprint](#_heading=h.5f5jlf4f3kg8) 7

[Planificación del Sprint](#_heading=h.5ee79z734wkk) 7

[Revisión de Sprint](#_heading=h.q8t4fz44vcqz) 8

[Retrospectiva de Sprint](#_heading=h.qz4tc4ya0iv) 8

[Conceptos tecnológicos](#_heading=h.p4dl8v7xwfy2) 9

[Android](#_heading=h.qzptey6jap6l) 9

[Java](#_heading=h.jqvkzkwghip0) 9

[XML](#_heading=h.ybzo35d3r17h) 9

[IOS](#_heading=h.ecvjegi95e7f) 9

[Swift](#_heading=h.6indtw62mz5h) 9

[Python](#_heading=h.xx0st0i4qmeh) 10

[Herramientas](#_heading=h.igub6tnz9nsk) 10

[Entorno de desarrollo integrado (IDE)](#_heading=h.tt4z5di0gx1t) 10

[Android Studio](#_heading=h.68l673twmqc4) 10

[Xcode](#_heading=h.146ifsrjbc9) 10

[Base de datos](#_heading=h.w9ex0dx3q07) 10

[Firebase](#_heading=h.wa04r1walb81) 10

[Diagrama de Clases](#_heading=h.lasfmi7ixz2t) 11

[Diagrama de Casos de Uso](#_heading=h.9yj348qekcfu) 11

[Diagramas UML](#_heading=h.j9v33xljr5vo) 12

[Krutchen Modelo 4 + 1](#_heading=h.ig6oblbwuaq) 12

[Vista Lógica](#_heading=h.f1yt7359xod6) 13

[Vista de procesos](#_heading=h.ag9m4ftb21kt) 13

[Diagrama de actividades](#_heading=h.5v24jjxt36g) 13

[Vista de Desarrollo](#_heading=h.pdcbnf8vlup8) 14

[Diagrama de componentes](#_heading=h.4gnpeq6zanvy) 14

[Diagramas de secuencia](#_heading=h.xuxnzv9khr0) 16

[Vista de despliegue](#_heading=h.cipwcxr54rti) 16

[Diagrama de despliegue](#_heading=h.qluodzdb00xt) 16

[Pruebas funcionales](#_heading=h.28i4ygv67eln) 17

[Organización del trabajo](#_heading=h.2et92p0) **18**

[Roles y Responsabilidades](#_heading=h.stw1uoyuuvyi) 18

[Desarrollo del Trabajo](#_heading=h.tyjcwt) **19**

[Back-end](#_heading=h.jq4l6p8mhjrz) 20

[IOS](#_heading=h.xoe5uu2wu4dg) 20

[Android](#_heading=h.l8fg5aqboamd) 22

[Validación y verificación del trabajo](#_heading=h.3dy6vkm) **22**

[**Conclusiones**](#_heading=h.4humxu13g0kw) **27**

[**Referencias**](#_heading=h.r1803gk2up9q) **29**

# 

# Introducción

En la actualidad, hemos sido conscientes de diversos cambios sociales los cuales van muy de la mano con la tecnología, así mismo, la internet es una de las más destacadas en los últimos tiempos. Gracias a esta, podemos interconectar personas, haciendo que las barreras sean cada vez menores. Hoy en día ya no es necesario tener una computadora para poder acceder a la internet, pues la mayor cantidad de usuarios accede a esta mediante los smartphones. Según INEI, en el 2018 el 79.6% de los usuarios de internet acceden a esta mediante su celular. Esto permite que diversas empresas y negocios tengan un gran auge en sus ganancias.

Pero la internet no es la única herramienta que facilita la vida de las personas en estos años. Encontramos que existen diversas aplicaciones móviles que, junto con la internet, nos dotan de herramientas para poder realizar tareas de una manera más simple. Podemos comunicarnos con amigos en otros países, escuchar millones de canciones, guardar archivos en la nube y comprar artículos, todo desde nuestro celular. Además, existen diversos aplicativos de compra y venta de artículos, tales como MercadoLibre, OLX, eBay. En estos, los usuarios pueden publicar diversos artículos que ya no usan y poder venderlos.

El presente trabajo consta en un aplicativo móvil, el cual estará disponible tanto en Android como en iOS. Este consta de una plataforma que facilita a los alumnos de la Universidad de Lima la compra y venta de productos variados relacionados con el ámbito académico, es decir, material universitario. La aplicación permite a los alumnos la creación de cuentas, donde podrán registrarse usando su correo universitario, además de ingresar la carrera la cual estudian y el nivel en el que se encuentre. Así mismo, los alumnos podrán buscar los productos que deseen, ya sea por la barra de búsqueda o por las categorías. Inclusive, dentro de la misma aplicación, podrán realizar publicaciones de artículos que deseen vender, donde podrán subir una foto del artículo.

La aplicación brindará los datos de los vendedores, de modo que los alumnos interesados en un determinado producto puedan contactarse con estos. Además, se tendrá un mecanismo de calificaciones, donde los propios usuarios podrán calificar tanto a compradores como a vendedores. Adicionalmente, los usuarios contarán con una lista de deseos, donde podrán almacenar aquellas publicaciones de su interés y que podrían adquirir en un futuro. Finalmente, el producto contará con un sistema de recomendaciones, el cual facilitará a los usuarios con publicaciones sugeridas de acuerdo a sus preferencias tomadas de sus listas de deseos, su carrera y el nivel en el que cursan actualmente.

Para realizar la aplicación en Android se utilizó Android Studio, haciendo uso del lenguaje Java y para la aplicación en iOS se desarrollado usando Xcode, empleando el lenguaje Swift. Además, utilizamos la base de datos libre PostgreSQL para administrar todos los datos que se manejan en el aplicativo. El Back-End se programó en Python, usando el framework Flask. Para almacenar las fotos de los perfiles y las publicaciones utilizamos Firebase Storage, para la gestión de las cuentas de los usuarios empleamos Firebase Authentication y el hosting del Back-End fue tercerizado a Heroku.

# Marco Teórico

Se desarrolló una aplicación que pueda ser utilizada para la compra y venta de artículos, donde los mismos usuarios pueden publicar y adquirir artículos deseados. Pero, como valor agregado, nos enfocamos en ámbito universitario. Es decir, proponemos el desarrollo de un aplicativo de compra y venta de artículos universitarios, donde los estudiantes puedan comercializar diverso material, como libros, útiles, uniformes e, inclusive, comida. Es necesario tener en cuenta que, en todo desarrollo de sistema de software es de suma importancia definir una metodología. Esta permite a los desarrolladores seguir alguna especificación en cada una de las etapas del desarrollo del sistema, en este caso una aplicación móvil, desde los requerimientos iniciales hasta las pruebas finales, que haga que la aplicación sea coherente y además formal.

#### **Ingeniería de Software**

La ingeniería de software es la disciplina de desarrollar y mantener sistemas de software que se comporten de manera confiable y eficiente, sean asequibles de desarrollar y mantener, y satisfagan todos los requisitos que los clientes han definido para ellos. Recientemente, ha evolucionado en respuesta a factores como el impacto creciente de los sistemas de software grandes y costosos en una amplia gama de situaciones, y la creciente importancia del software en aplicaciones críticas para la seguridad. La ingeniería de software es de carácter diferente de otras disciplinas de ingeniería debido a la naturaleza intangible del software y la naturaleza discontinua de la operación del software. Busca integrar los principios de las matemáticas y la informática con las prácticas de ingeniería desarrolladas para artefactos físicos tangibles.

Según la definición de Fritz Bauer de ingeniería de software en 1960: “Ingeniería de Software es el establecimiento y uso de principios robustos de ingeniería, orientados a obtener software que sea fiable y funcione de manera eficiente sobre máquinas reales”.

Algunos objetivos primordiales de la ingeniería de software:

* Lograr programas de alta calidad de un comportamiento predecible
* Lograr programas que sean fácilmente modificables (y mantenibles)
* Simplificar los programas y el proceso de desarrollo de programas
* Lograr mejores predicciones y controles en el proceso de desarrollo
* Acelerar el desarrollo de sistemas - Aminorar los costos del desarrollo de sistemas. (*Pontificia Universidad Javeriana, Cali Ingeniería de Software Profesora: Olga Lucía Roa*, n.d.)

#### **Conceptos generales**

##### **Front-End**

Es la parte del desarrollo web que se dedica de la parte frontal de un sitio web, en pocas palabras del diseño de un sitio web, desde la estructura del sitio hasta los estilos como colores, fondos, tamaños hasta llegar a las animaciones y efectos.

##### **Back-End**

El back-end es el área que se dedica a la parte lógica de un sitio web, es el encargado de que todo funcione como debería, es la parte de atrás que, de alguna manera, no es visible para el usuario ya que no se trata de diseño, o elementos gráficos, se trata de programar las funciones que tendrá un sitio. El back-end es la programación dura y pura, desde la programación de las funciones del sitio hasta bases de datos e incluso más.

**Framework**

Los frameworks tienen como objetivo principal ofrecer una funcionalidad definida, auto contenida, siendo construidos usando patrones de diseño, y su característica principal es su

alta cohesión y bajo acoplamiento (Chaudron et al., 2004). Para acceder a esa funcionalidad, se construyen piezas, objetos, llamados objetos calientes, que vinculan las necesidades del sistema con la funcionalidad que este presta. Esta funcionalidad, está constituida por objetos llamados fríos, que sufren poco o ningún cambio en la vida del framework, permitiendo la portabilidad entre distintos sistemas.

##### **Modelo-Vista-Controlador (MVC)**

Se trata de un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la interacción del usuario (Reenskaug, 2003). Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

#### **Metodologías Ágiles**

Podemos definir las metodologías ágiles como un conjunto tareas y procedimientos dirigidos a la gestión de proyectos (Sharma, Sarkar, & Gupta, 2012). Son aquellos métodos de desarrollo en los cuales tanto las necesidades como las soluciones a estas evolucionan con el pasar del tiempo, a través del trabajo en equipo de grupos multidisciplinarios que se caracterizan por tener las siguientes cualidades:

* Desarrollo evolutivo y flexible.
* Autonomía de los equipos.
* Planificación.
* Comunicación.

Existen diferentes opciones ágiles entre las cuales podemos destacar las siguientes: Scrum, programación extrema (XP) y Kanban, siendo estas tres (03) las alternativas más utilizadas. Es importante mencionar, que todas las metodologías ágiles cumplen con el Manifiesto ágil, el cual se encuentra compuesto por doce (12) principios agrupado en cuatro (04) valores fundamentales:

* Individuos e interacciones sobre procesos y herramientas.
* Software funcionando sobre documentación extensiva.
* Colaboración con el cliente sobre negociación contractual.
* Respuesta ante el cambio sobre seguir un plan.

En nuestro caso, definiremos Scrum, ya que fue la metodología utilizada durante la realización del proyecto.

##### **SCRUM**

Esta metodología, es un marco de trabajo de procesos ágiles que trabaja con el ciclo de vida iterativo e incremental, donde se va liberando el producto por pares de forma periódica, aplicando las buenas prácticas de trabajo colaborativo (en equipo), facilitando el hallazgo de soluciones óptimas a los problemas que pueden ir surgiendo en el proceso de desarrollo del proyecto (Sharma et al., 2012).

Con Scrum se realizan entregas regulares y parciales (sprint) del producto final, todas ellas con una prioridad previamente establecida que nace según el beneficio que aportan al cliente, minimizando los riesgos que pueden surgir de desarrollos extremadamente largos. Es por tal motivo, que Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesitan obtener resultados de manera inmediata y donde son fundamentales los siguientes aspectos: la innovación, la productividad, la flexibilidad y la competitividad.

Los eventos prescritos se usan en Scrum para crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Todos los eventos tienen un calendario. Una vez que comienza un Sprint, su duración es fija y no se puede acortar o alargar. Los eventos restantes pueden finalizar siempre que se logre el propósito del evento, asegurando que se pase una cantidad de tiempo adecuada sin permitir desperdicio en el proceso.

