UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA DE INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a
la obtención del título de
Ingeniero de Sistemas

PROYECTO TÉCNICO:

"APLICACIÓN MÓVIL INFORMATIVA Y DE APOYO ACADÉMICO DE LA UNIVERSIDAD POLITÉCNICA SALESIANA"

AUTORES:

PABLO ANDRÉS TORRES PEÑA

BOLÍVAR ANDRÉS ANDRADE SOLÓRZANO

TUTOR:

ING. CRISTIAN FERNANDO TIMBI SISALIMA, MsC.

CUENCA - ECUADOR

2021

CESIÓN DE DERECHOS DE AUTOR

Nosotros, Pablo Andrés Torres Peña con documento de identificación N° 0105772818 y Bolívar Andrés Andrade Solórzano con documento de identificación N° 0104987136, manifestamos nuestra voluntad y cedemos a la Universidad Politécnica Salesiana la titularidad sobre derechos patrimoniales en virtud que somos autores del trabajo de titulación: **"APLICACIÓN MÓVIL INFORMATIVA Y DE APOYO ACADÉMICO DE LA UNIVERSIDAD POLITÉCNICA SALESIANA"**, mismo que ha sido desarrollado para optar por el título de: *Ingeniero de Sistemas*, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en nuestra condición de autores nos reservamos los derechos morales de la obra antes citada. En concordancia, suscribimos este documento en el momento que se hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, diciembre de 2021.

Pablo Andrés Torres Peña
C.I. 0105772818

Bolívar Andrés Andrade Solórzano
C.I. 0104987136

CERTIFICACIÓN

Yo, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: **"APLICACIÓN MÓVIL INFORMATIVA Y DE APOYO ACADÉMICO DE LA UNIVERSIDAD POLITÉCNICA SALESIANA"**, realizado por Pablo Andrés Torres Peña y Bolívar Andrés Andrade Solórzano, obteniendo el *Proyecto Técnico* que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana.

Cuenca, diciembre de 2021.

Ing. Cristian Fernando Timbi Sisalima, MsC.

C.I. 0103709911

DECLARATORIA DE RESPONSABILIDAD

Nosotros, Pablo Andrés Torres Peña con documento de identificación N° 0105772818 y Bolívar Andrés Andrade Solórzano con documento de identificación N° 0104987136, autores del trabajo de titulación: **"APLICACIÓN MÓVIL INFORMATIVA Y DE APOYO ACADÉMICO DE LA UNIVERSIDAD POLITÉCNICA SALESIANA"**, certificamos que el total contenido del *Proyecto Técnico,* es de nuestra exclusiva responsabilidad y autoría

Cuenca, diciembre de 2021.

Pablo Andrés Torres Peña
C.I. 0105772818

Bolívar Andrés Andrade Solórzano
C.I. 0104987136

RESUMEN

La Universidad Politécnica Salesiana se ha caracterizado por brindar a sus estudiantes y docentes herramientas y tecnología que les permita realizar sus actividades diarias de forma eficaz y eficiente por tal motivo se ha pensado en realizar esta aplicación para que cumpla estos propósitos, para poder iniciar el desarrollo se tuvo que realizar el levantamiento de requerimientos para eso nos valimos de encuestas realizadas en la página institucional en las cuales a los alumnos se les dio una serie de módulos para que se elijan cuáles de estos brindarían una mejor eficiencia en ciertos procesos además para que los estudiantes tengan una mayor facilidad para consultar información, entre los módulos que se seleccionaron están: chat, pagos o deudas, estado de solicitudes, carnet virtual, horarios de estudiantes, noticias y eventos de la universidad, información del estudiante, una vez realizada esta encuesta se puedo comenzar con el levantamiento de requerimientos, luego de este levantamiento se comenzó con la fase de diseño para la cual se usaron bosquejos de las diferentes pantallas para las funcionalidades de la aplicación, en la fase de implementación de la solución se eligió como lenguaje de implementación Dart con su framework flutter, se eligió este lenguaje por su eficiencia en el desarrollo de un proyecto, además de que es multiplataforma esto quiere decir que el software desarrollado para Android podría ejecutarse sin mayor problema en IOS lo único que se modificaría para esto son los permisos necesarios propios de cada sistema operativo, además otra gran ventaja de este framework es que permite que la aplicación se comporte tal como si estuviera desarrollada en forma nativa en cada una de las plataformas, la idea de usar Dart y Firebase fue probar tecnologías relativamente nuevas que nos brindaran eficiencia y eficacia para poder realizar el desarrollo de la aplicación.