###### **Sprint**

un Sprint, un período de tiempo de un mes o menos durante el cual se crea un Incremento de producto "Listo", utilizable y potencialmente liberable. Los sprints tienen duraciones constantes a lo largo de un esfuerzo de desarrollo. Un nuevo Sprint comienza inmediatamente después de la conclusión del Sprint anterior. Durante el Sprint:

* No se realizan cambios que pongan en peligro el Objetivo Sprint
* Los objetivos de calidad no disminuyen
* El alcance puede aclararse y renegociarse entre el propietario del producto y el equipo de desarrollo a medida que se aprende más.

Los principales elementos de Scrum son Product Backlog, lista priorizada de funcionalidades técnicas y de negocio; Sprint Backlog identifica y define el trabajo a ser alcanzado por el equipo de desarrollo durante un Sprint; Incremento, parte de un sistema desarrollado en un Sprint. Este producto desarrollado es potencialmente entregable al final de cada Sprint, implica que todo está́ completamente terminado en cada Sprint. Estas iteraciones (llamadas sprints) se repiten de forma continua hasta que el cliente da por cerrado el producto.

###### **Planificación del Sprint**

La planificación de Sprint tiene un límite de tiempo de hasta ocho horas para un Sprint de un mes. Para Sprints más cortos, el evento suele ser más corto. El Scrum Master asegura que el evento se lleve a cabo y que los asistentes comprendan su propósito. El Scrum Master le enseña al Equipo Scrum a mantenerlo dentro del tiempo.

Sprint Planning responde a lo siguiente:

¿Qué se puede entregar en el Incremento resultante del próximo Sprint?

¿Cómo se logrará el trabajo necesario para lograr el Incremento?

El trabajo se selecciona de la Pila de productos y se coloca en la Pila de Sprint. Ahora recuerde que el trabajo en Sprint Backlog no es un compromiso, es un pronóstico. El único contenedor de un Sprint es su caja de tiempo, no el trabajo planeado para el Sprint. Obtén más información sobre el Backlog de Sprint y el Backlog del Producto.

###### **Revisión de Sprint**

Se realiza una Revisión de Sprint al final del Sprint para inspeccionar el Incremento y adaptar la Cartera de Producto si es necesario. Podría haber una sola implementación o muchas implementaciones durante un Sprint que conducen a ese Incremento para ser inspeccionado.

Durante la Revisión de Sprint, el Equipo Scrum y las partes interesadas colaboran sobre lo que se hizo en Sprint. En base a eso y a cualquier cambio en el Backlog del Producto durante el Sprint, los asistentes colaboran en las siguientes cosas que podrían hacerse para optimizar el valor. Esta es una reunión informal, no una reunión de estado, y la presentación del Incremento tiene como objetivo obtener comentarios y fomentar la colaboración.

Esto es como máximo una reunión de cuatro horas para Sprints de un mes. Para Sprints más cortos, el evento suele ser más corto. El Scrum Master asegura que el evento se lleve a cabo y que los asistentes comprendan su propósito. El Scrum Master enseña a todos los involucrados a mantenerlo dentro del tiempo.

###### **Retrospectiva de Sprint**

La Retrospectiva de Sprint es una oportunidad para que el Equipo Scrum se inspeccione a sí mismo y cree un plan de mejoras que se implementarán durante el próximo Sprint.

La Retrospectiva de Sprint ocurre después de la Revisión de Sprint y antes de la próxima Planificación de Sprint. Esto es como máximo una reunión de tres horas para Sprints de un mes. Para Sprints más cortos, el evento suele ser más corto. El Scrum Master asegura que el evento se lleve a cabo y que los asistentes comprendan su propósito. Esta es la oportunidad para que el Equipo Scrum mejore y todos los miembros deben estar presentes.

Durante la Retrospectiva de Sprint, el equipo discute:

* Lo que salió bien en el Sprint
* ¿Qué podría mejorarse?
* ¿Qué nos comprometeremos a mejorar en el próximo Sprint?

#### **Conceptos tecnológicos**

##### **Android**

Es un sistema operativo basado en Linux, diseñado en especial para dispositivos móviles con pantalla táctil como teléfonos y tablets. Fue desarrollado por Android, pero la compañía fue comprada por Google en 2005 (Dimarzio, 2017).

Actualmente es uno de los sistemas operativos móviles más extendidos del mercado. Sus aplicaciones se desarrollan en el lenguaje de programación Java, mediante un kit de desarrollo denominado Android SDK (Android Software Development Kit).

##### **Java**

Java es un lenguaje de programación, diseñado para ser concurrente, basado en clases y orientado a objetos, así como una plataforma informática lanzada por primera vez por Sun Microsystems en 1995 (Gosling, Joy, Steele, y Bracha, 2005). Una enorme cantidad de aplicaciones y sitios web no funcionarán a menos que tenga Java instalado, y se crean más cada día.

##### **XML**

El XML proviene de un lenguaje que inventó IBM allá por los años 70. El lenguaje de IBM se llama GML (General Markup Language) y surgió por la necesidad que tenían en la empresa de almacenar grandes cantidades de información de temas diversos. XML es un Lenguaje de Etiquetado Extensible muy simple, pero estricto que juega un papel fundamental en el intercambio de una gran variedad de datos. Es un lenguaje muy similar a HTML, pero su función principal es describir datos y no mostrarlos como es el caso de HTML. XML es un formato que permite la lectura de datos a través de diferentes aplicaciones, sirve entre otras cosas para estructurar, almacenar e intercambiar información.

##### **IOS**

iOS es el sistema operativo móvil de Apple utilizado para ejecutar los populares dispositivos iPhone, iPad y iPod Touch. Anteriormente conocido como iPhone OS, el nombre fue cambiado con la introducción del iPad. iOS utiliza una interfaz multitáctil donde se utilizan gestos simples para operar el dispositivo, como deslizar el dedo por la pantalla para pasar a la página siguiente o pellizcar los dedos para alejar. Hay más de un millón de aplicaciones iOS disponibles para descargar en la tienda de aplicaciones de Apple, la tienda de aplicaciones más popular de cualquier dispositivo móvil.

##### **Swift**

Swift es una de propósito general, multiparadigma, compilado lenguaje de programación desarrollado por Apple Inc. para iOS, iPadOS, macOS, watchos, TVOS, Linux, y z / OS. Swift está diseñado para funcionar con los marcos Cocoa y Cocoa Touch de Apple y el gran cuerpo de código Objective-C existente escrito para productos Apple. Está construido con el marco del compilador LLVM de código abierto y se ha incluido en Xcode desde la versión 6, lanzada en 2014. En las plataformas Apple, utiliza la biblioteca de tiempo de ejecución Objective-C que permite que los códigos C, Objective-C, C ++ y Swift se ejecuten dentro de un programa.

##### **Python**

Python es un lenguaje de programación interpretado de tipado dinámico cuya filosofía hace hincapié en una sintaxis que favorezca un código legible (Guttag, 2016). Se trata de un lenguaje de programación multiparadigma y disponible en varias plataformas.

#### **Herramientas**

##### **Entorno de desarrollo integrado (IDE)**

Un IDE es un entorno de programación empaquetado como un programa de aplicación. Consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. En nuestro caso, hemos utilizado dos entornos que explicamos a continuación.

###### **Android Studio**

Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

###### **Xcode**

Introducido en octubre de 2003 junto a la versión 10.3 de Mac OS X, Xcode es el entorno de desarrollo integrado (IDE) creado por Appple que trabaja de manera conjunta con Interface Builder (herramienta gráfica para crear interfaces de usuario) y que se puede descargar de manera gratuita desde la Mac App Store. Este conjunto de herramientas además incluye una colección de compiladores del proyecto GNU (GCC) y puede compilar código C, C++, Objective-C, Objective-C++, AppleScript y Java gracias a una gran gama de modelos de programación.

##### **Base de datos**

###### **Firebase**

Firebase proporciona una base de datos en tiempo real, back-end y organizada en forma de árbol JSON. El servicio proporciona a los desarrolladores de aplicaciones una API que permite que la información de las aplicaciones sea sincronizada y almacenada en la nube de Firebase. (Rodrigo, Tutor, Ramón, & Nieto Rodrigo, n.d.)

1. Firebase Auth: Firebase Auth es un servicio que puede autenticar los usuarios utilizando únicamente código del lado del cliente. Incluye la autenticación mediante proveedores de inicio de sesión como Facebook, GitHub, Twitter, Google, Yahoo y Microsoft; así como los métodos clásicos de inicio de sesión mediante correo electrónico y contraseña. Además, incluye un sistema de administración del usuario por el cual los desarrolladores pueden habilitar la autenticación de usuarios con email y contraseña que se almacenarán en Firebase.
2. Firebase Storage: Firebase Storage proporciona cargas y descargas seguras de archivos para aplicaciones Firebase, sin importar la calidad de la red. Se puede utilizar para almacenar imágenes, audio, vídeo, o cualquier otro contenido generado por el usuario. Firebase Storage se basa en el almacenamiento de Google Cloud Storage.

#### **Diagrama de Clases**

Los diagramas de clases describen la estructura estática de un sistema. Las cosas que existen y que nos rodean se agrupan naturalmente en categorías. Una clase es una categoría o grupo de cosas que tienen atributos (propiedades) y acciones similares. Un ejemplo de ello es la clase “Carrera” que tiene atributos como el “id” y “nombre” del alumno. Entre las acciones de las cosas de esta clase se encuentran: “agregarCarrera” y “EliminarCarrera”.



#### **Diagrama de Casos de Uso**

Un caso de uso es una descripción de las acciones de un sistema desde el punto de vista del usuario. Es una herramienta valiosa dado que es una técnica de aciertos y errores para obtener los requerimientos del sistema, justamente desde el punto de vista del usuario. Los diagramas de caso de uso modelan la funcionalidad del sistema usando actores y casos de uso. Los casos de uso son servicios o funciones provistas por el sistema para sus usuarios.

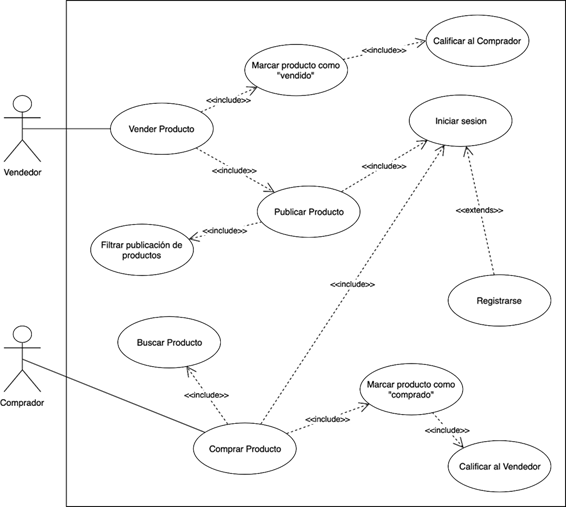
Se identificaron a dos actores dentro del sistema planteado en nuestra aplicación.

1. Comprador: Alumno universitario que desea usar la aplicación para encontrar útiles que vaya a usar durante clases a un precio accesible. Buscará entre los productos ofrecidos y seleccionará la oferta que crea más conveniente.
2. Vendedor: Alumno universitario que decide utilizar la plataforma como un medio por el cual ofertar un producto universitario que haya usado en algún curso durante su carrera pero que ya no necesite. Este usuario podrá publicar un anuncio en el que ofrezca el producto para que potenciales compradores lo puedan ver.

Existen dos casos de uso de negocio, que corresponden a los actores previamente mencionados y que cubren los dos principales motivos por los que usar la aplicación.

1. Comprar producto: Comprende el hecho de utilizar la aplicación para comprar un producto en específico. Para llegar a esto, el usuario puede buscar directamente el producto que tiene decidido comprar mediante la barra de búsqueda, o puede escanear los distintos anuncios de productos a través de las categorías y, finalmente, elegir la mejor oferta y contactar con el vendedor para llevar a cabo la compra.
2. Vender producto: Caso de uso que será llevado a cabo por quienes deseen asumir el rol de vendedores y ofertar algún producto a través de la aplicación. Lo harán publicando un anuncio en el cual den información sobre su producto, y podrán esperar a que algún interesado se disponga a comprar el producto para así iniciar el contacto con ellos.