ABSTRACT

The Salesian Polytechnic University has been characterized by providing its students and teachers with tools and technology that allow them to carry out their daily activities effectively and efficiently, for this reason it has been thought of making this application to meet these purposes, to be able to start the development, the requirements survey had to be carried out, for that we used surveys carried out on the institutional page in which the students were given a series of modules so that they could choose which of these would provide better efficiency in certain processes in addition to students have a greater facility to consult information, among the modules that were selected are: chat, payments or debts, application status, virtual card, student schedules, university news and events, student information, once this survey I can begin with the survey of requirements, after this survey the design phase began for which sketches of the different screens were used for the functionalities of the application, in the implementation phase of the solution, Dart was chosen as the implementation language with its Flutter framework, this language was chosen for its efficiency in the development of a project in addition to being multiplatform, this means that the software developed for Android could run without major problem in IOS, the only thing that would be modified for this are the necessary permissions of each operating system, as well as another great advantage of this framework is that it allows the application to behave as if it were developed natively on each of the platforms, the idea of using Dart and Firebase was to test technologies relatively e new ones that will give us efficiency and effectiveness to be able to carry out the development of the application.

DEDICATORIA

El presente trabajo lo dedico a mis padres, por su amor, trabajo y sacrificio en todos estos años, gracias a ustedes he logrado llegar hasta aquí y convertirnos en lo que soy. Ha sido el orgullo y el privilegio de ser su hijo, son los mejores padres.

A mi familia por haber sido un apoyo a lo largo de toda nuestra carrera universitaria y a lo largo de mi vida.

A todas las personas especiales que me han acompañado en esta etapa, aportando a mi formación tanto profesional y como ser humano.

Andrés Andrade S.

El presente trabajo correspondiente a mi titulación lo dedico principalmente a mis padres, Pablo Rene Torres y Alicia Isabel Peña, quienes me han apoyado con mi formación educativa desde mis primeros pasos hasta estas instancias, sabiéndome guiar con sus valores y sabiduría para escoger mi propio camino, mostrándome por medio del ejemplo que, con disciplina, sacrifico se logra conseguir los objetivos, siendo así pilares fundamentales en como soy ahora como persona y a su vez con todo su apoyo y paciencia en el trascurso de la culminación de carrera profesional.

A mi hermana, quien supo darme el apoyo correcto a su manera, siempre estando presente en los momentos más necesarios.

A mis familiares y amigos, quienes han sido parte de esta etapa, en donde supieron brindar una acogida grata tanto como compañero, amigo y representante estudiantil, sin muchos de ellos se habría escogido diferente camino; también a quienes de ellos que con sus ideas colaboraron para la realización de este proyecto.

Pablo Andrés Torres Peña

AGRADECIMIENTO

A mi Madre y mi Padre que con su esfuerzo y dedicación me ayudaron a culminar mi carrera universitaria y me dieron el apoyo suficiente para no decaer cuando todo parecía complicado e imposible.

A mi tutor de tesis, por haberme guiado, no solo en la elaboración de este trabajo de titulación, sino a lo largo de mi carrera universitaria y haberme brindado el apoyo para desarrollarme profesionalmente y seguir cultivando mis valores.

A la Universidad Politécnica Salesiana, por haberme brindado tantas oportunidades y enriquecerme en conocimiento.

Andrés Andrade S.

Agradezco a mis padres, por haberme dado la oportunidad de estudiar lo que realmente me gusta, por permitirme desenvolverme de la mejor manera dentro de la misma, ayudándome a superar cualquier reto que se haya presentado y solventar y superar los errores que se presentaron en el camino.

Agradezco al Economista Cesar Vázquez, ex Vicerrector de la sede Cuenca, quien fue de los primeros en apoyar y direccionar las ideas de este proyecto, y también quien fue un gran guía en el proceso de representación estudiantil, lo cual tuvo gran repercusión a la hora de tomar decisiones para fijar los objetivos de este proyecto.

Agradezco al Ingeniero Cristian Timbi, nuestro tutor de tesis quien no solamente supo direccionarnos de la mejor manera para la elaboración de todo este proyecto, si no también fue un gran docente en materias con gran exigencia, en donde esa misma exigencia que nos pedía en clases hoy en día se agradece ya que nos ha formado de tal forma que podemos afrontar cualquier reto que se nos presente.

Agradezco a cada uno de los docentes de la Universidad, gracias a cada una de sus enseñanzas sobre la materia y sobre todo a las experiencias laborales propias que nos compartieron, que han formado parte fundamental para que yo me pueda mostrarme como un profesional.

Agradecimiento al personal de la Universidad del departamento de TI y del Portal Web, quienes fueron clave en el desarrollo de este proyecto, gracias a su ayuda hemos culminado de una manera idónea los objetivos planteados.

Pablo Andrés Torres Peña

ÍNDICE GENERAL

LISTA DE FIGURAS

LISTA DE TÉRMINOS Y ABREVIACIONES

AOT: por sus siglas en ingles “Ahead-of-time”, compilado a código nativo.

Flutter: tecnología de código abierto creado por Google para crear aplicaciones móviles, escritorio y web.

JIT: por sus siglas en ingles “Just-in-time”

Landscape: orientación horizontal de un dispositivo móvil.

Portrait: orientación vertical de un dispositivo móvil.

S.O: Sistema operativo.

StatelessWidget: componente que nunca cambia su estado.

StatefullWidget: componente que conoce y maneja su estado y los cambios de este.

UI: interfaz de usuario.

UX: experiencia de usuario.

Void: vacío, generalmente usada en funciones que no retornan ningún valor u objeto.

Widget: en Flutter, es una clase que declara y construye un compone de UI.

CAPÍTULO 1. LINEAMIENTOS GENERALES

1.1. INTRODUCCIÓN

Al hablar de Universidad como tal, se consideran temas como el estudio, avances, experimentación, de igual manera de innovación; el lugar donde se genera y aplica el conocimiento, etc. Bajo esa premisa la Universidad Politécnica Salesiana (UPS) no es la excepción, ya que se ha destacado por ser pionera en la innovación educativa desde diferentes ámbitos, como lo es la academia, la investigación, el deporte y la cultura, siempre pensando en una mejora continua de la experiencia universitaria de sus estudiantes y de la comunidad universitaria en general.

Las nuevas tecnologías en el ámbito de la educación han sufrido transformaciones que a grandes rasgos han supuesto un incremento de la portabilidad, una conexión permanente, una reducción del tamaño de los instrumentos, y un aumento en la potencia y velocidad de procesado. En este sentido, actualmente, el máximo exponente es el Smartphone, que es una evolución del teléfono móvil. (Guerrero & Bautista, 2017)

La Universidad de Oviedo la que cuenta con su aplicación móvil donde toda la información pública de la universidad, como, por ejemplo: su oferta académica, ubicación, información sobre diferentes trámites, redes sociales, etc., a disposición de cualquier miembro de la comunidad universitaria y del público en general, puesto que no se necesitan credenciales para acceder a dicha información. (Universidad de Oviedo, Servicio de Informática y Comunicaciones, n.d.) Así también posibilita una Tarjeta Universitaria Inteligente Virtual, la que dispone de todos los elementos en un primer nivel: código de barras y QR e información y acceso a los servicios prestados. Además, en la aplicación se podrá encontrar diferentes

contenidos dinámicos, desde las noticias de la Universidad, eventos que se vayan celebrando o retos. La aplicación también cuenta con la sección de Ventajas, donde puedes encontrar diferentes sorteos con grandes premios y descuentos exclusivos que llamarán tu atención.