Los casos de uso de la aplicación se corresponden con cada una de las funcionalidades que tiene la misma, así pues, el diagrama de casos de uso será el siguiente.



#### **Diagramas UML**

En la aplicación utilización el Lenguaje Unificado de Modelado (UML) cuando se preparan todas las propuestas para la aplicación. UML es un lenguaje estándar de modelado para visualizar, especificar, construir y documentar artefactos de un sistema de software (Larman, 1999). Ofrece una forma estándar para describir los planos de un sistema, incluyendo cosas conceptuales como procesos de negocio y funciones de sistemas tanto como cosas concretas como sentencias de lenguajes de programación, esquemas de las bases de datos y los componentes de software.

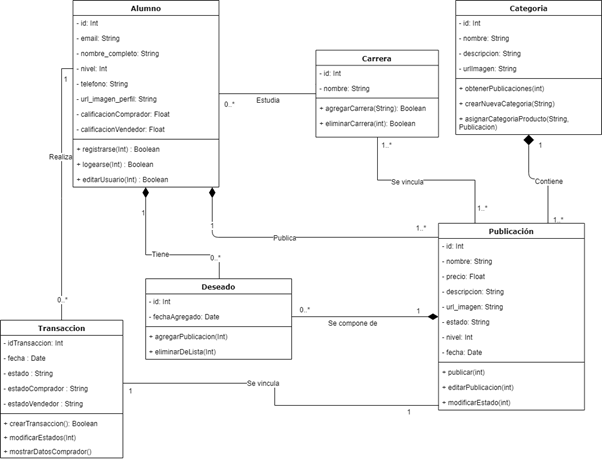
El lenguaje UML, como antes se mencionó, ofrece una forma estándar de describir planos de un sistema, los diagramas fruto de este proceso son detallados usando el modelo arquitectónico “4 + 1” de Kruchten que será explicado enseguida.

#### **Krutchen Modelo 4 + 1**

El modelo 4+1 es una forma de poder organizar toda la documentación de diseño de nuestro software según distintas vistas y en torno a casos de uso (Kruchten, 1995).

##### **Vista Lógica**

La vista lógica permite representar una expresión conceptual para resolver los requerimientos funcionales, plasmando su estructura en un diseño gráfico que expresa la funcionalidad global de la aplicación móvil. En ella se incluyen los diagramas de clase descritos anteriormente que permiten visualizar las necesidades del usuario en un entorno de trabajo, representando las relaciones, notaciones y reglas entre las diferentes entidades del sistema. Por ejemplo, se puede observar el diagrama de clases con todas sus clases y relaciones. Cada clase cuenta con atributos y métodos.



En Android todas las interfaces representan un **Activity**, estas actividades están formadas por dos partes: la parte lógica que es un archivo .java que contiene el código fuente para manipular dicha actividad y la parte gráfica que es un XML que contiene los elementos que vemos en la pantalla que son declarados mediante etiquetas.

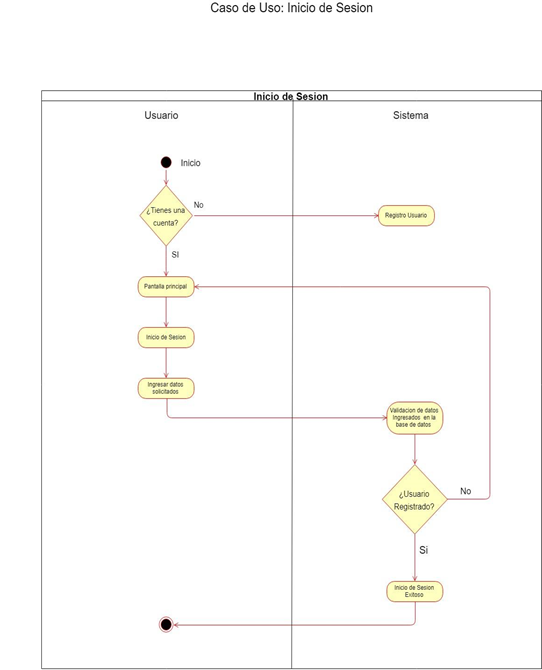
##### **Vista de procesos**

En esta vista se muestra la dinámica de la interacción de los objetos especificados en la Vista Lógica.

###### **Diagrama de actividades**

Un diagrama de actividades muestra el flujo de actividades, siendo una actividad una ejecución general entre los objetos que se está ejecutando en un momento dado dentro de una máquina de estados, el resultado de una actividad es una acción que producen un cambio en el estado del sistema o la devolución de un valor. Las acciones incluyen llamadas a otras operaciones, envío de señales, creación o destrucción de objetos o simples cálculos.

A continuación, se mostrará el ejemplo de Inicio de Sesión y los pasos que sigue la aplicación. En el diagrama de iniciar sesión, se representa el proceso que sigue la aplicación cuando un usuario quiere acceder a la misma.



Para poder iniciar sesión, deberás estar registrado, es por ello que la aplicación te pregunta si tienes una cuenta, en caso de que no, te mandara a la pantalla para que te registres, en caso de que si tengas cuenta deberás de Ingresar tu correo electrónico (Previamente validado en el registro) y tu contraseña. Una vez ingresado los datos, el sistema hará la validación de los datos previamente ingresados, si todo está correcto debería de aparecer la pantalla principal de la aplicación.

##### **Vista de Desarrollo**

Su objetivo es mostrar aspectos directos a la implementación (desarrollo) de la solución de software.

###### **Diagrama de componentes**

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes.



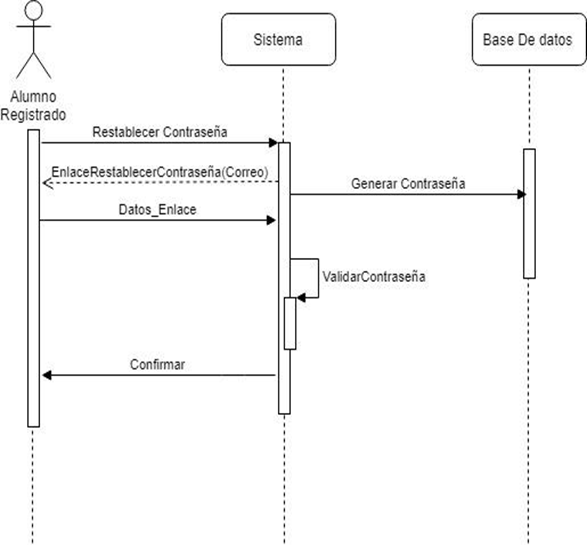
En la ilustración se pueden observar los componentes que conforman a la aplicación, se utilizó Firebase como base de datos, Firebase Auth para gestionar las cuentas, el logeo y el envío de correo de validación, Firebase Storage para guardar las fotos. Además, se utilizó Heroku para el despliegue de la aplicación, también en los componentes se incluyen las librerías utilizadas en Android e iOS, así como demás componentes para poder realizar el desarrollo tanto en IOS como en Android.

A continuación, se definen algunos de los componentes utilizados.

1. **Heroku:** Heroku permite desplegar en cuestión de minutos aplicaciones en diferentes lenguaje Open Source (Ruby, Node.js, Java, Python, Clojure, Scala, Go and PHP), permitiendo integrar desde un solo lugar más de 150 servicios (bases de datos, logs, monitoreo, sistemas de caching, almacenamiento de archivos, notificaciones, entre otros) a las aplicaciones, así como realizar la integración con diferentes sistemas externos (en el centros de datos de la empresa o sobre otras soluciones en la nube).}
2. **Layout: XML:** Parte gráfica que es un XML que contiene los elementos que vemos en la pantalla que son declarados mediante etiquetas.
3. **StoryBoard:** Son una característica emocionante introducida por primera vez en iOS 5, que ahorra tiempo al crear interfaces de usuario para sus aplicaciones. Los StoryBoard permiten crear prototipos y diseñar múltiples vistas de controlador de vista dentro de un archivo, y también le permiten crear transiciones entre controladores de vista.

###### **Diagramas de secuencia**

Los diagramas de secuencia representan la interacción de objetos representados en líneas de vida a través del tiempo por medio de mensajes, los cuales están compuestos por una serie de elementos como flechas que se dirigen desde la línea de vida origen hasta la línea de vida destino y representan el intercambio de mensajes. A continuación, se realizará la descripción del diagrama de secuencia correspondiente al caso de uso “Recuperar contraseña”.



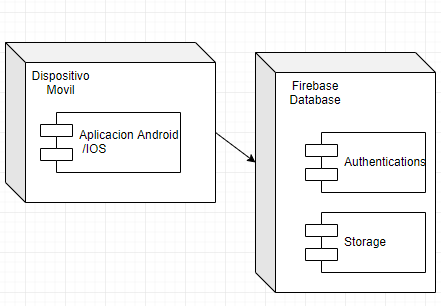
La ilustración hace referencia al diagrama de secuencia para la recuperación de la contraseña para poder entrar a la aplicación móvil que está compuesto por actores, líneas de vida, entidades y relaciones de interacción donde el usuario será el actor que interactúa con las líneas de vida aplicación móvil, base de datos y lógica del negocio por medio de mensajes enviados entre ellas. El actor usuario solicita “recuperar la contraseña”, hecho eso al ingresar tu código de alumno, te saldrá un aviso diciendo que “Le hemos enviado instrucciones a su correo electrónico, después de ello tendrás que entrar al link enviado por correo electrónico, generas tu nueva contraseña con los parámetros correspondientes a las credenciales de ingreso de la aplicación móvil donde se valida la nueva contraseña, para finalizar se confirma la nueva contraseña.

##### **Vista de despliegue**

Su objetivo es mostrar aspectos de la instalación y despliegue físico de la solución de software.

###### **Diagrama de despliegue**

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos del hardware (nodos o módulos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos/módulos.



En este se pueden observar dos módulos que serían elementos de software o hardware. Estos son conectados mediante asociaciones de comunicación, en este caso se observa el diagrama de despliegue del caso de uso “Registrar usuario”, teniendo en cuenta que interviene el software de base de datos “Firebase”, que será la base de datos con la que interactuará el dispositivo móvil. Además, se puede observar que se utilizará Firebase Authentication para la gestión de cuentas y el envío del correo de validación, y Firebase Storage para almacenar las fotografías.

#### **Pruebas funcionales**

Las pruebas funcionales tienen como objetivo validar si el comportamiento observado de la app probada cumple o no con las especificaciones del usuario (Offutt y Ammann, 2008). El plan trata en establecer la cronología, riesgos, condiciones, requisitos, casos de las pruebas, entorno que hay que probar para obtener un sistema con recepción total de los interesados y entrar en operación con la totalidad de las funcionalidades requeridas para estos.

Enseguida se explica el caso de uso Whishlist.

1. Whishlist
   1. Instrucciones:
      1. Dirigirse a la descripción de algún producto.
      2. Seleccionar el botón de estrella en la esquina superior derecha de la pantalla.
      3. Aceptar la alerta emergente.
   2. Criterios de aceptación:
      1. Dirigirse a la pestaña de “Perfil” y seleccionar el botón “Lista de deseado”. Ahí verificar que figure el producto previamente agregado.