La Universidad de Guayaquil brinda una aplicación móvil para sus estudiantes, la cual es denominada “My UG”, con la cual los estudiantes podrán estar al día de su desempeño académico, consultar notas, malla curricular y horarios de clases. Consta con otras características y opciones tales como: Mapa, Directorio, Información y opción de matriculación, lo estudiantes podrán hacer uso de esta opción cada que este activo el periodo de matrícula. Para ingresar a “My UG”, en el caso de ser Estudiante de la Universidad de Guayaquil, se inicia sesión con las credenciales del SIUG. En el caso de no pertenecer a ella se podrá hacer uso de las opciones que no necesitan credenciales para acceder a ellas. (Mi UG - Universidad de GuayaquilSmart Campus, n.d.)

En el presente documento se dará una breve introducción a los conceptos de las tecnologías usadas además se indicará todo el proceso necesario que se siguió para el desarrollo de las aplicaciones tanto móvil como web de la tesis.

1.2. PROBLEMA

Antecedentes

Dentro de las instituciones educativas universitarias se realizan múltiples procesos de diferente índole; tanto académico, administrativo, publicitario, etc.; esto ha llevado que dichas instituciones busquen maneras de realizar los procesos cada vez más eficientemente, mediante el uso de tecnologías las cuales a su vez están en constante evolución.

La idea del uso de la tecnología para automatizar procesos o mejor la eficacia de los procesos ya existentes, está siempre presente independientemente de la finalidad, y en este caso no es la excepción, ya que; la Universidad Politécnica Salesiana en su búsqueda de la mejora continua, busca brindar a sus estudiantes y docentes tecnología que ayude a realizar los procesos internos de una manera más ágil; donde procesos como carnetización y comunicación efectiva entre docentes estudiantes fuera de clases han presentado dificultades. La problemática nace en cómo hacer que una nueva herramienta tecnológica (aplicación móvil) de la institución universitaria sea de utilidad, requerida y de mayor beneficio para todos sus usuarios, para que la misma sea utilizada por los usuarios y no quede obsoleta.

Explicación del problema

Para algunos de los procesos internos de la UPS se requiere una identificación de la persona que solicita algún insumo o servicio que brinde la misma, es por esto por lo que el proceso de carnetización de un estudiante o de un colaborador de la Universidad ha sido actividad desarrollada por la misma con gran importancia ya que se la usaría para varios procesos internos como prestamos, ingresos, uso de espacios, etc. Sin embargo, no ha tenido el impacto deseado ya que no es obligatorio en los distintos espacios, pero esto se debe a que los diferentes encargados de los espacios o quienes brinden algún servicio, están conscientes que gran parte de la población de la universidad carece del mismo, ya que el proceso para la obtención del carnet ha tenido importantes dificultades. Así también, una correcta interacción entre los usuarios y los procesos universitarios mejora notablemente la experiencia académica y estudiantil, por lo que es necesario contar con un sistema que integre diferentes elementos académicos e informativos como calificaciones y horarios.

Por otra parte, el uso de una aplicación móvil con finalidad de la mejora de los procesos universitarios ha sido anhelada y requerida por parte de la comunidad, de manera especial por los estudiantes los cuales cada vez más usan un teléfono inteligente (smartphone) como una herramienta funcional para el desarrollo de sus actividades tanto cotidianas como académicas.

Importancia y alcances

La aplicación de la Universidad Politécnica Salesiana está dirigida a los miembros de la misma tanto estudiantes como docentes de las tres sedes siendo un aproximado de 26.000 usuarios, brindándoles una mejor experiencia a la hora de realizar determinados procesos dentro de la institución, mejorando así la eficiencia y dinamismo para con sus usuarios, sin embargo, no quita que personas externas a la universidad puedan hacer usos de la misma ya que también tendrá parte informativa tanto de la institución como de sus diferentes eventos y participaciones; otorgando también una mejor imagen como institución hacia el exterior.

1.3. OBJETIVOS

1.3.1. Objetivo General

Diseñar, desarrollar e implementar una aplicación móvil informativa y de apoyo académico de la Universidad Politécnica Salesiana.

1.3.2. Objetivos Específicos

1. OE1. Estudiar los fundamentos de desarrollo de aplicaciones móviles y definir los requerimientos a ser considerados en la aplicación móvil.