# Organización del trabajo

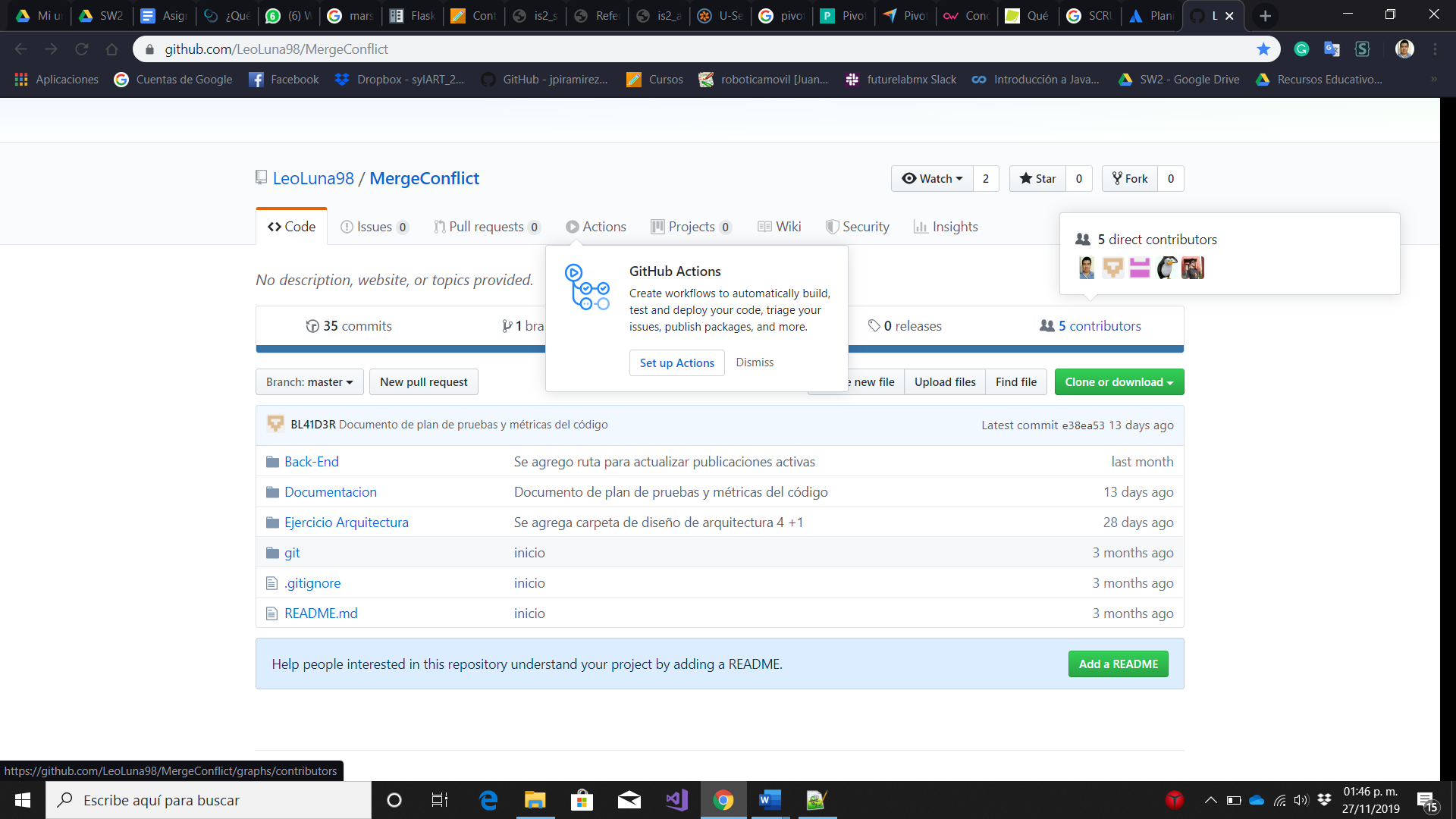
Se adoptó la metodología ágil de Scrum, todos los casos de usos se planificaron y se registraron como Sprint Backlog. Se realizó reuniones una vez por semana (Daily meetings) antes de la fecha de presentación.

Para la realización de este proyecto se contó con la participación de cinco integrantes los cuales se encuentran mencionados entre los integrantes de este documento. Cada uno es especialista en distintas áreas tales como programación del Back-End, desarrollo en Android, iOS, entre otros. Cada uno se desempeñó en distintos roles a lo largo del proyecto a fin de construir el producto final en base a sus capacidades.

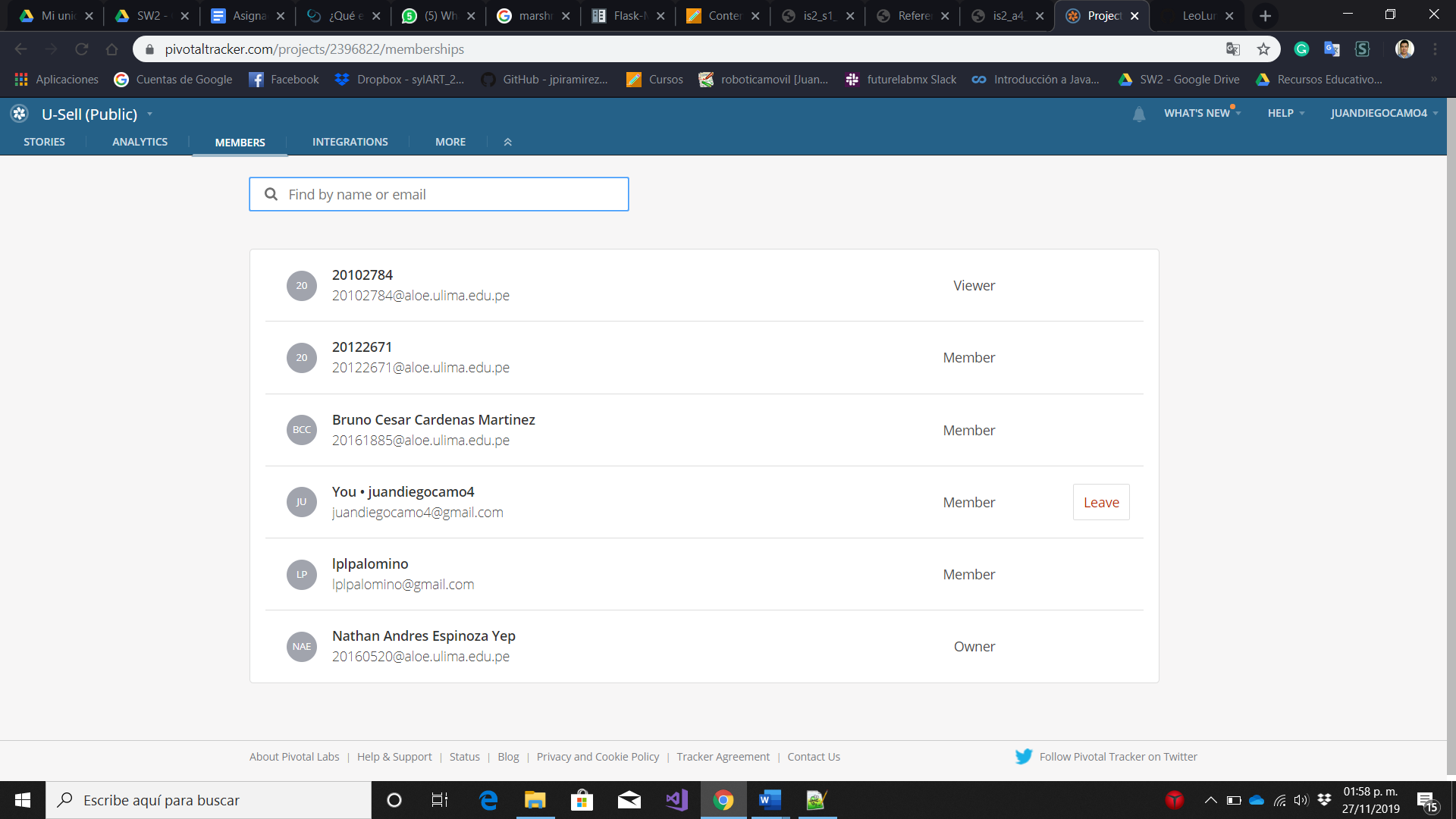
##### **Roles y Responsabilidades**

* Desarrollador de Android (Bruno Cárdenas): encargado del desarrollo de la aplicación para la plataforma Android.
* Desarrollador de IOS (Leonardo Luna): encargado del desarrollo de la aplicación para la plataforma iOS.
* Desarrollador del Back-End (Nathan Espinoza, Leonardo Luna y Juan Diego Calderón): encargado del desarrollo de la aplicación back-end para acceder a la base de datos.
* Calidad (Nathan Espinoza, Edgar Deceno y Juan Diego Calderón): encargados de evaluar el funcionamiento del proyecto. Detectar errores del programa. Asegurar que el producto cumpla con los requisitos planteados y haga lo que debe de hacer. Probar los programas en diversos escenarios que el usuario podría estar.

Para cumplir con esta metodología utilizamos la herramienta de Github para poder realizar las tareas, tanto para subir la documentación, como trabajar con el manejo de versiones del trabajo.



También, utilizamos Pivotal Tracker que es una herramienta para la gestión de proyectos ágiles. Asimismo, esta es un utensilio para planificación de proyectos basada en historias que se inspira en métodos de software ágiles. En pocas palabras, el software ágil se actualiza regularmente con ajustes incrementales en lugar de una gran actualización, con la idea de proporcionar un software más rápido y con mayor capacidad de respuesta.

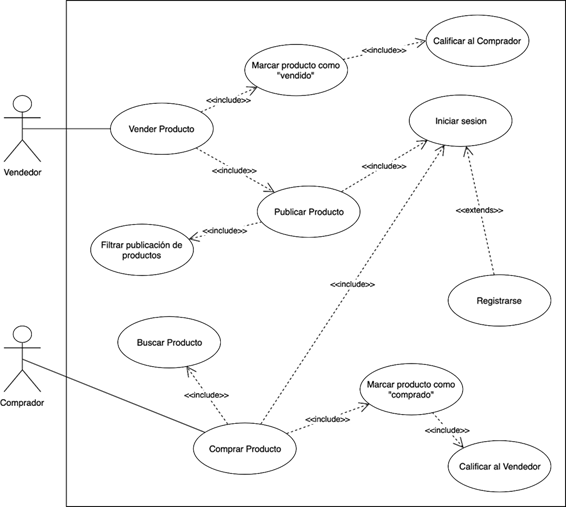


Además, utilizamos Google Docs para realizar la documentación, se optó por esta herramienta ya que podemos hacer una colaboración en tiempo real. Sin embargo, luego de concluir con la colaboración en esta herramienta, también subimos la versión del documento al repositorio de GitHub para mantener también una constancia de la versión del documento y la fecha.

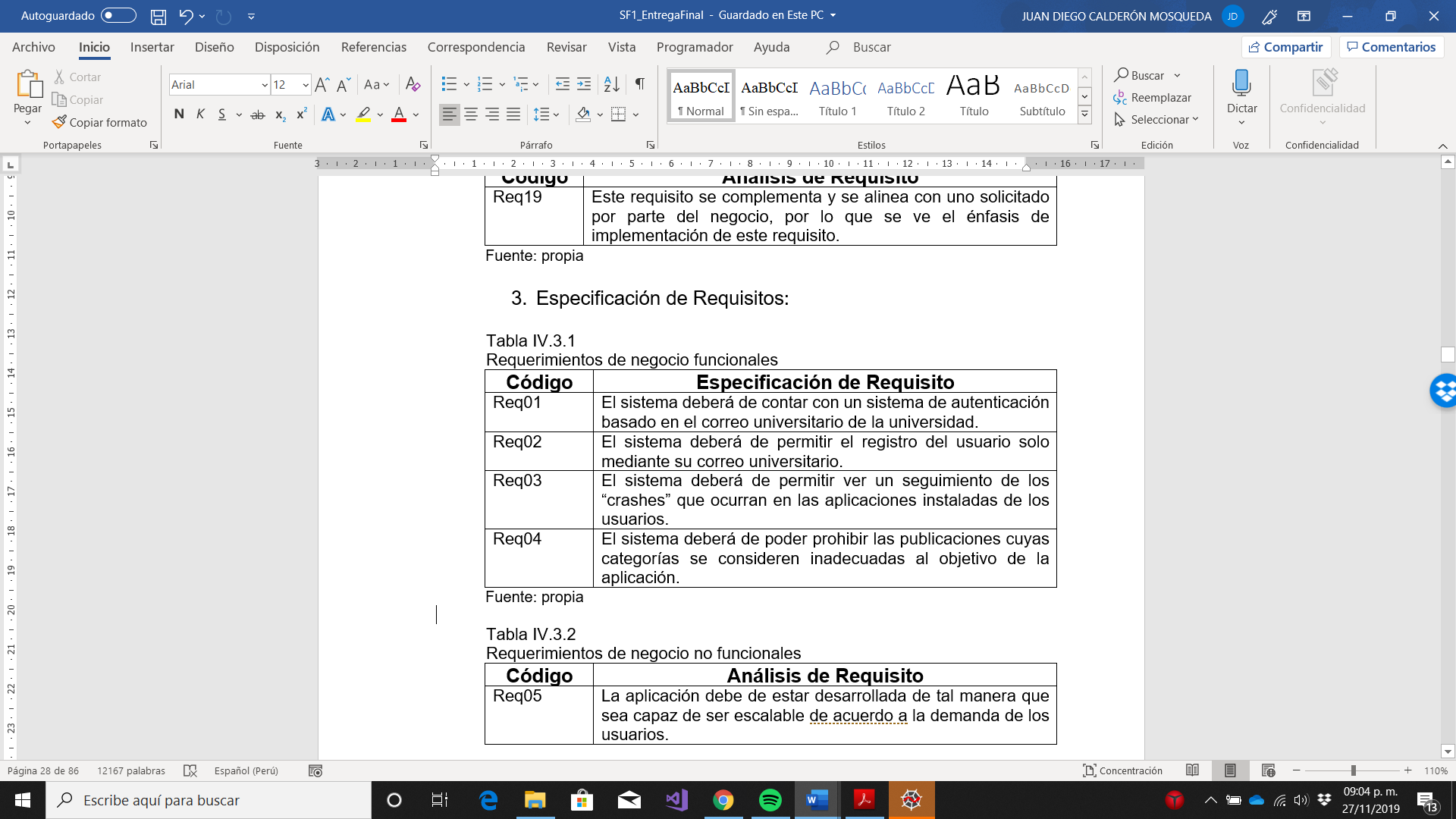
# Desarrollo del Trabajo

Para realizar la aplicación en Android se utilizó Android Studio, haciendo uso del lenguaje Java y para la aplicación en iOS se desarrollado usando Xcode, empleando el lenguaje Swift. Además, utilizamos la base de datos libre PostgreSQL para administrar todos los datos que se manejan en el aplicativo. El Back-End se programó en Python, usando el framework Flask. Para almacenar las fotos de los perfiles y las publicaciones, utilizamos Firebase Storage. Para la gestión de las cuentas de los usuarios, empleamos Firebase Authentication y el hosting del Back-End fue tercerizado a Heroku.

Antes de realizar la aplicación, se realizaron los diagramas UML para los casos de uso, para comprender la interacción entre el usuario y el sistema, así como la forma en que el sistema debe responder a las peticiones del usuario, presentando de manera visual el comportamiento esperado para iniciar la fase de programación.



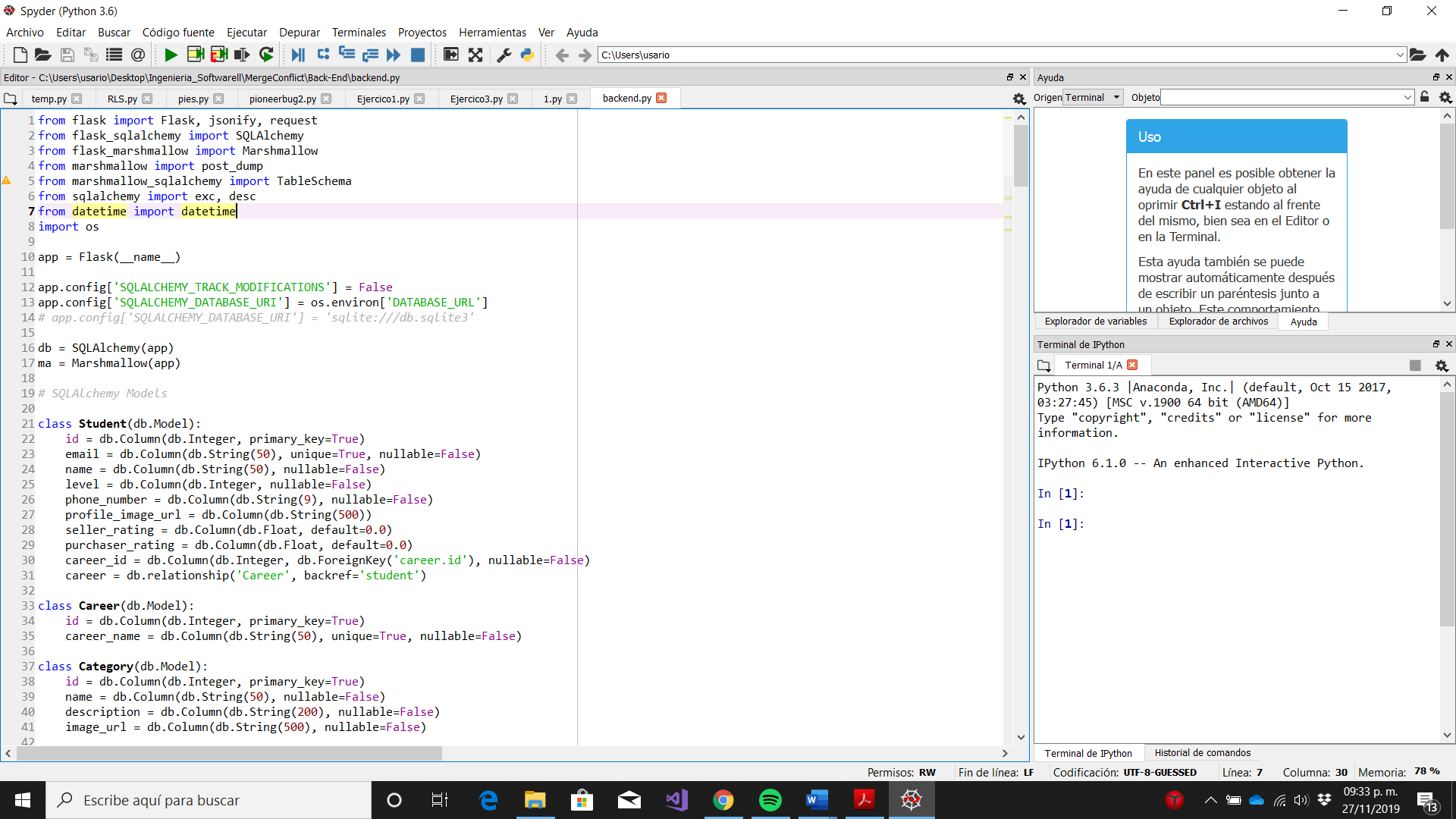
Así mismo, se definieron los requerimientos de negocio funcionales del sistema. Estos indican cómo debe comportarse y responder el mismo como se muestra en la ilustración siguiente.



En seguida se explican parte del código que fueran implementadas durante la realización de dicho proyecto.

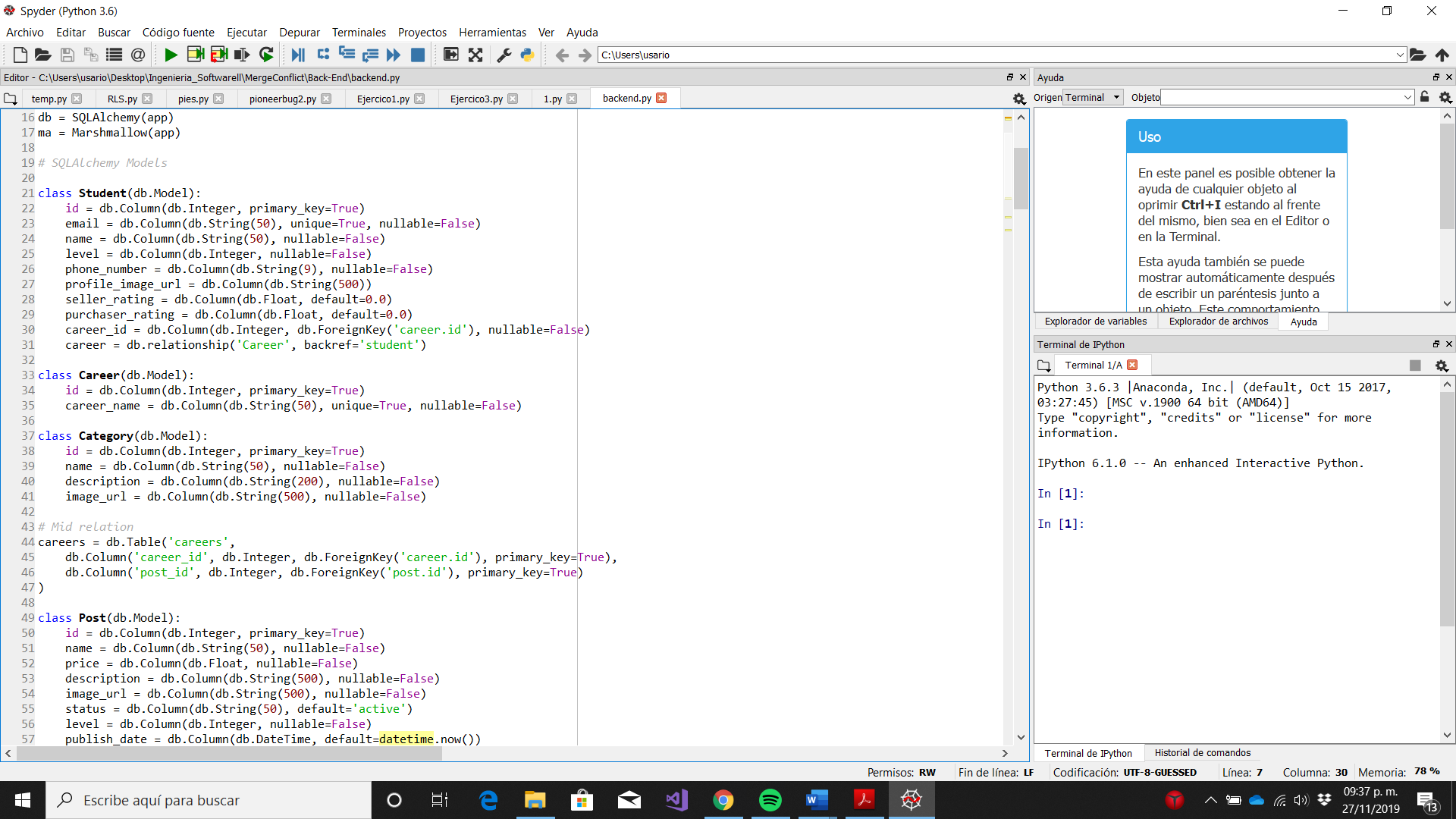
##### **Back-end**

Como se mencionó anteriormente, el Back-End se programó en Python, usando el framework Flask.



En la ilustración se puede observar a Flask-Marshmallow que es una capa de integración delgada para Flask (un marco web de Python) y Marshmallow (una biblioteca de serialización / deserialización de objetos) que agrega características adicionales a Marshmallow, incluidos los campos de URL e hipervínculos para las API listas para HATEOAS. También (opcionalmente) se integra con Flask-SQLAlchemy.

A partir de ello se declaran los modelos, mostrados en la siguiente ilustración.

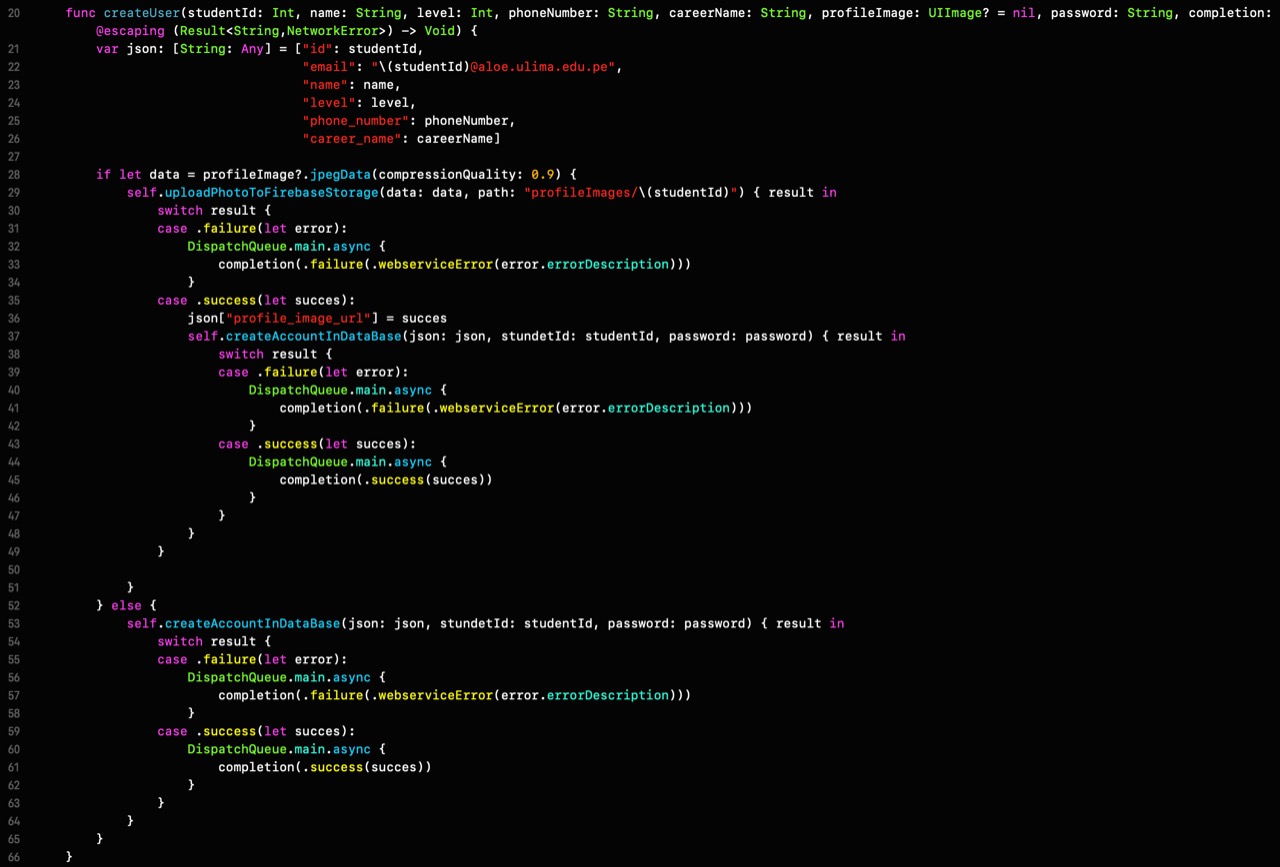


##### **IOS**

En la ilustración siguiente se muestra el conjunto de funciones utilizadas para manejar "Core Data", que es un framework de Apple que permite la persistencia de datos en dispositivos iOS. Estas funciones básicamente son utilizadas para crear entidades y luego almacenarlas, así como para realizar consultas y borrar datos.



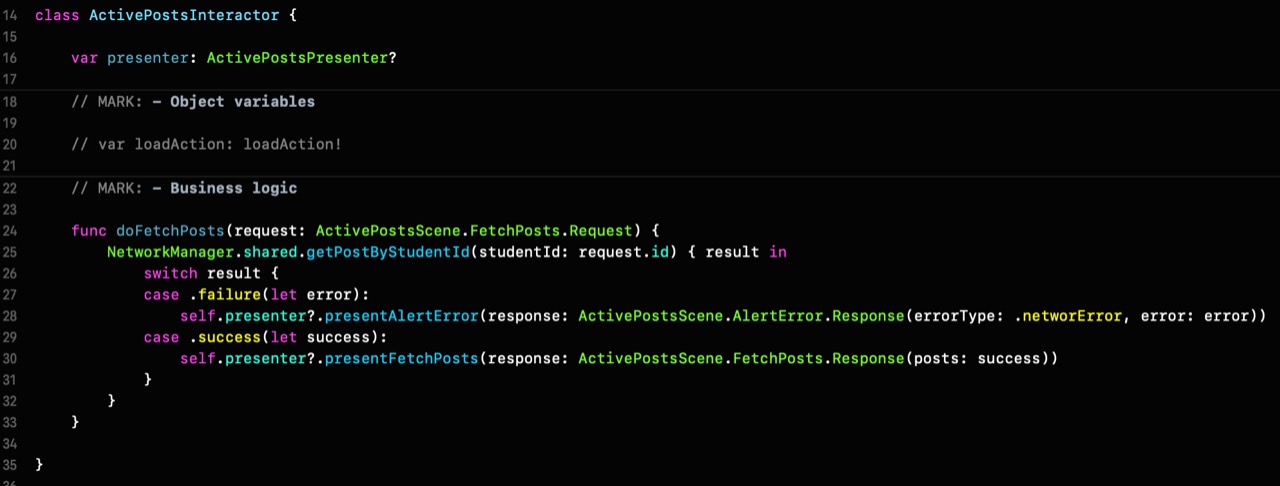
En la siguiente ilustración se muestran las funciones encargadas de crear un usuario en la base de datos mediante un servicio web. Aquí, así mismo, se crear el usuario en Firebase para poder utilizar la autenticación que este servicio nos proporciona, como mandar correo de verificación y validar, posteriormente, al usuario cuando quiera iniciar sesión.



En la siguiente ilustración se muestra una clase usada para asignar los colores a utilizar en la app. Se creó, principalmente, para mantener la consistencia de colores en toda la aplicación sin tener que estar indicando el código de color RGB cada vez que se requiera el color.



En la siguiente ilustración se muestra un ejemplo de uso de interactor, el cual es una clase que conforma el patrón de arquitectura usada para la aplicación iOS (VIPER) que se encarga de obtener los datos necesarios para la pantalla a mostrar.

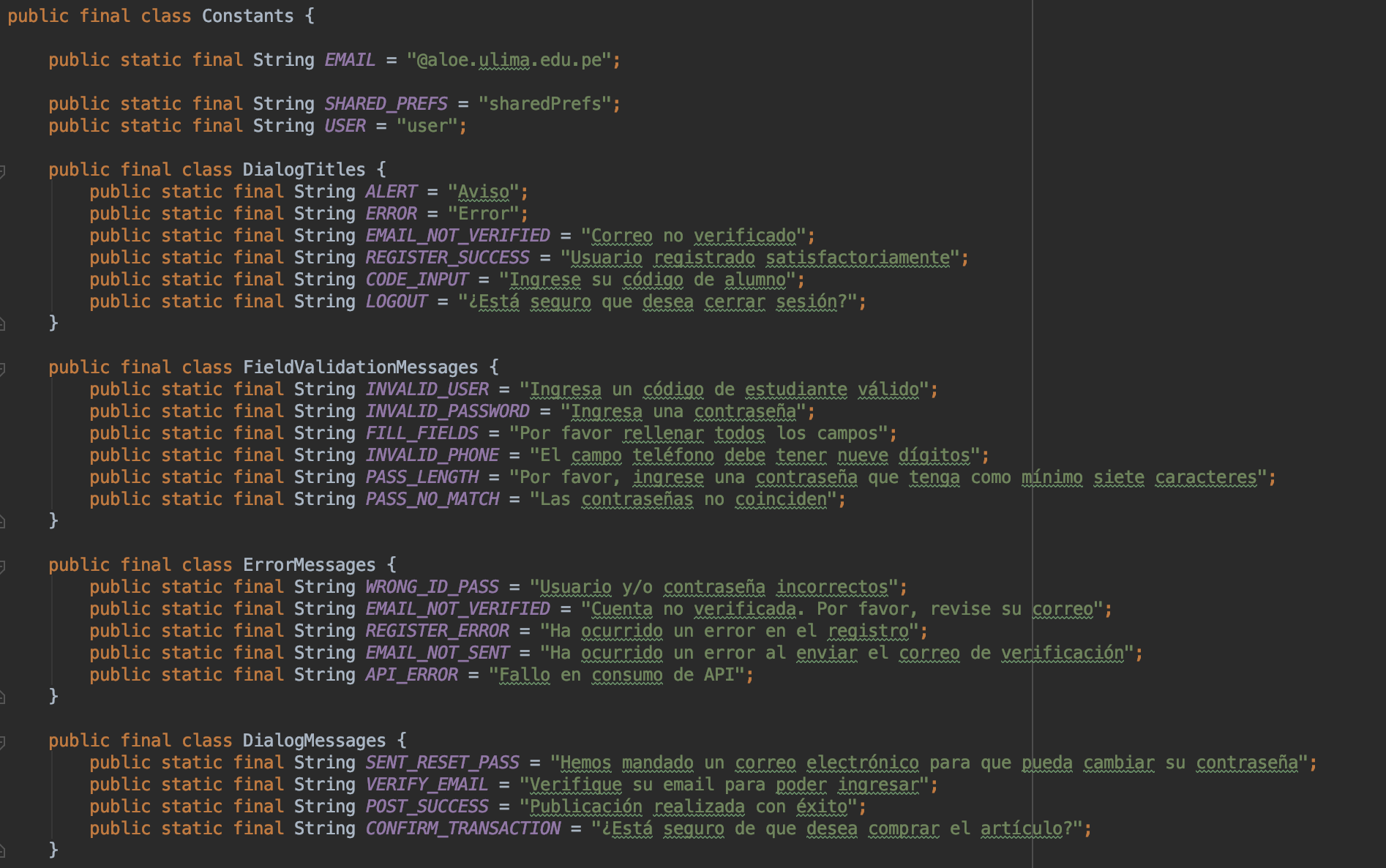


##### **Android**

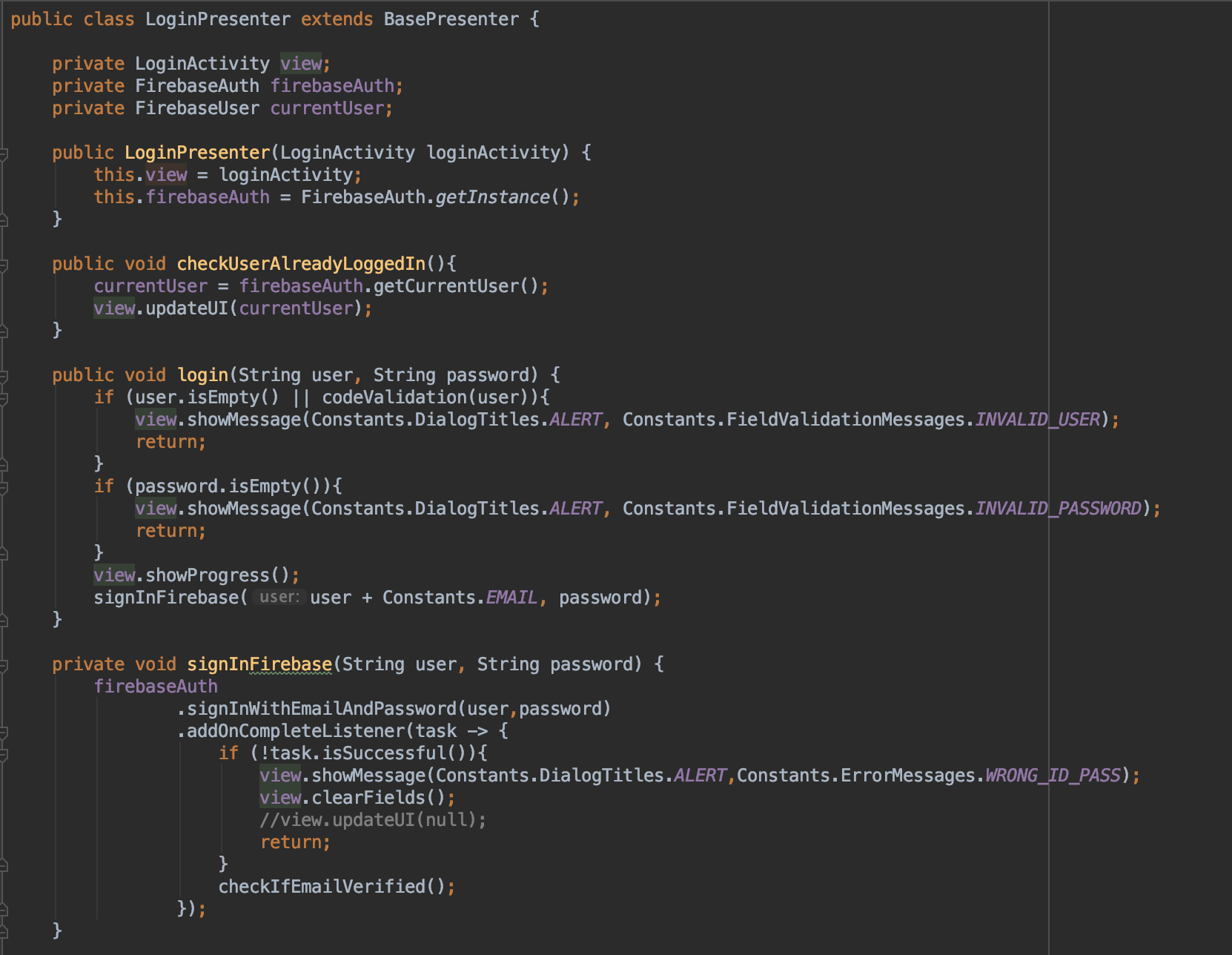
Por otro lado, la programación en Android utiliza el lenguaje de Java con el IDE de Android Studio. Al igual que iOS, los usuarios se crean tanto en las bases de datos de Firebase Authentication como en el backend en Python con PostgreSQL que desplegamos en Heroku. Para ello, se utilizó los siguientes dos métodos, en el cual el primero es el necesario para crear el usuario en nuestra base de datos en PostgreSQL, y este utiliza un método POST para enviar los atributos necesarios al web service. Asimismo, en este método también se llama al segundo, en donde se crear especificamente el usuario en la base de datos de Firebase, pero sólo utilizamos esta para manejar la autenticación de los usuarios y para el storage de las fotos de las publicaciones.



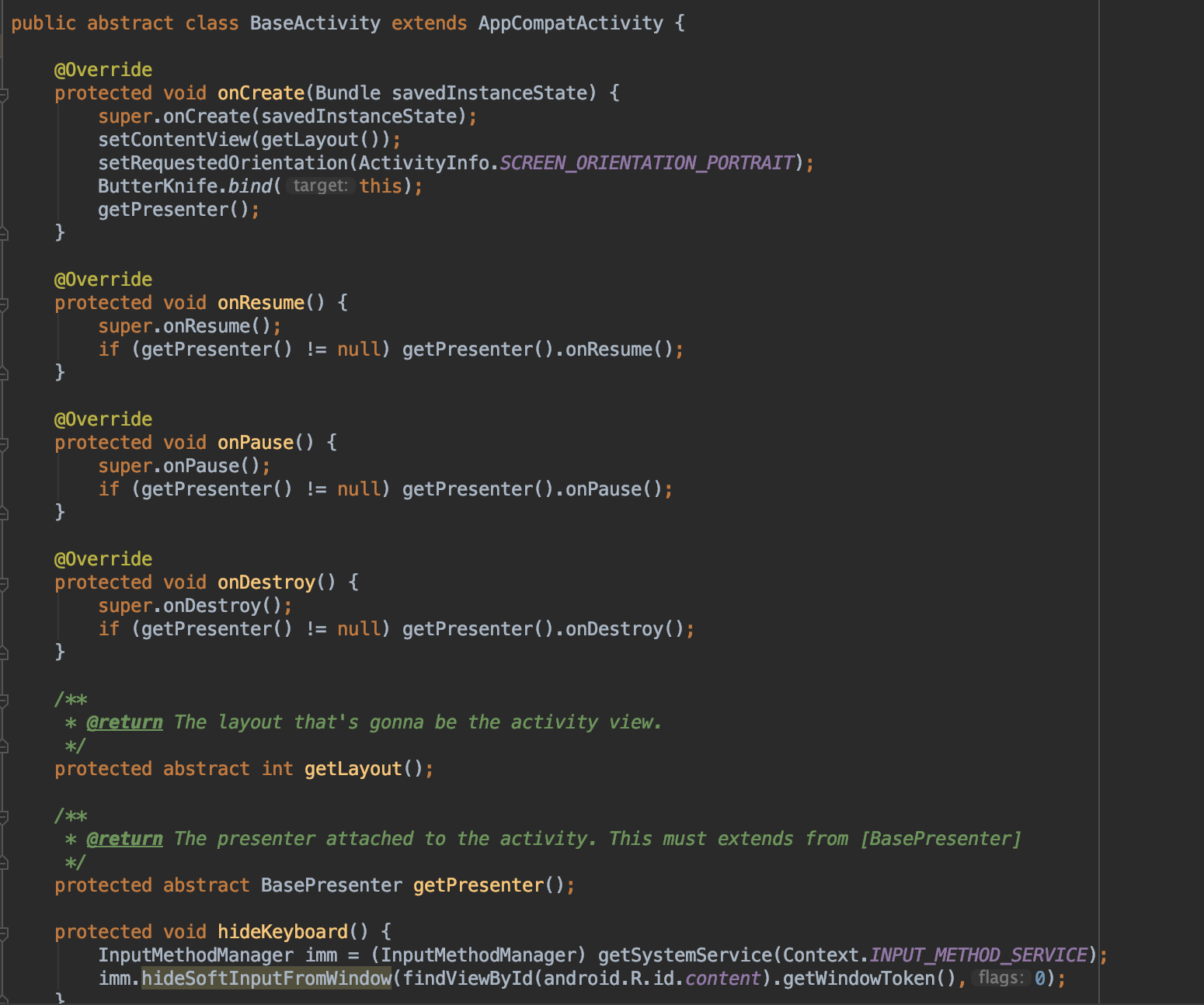
Por otro lado, a continuación, podemos observar cómo en Java hemos definido una clase entera de solo variables estáticas. Esta nos sirvió para poder tener todos los títulos para los distintos “pop-ups” o ventanas emergentes que hay a lo largo de la aplicación en un solo lugar. De esta manera, tenemos a todos estos títulos centralizados en una sola clase, y así tener mejor acceso a estos por si queremos cambiar alguno o agregar otro.



La siguiente ilustración muestra parte del código del login y se puede apreciar un ejemplo del uso de la clase Presenter, la cual es una clase que conforma el patrón de arquitectura usada para la aplicación Android. Además, se observa que se encarga de obtener los datos del login, para así, poder hacer la validación en Firebase del usuario y la contraseña. pues como dice la teoría del patrón de arquitectura MVP (Model, View, Presenter), la clase presentar se encarga de realizar la lógica de negocio de la aplicación y funciona como puente entre las clases Model y View.



En la ilustración siguiente se observa un ejemplo de los distintos “base activities” que se usó para la aplicación en Android. Esta técnica se basa en tener una clase padre o base de la cual hereda todos los demás activities que hay en la aplicación, así permitiendo emplear el concepto de herencia de la programación orientada a objetos y heredar atributos de este base activity que todos los demás activities utilizan.



# Validación y verificación del trabajo

**Plan de pruebas**

Como se mencionó anteriormente en el texto, se utilizaron diversas técnicas para validar y verificar nuestro desarrollo de la aplicación. En primer lugar, era necesario validar los requerimientos funcionales que estabas desarrollando en las aplicaciones. Por ello, realizamos un plan de pruebas para ejecutar al concluir con todas las historias de usuario, y comprobar la correcta funcionalidad de la aplicación. Asimismo, en esta definimos como comprobar cada historia de usuario mediante la especificación de las instrucciones que debe seguir el sujeto de prueba y los criterios de aceptación para el caso. Del mismo modo, el sujeto de pruebas se refiere al participante de la prueba que va a ser quien ejecute las instrucciones del documento, el cual en nuestro caso es el usuario final de la aplicación, es decir, un alumno de la Universidad de Lima.

El siguiente es el caso de prueba definido para la historia de usuario “Iniciar Sesión”:

1. Inicio de Sesión del Alumno
   1. Instrucciones:
      1. Ingresar correo electrónico (Previamente validado en el registro)
      2. Ingresar contraseña
      3. Presionar el botón “Iniciar Sesión”
   2. Criterios de Aceptación:
      1. Si todo está correcto debería de aparecer la pantalla principal de la aplicación, con sus datos previamente registrados.

Además, en el documento de pruebas se definió los riesgos que podrían afectar las pruebas a la hora de su ejecución, así como los participantes del proceso, como se mencionó en antes, y el cronograma de ejecución de las mismas.

**Análisis de métricas de código**

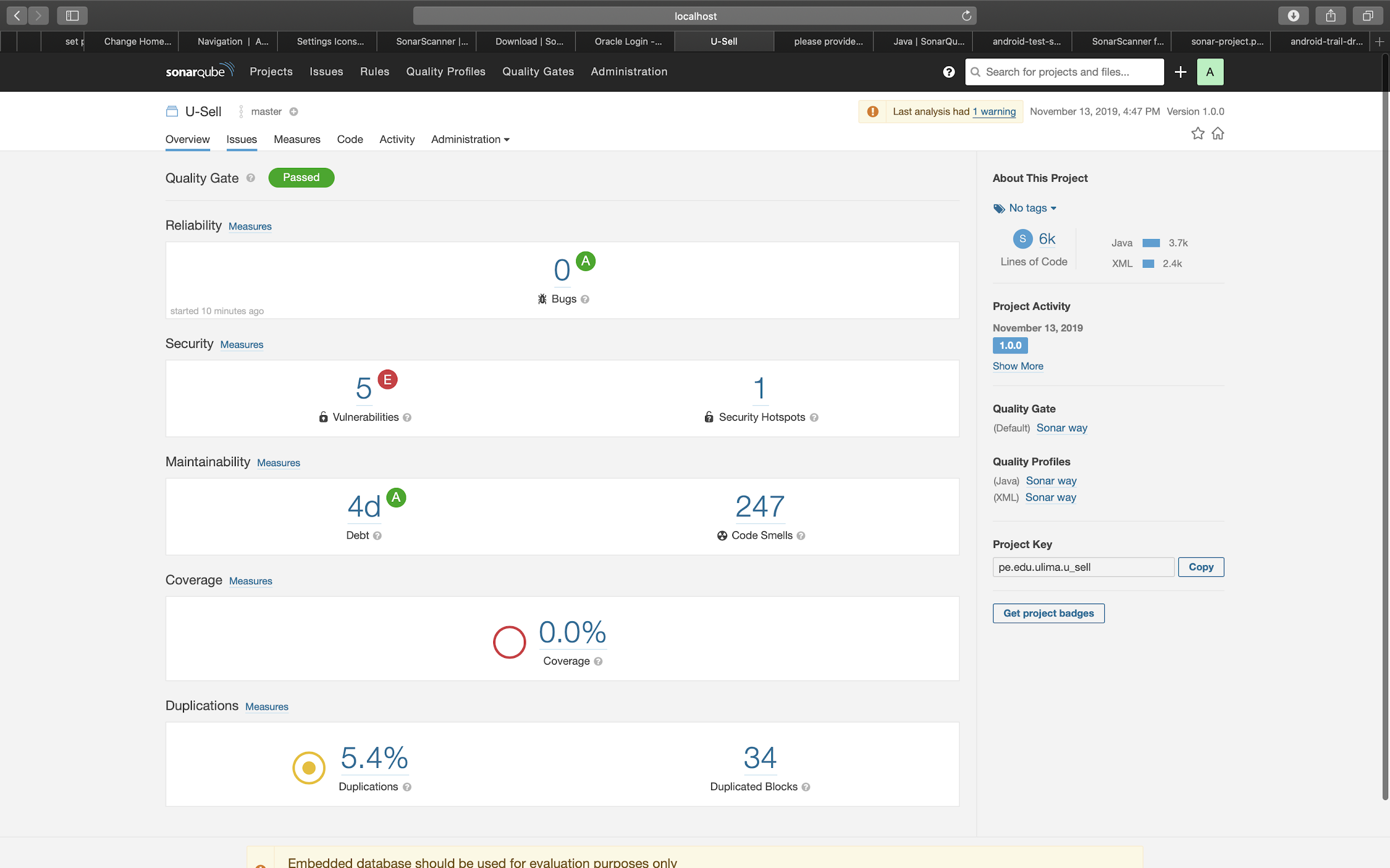
Por otro lado, para comprobar la programación que se realizó en ambas plataformas móviles, se ejecutaron pruebas de métricas de código para cada una. En ambos casos, se utilizó la herramienta de SonarQube para analizar el código fuente de la aplicación.

En el caso del código en Swift, fue necesario realizar pasos extras para poder ejecutar el programa de SonarQube. Por ello, primero se tuvo que instalar algunos paquetes extras en la computadora, como SwiftLint y Taylor. Luego, configurar el “Scanner Properties” para el SonarQube y también hacer un script en FastLane para ejecutar dicho software. Después de ejecutar estos pasos, recién se pudo ejecutar el servidor de SonarQube para poder obtener los resultados del análisis del código programado para iOS.

Asimismo, para poder ejecutar el software de análisis en Android, en el cual se utilizó el lenguaje de Java, no se tuvo que hacer pasos extras como en el caso de Swift. Por ello, se siguió la guía proporcionada por el curso de Ingeniería de Software II con las instrucciones para correr SonarQube. De este modo, luego de hacer los pasos correctamente con algunos errores de configuración al principio, se logró correr el software de análisis en Java sin mayores inconvenientes.

Sin embargo, luego de ejecutar SonarQube en ambas plataformas, nos dimos cuenta que el reporte del código en Swift no informaba acerca de la complejidad ciclomática del mismo, a diferencia del código en Java que si lo hacía. Esto fue un percance, pues era uno de las métricas que más nos interesaba analizar y comparar entre ambos. A pesar de ello, si logramos obtener un informe relevante para ambos y solo hubo algunos errores como vulnerabilidades o bugs que ya fueron arreglados luego del análisis. Además, se presentaron algunos code smells menores (errores de indentación y espacios) y algunos significativos (long parameter list, forced cast).

Resultado del análisis de métricas del código en Android.



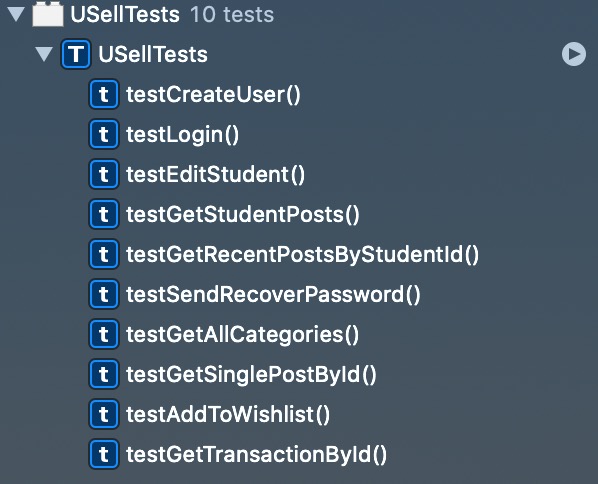
**Pruebas Unitarias**

Durante el desarrollo del proyecto también se realizaron pruebas unitarias para el código en cada plataforma. Las pruebas unitarias son pruebas automáticas que simulan la interacción entre entidades externas con el componente de la aplicación. Por ello, son piezas de código que se ejecutan para probar automáticamente el correcto funcionamiento del componente que se ha programado.

Las pruebas unitarias que se ejecutaron en ambas plataformas fueron las de:

* Crear usuario
* Iniciar sesión
* Editar usuario
* Obtener publicaciones del usuario
* Obtener últimas publicaciones por ID del usuario
* Recuperar contraseña
* Obtener categorías
* Obtener publicación por ID
* Agregar a lista de deseados
* Obtener transacción por ID

La siguiente imagen es el conjunto de pruebas unitarias en Xcode programadas en Swift.



Por ello, para programar las pruebas primero se hizo que la función ejecute un caso específico de funcionalidad de la aplicación y dentro de la misma también se ingresa directamente el resultado esperado luego de ejecutar la funcionalidad. Por ejemplo, en el test de crear usuario se hizo que la función cree un usuario con unos datos predeterminados, y lo que la prueba evalúa es el resultado de esta operación. Es decir, el test comprueba que el resultado del registro del usuario sea exitoso y que el usuario registrado tenga los atributos que se ingresó a la función.



Asimismo, luego ejecutar las 10 pruebas, todas fueron exitosas con respecto a los valores esperados y los atributos que se ingresaron en la función. Esto nos ayudó a verificar que las funcionalidades de nuestras aplicaciones están trabajando de la manera esperada sin errores en el proceso.

**Pruebas Funcionales**

Debido a que nuestra aplicación solo iba a ser programada en dispositivos móviles con sistemas operativos iOS y Android, esta tarea se nos dificultó al punto de no poder realizarla. Esto se debió a que no encontramos plataforma alguna para poder hacer dichas pruebas con el lenguaje Swift o el sistema operativo de iOS. Además, cuando recurrimos a tratar de hacer las pruebas funcionales en Android, los softwares que encontramos no funcionaban con nuestro código o simplemente no funcionaban en absoluto por estar deprecadas.

Se trató de utilizar el software de Robotium para realizar las pruebas funcionales en Android, sin embargo, esta no funcionó con nuestro código. En primer lugar, agregamos un plug-in llamado “Robotium Recorder” al IDE de Android Studio, este su suponía que nos iba a dejar grabar un script de pruebas funcionales al igual que Selenium. Sin embargo, al tratar de crear el script el recorder no nos dejó debido a bug del mismo. Esto se debe a que nuestra aplicación también utiliza partes del nuevo framework que lanzó Google hace poco llamado AndroidX. Al parecer este framework no es soportado por el software de Robotium, por lo que no nos permitió utilizarlo.

**Sistema de integración y despliegue**

Nuestro sistema utiliza un sistema de integración y despliegue automático en Heroku y un repositorio git en GitHub. Es decir, el backend de la aplicación, luego de ser programado en Python con las librerías correspondientes, es subido a un repositorio en git, del cual luego es desplegado a Heroku en la nube.

# 

Luego de que el backend este hosteado ahora en la nube de Heroku, las aplicaciones en Android y iOS ahora podrán consumir los servicios del mismo en cualquier momento. En el entorno de Heroku se pueden ejecutar los scripts necesarios para poder hacer las pruebas correspondientes antes de desplegar el programa. Por ello, ahora se dice que se tiene una integración, pues los desarrolladores del backend ahora pueden subir al repositorio nuevas versiones con los cambios respectivos, y el despliegue continuo con Heroku hará que se cree automáticamente un nuevo build de esa versión del código.

# Conclusiones

En primera instancia es conveniente indicar que el proyecto se originó como una iniciativa y a su vez como una necesidad para compra y venta de artículos, donde los mismos usuarios son los encargados de publicar y adquirir artículos deseados. Pero, como valor agregado, fue enfocado en el ámbito universitario. Es decir, un aplicativo de compra y venta de artículos universitarios, donde los estudiantes pudieran comercializar diverso material universitario, como libros, útiles, uniformes e, inclusive, comida.

La utilización de diagramas UML (Lenguaje Unificado de Modelado) nos permitió comprender la interacción entre el usuario y el sistema, así como la forma en que el sistema debe responder a las peticiones del usuario, presentando de manera visual el comportamiento esperado para iniciar la fase de programación.

En cuanto a nuestra visión personal, la realización de este proyecto nos ha demostrado aplicar de forma práctica mucha de la teoría aprendida en ciertas asignaturas, como puede ser la realización de una captura de requisitos real, la dificultad de entender a un cliente en este caso los estudiantes. Por otra parte, hemos aprendido la importancia que tiene realizar un buen diseño de base de datos, con una perspectiva global, flexible y que permita el crecimiento del aplicativo sin incurrir en cambios de estructura.

Como conclusión de este proyecto, lo más básico, pero no por ello menos importante que hemos aprendido, es que la realización de un proyecto no es algo trivial, sino que pueden surgir muchos contratiempos inesperados, y la capacidad para analizar y superar dichos contratiempos es una habilidad que un buen equipo debe de tener, además de establecer e imponer una muy buena organización y planificación.

Finalmente, se puede validar que es posible realizar un proyecto de gran importancia en muy corto tiempo, eso sí, siempre y cuando los miembros estimen y dividan sus partes de manera adecuada; para ello las metodologías ágiles ayudan a que cada uno a cumplir un rol en el proyecto. Adicionalmente, gracias a herramientas como Github nos ayudaron a llevar un control sobre las versiones y así poder llevar un registro y administrar cualquier cambio en el código del proyecto de software.

# Referencias

*AcmQueue*. (s.f.). Obtenido de <https://queue.acm.org/detail.cfm?id=3294770>

*Apple Xcode - Xcode - Apple Developer*. (s.f.). Obtenido de <https://developer.apple.com/documentation/xcode_release_notes/xcode_10_2_release_notes?language=objc>

Bauer, F. (1968). Software Engineering. *Report on a conference sponsored by the NATO Science Comittee*.

Chaudron, M., Friso, J., van-Hee, K., Hemerik, K., Somers, L., & Verhoeff, T. (2004). Software Engineering Reference Framework. *(Computer science reports: Eindhoven University of Technology, 0439*, 1-12.

Developer. (s.f.). *Swift*. Obtenido de <https://developer.apple.com/swift/>

Dimarzio, J. (2017). *Beginning Android Programming with Android Studio.* Wrox.

*Firebase*. (s.f.). Obtenido de <https://firebase.google.com/docs/database/>

Github. (s.f.). *The world's leading software development platform-GitHub*. Obtenido de <https://github.com>

Gosling, J., Joy, B., Steele, G., & Bracha, G. (2005). *The Java language specification.* Addison Wesley, 3rd Edition.

Guttag, J. (2016). *Introduction to Computation and Programming Using Python.* London, England: The MIT Press.

Heroku. (s.f.). *Heroku: Cloud Application Platform*. Obtenido de <https://www.heroku.com>

*IBM knowledge center*. (s.f.). Obtenido de Defining use cases: <https://www.ibm.com/support/knowledgecenter/en/SSWSR9_11.0.0/com.ibm.pim.dev.doc/pim_tsk_arc_definingusecases.html>

Kruchten, P. (1995). Architectural Blueprints - The “4+1” View Model of Software Architecture. *IEEE Software, 12(6)*, 42-50.

Larman, C. (1999). *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Prentice Hall.

Microsoft. (2019). *Diagramas de actividades UML: Referencia*. Obtenido de <https://docs.microsoft.com/es-es/visualstudio/modeling/uml-activity-diagrams-reference?view=vs-2015&redirectedfrom=MSDN>

Microsoft. (2019). *Diagramas de secuencia UML: Referencia*. Obtenido de <https://docs.microsoft.com/es-es/visualstudio/modeling/uml-sequence-diagrams-reference?view=vs-2015&redirectedfrom=MSDN>

Offutt, J., & Ammann, P. (2008). Introduction to Software Testing. *Cambridge University Press*.

Pressman. (2002). *Ingeniería del Software: Un Enfoque Práctico* ( 6a Edición. ed.). McGraw-Hill.

Reenskaug, T. (2003). The Model-View-Controller (MVC) Its Past and Present. *University of Oslo*.

Sharma, S., Sarkar, D., & Gupta, D. (2012). Agile Processes and Methodologies: A Conceptual Study. *International Journal on Computer Science and Engineering (IJCSE)*, 892-898.

*SonarQube*. (2019). Obtenido de <https://docs.sonarqube.org/latest/>

Studio, A. (s.f.). *Documentation | Android Developers*. Obtenido de <https://developer.android.com/docs>

Definitiva, L. G. (n.d.). *La Guía de Scrum*.

*Pontificia Universidad Javeriana, Cali Ingeniería de Software Profesora: Olga Lucía Roa*. (n.d.).

Rodrigo, N., Tutor, J., Ramón, M.-R., & Nieto Rodrigo, J. (n.d.). *“Desarrollo de una aplicación web, con Front-end y Back-end, para compraventa de segunda mano” TRABAJO FINAL DE GRADO*.