2. OE2. Diseñar y desarrollar la interfaz de la aplicación móvil a fin de que cumpla con los lineamientos gráficos de la Universidad.
3. OE3. Desarrollar la aplicación móvil de acuerdo con los requerimientos definidos, considerando la integración con los sistemas de información institucional.
4. OE4. Validar el correcto funcionamiento de la aplicación por medio de la ejecución de las diferentes pruebas de software.
5. OE5. Elaborar la documentación de la aplicación desarrollada.

CAPÍTULO 2. FUNDAMENTOS TEÓRICOS

2.1. PLATAFORMAS EN DISPOSITIVOS MÓVILES

Los diferentes sistemas operativos están orientados a cumplir necesidades específicas de cada uno de los dispositivos móviles como son los Smartphone, Tablet entre otros; dichos dispositivos móviles son aparatos electrónicos en su mayoría de pequeño tamaño con diferentes capacidades de procesamiento y conexiones permanentes o intermitente a una red (2009). Oliver en su trabajo concluyó que las diferencias entre los distintos tipos de plataformas evidencian sus ventajas y desventajas entre las mismas, en términos generales ninguna de ellas se puede considerar superior a sus competidores.

Con el constante avance tanto de hardware como software en los Smartphone, es lógico pensar también que sus sistemas operativos han evolucionado, además de otros sistemas operativos que han ido quedando obsoletos hasta ya no usarse. (Oliver,2019).

IOS

Este sistema operativo ha sido desarrollado para el iPhone por la empresa Apple, aunque también se utiliza actualmente en el iPad, que es una tableta basada en el iPhone que podríamos considerar como un intermediario entre el iPhone y la computadora MAC propiedad de la misma empresa. Su primera versión fue presentada en 2007 siendo el primer Teléfono Móvil completamente Smartphone presentado, actualmente se encuentra en la versión número 11. Se encuentra en segundo lugar, por detrás de Android, con respecto a la cuota de mercado mundial. (Días., 2016)

ANDROID

Se basa en el Kernel de Linux, fue creado por Android Inc. que posteriormente fue comprada por Google. Está diseñado para smartphones en su principio, pero con el constante avance de la tecnología actualmente podemos encontrarlo en varios equipos como son relojes, televisores entre otros. Fue presentado en 2007, y actualmente va por la versión 11. A En la actualidad como se explicó anteriormente es un sistema operativo que se lo encuentra en varios equipos electrónicos convirtiéndolo en el sistema operativo móvil más usado del mundo. (Días., 2016)

2.2. PLATAFORMAS DE DESARROLLO

El desarrollo de aplicaciones es un campo de gran crecimiento, con el objetivo de facilitar el progreso y la implementación de las aplicaciones móviles. En la actualidad se destacan tres tipos plataformas de desarrollo, siendo estas:

NATIVAS

Las aplicaciones que se desarrollan de forma nativa tienen archivos ejecutables que se descargan directamente al dispositivo por medio de las diferentes tiendas propias para cada sistema operativo y se almacenan localmente. La instalación de estas aplicaciones generalmente lo realiza directamente el usuario en su dispositivo móvil, en algunos casos cuando se trata de aplicaciones empresariales lo realiza el departamento de TI de la empresa, además existen otros métodos que a veces ofrece el proveedor de estas aplicaciones para su descarga e instalación, una vez que la aplicación ha sido instalada en el dispositivo, el usuario la ejecuta como cualquier otro servicio del dispositivo, estas aplicaciones desarrolladas de

forma nativa pueden acceder libremente a todas las APIs que el proveedor del SO ponga a disposición además por lo general, tiene características y funciones únicas que posee ese SO móvil en particular. El proceso de desarrollo suele ser diferente para los distintos sistemas operativos cuando se trata de una aplicación nativa ya que cada empresa dueña del SO tiene su leguaje de desarrollo especifico, además de lo expuesto anteriormente cabe recalcar que el SDK es específico de la plataforma, y cada SO móvil viene con sus propias herramientas. (Días, 2016)

WEB

En la actualidad existen dispositivos móviles que cuentan con navegadores potentes los cuales brindan soporte a varias funcionalidades nuevas presentes en HTML5, algunas de estas funcionalidades son Cascading Style Sheets 3 (CSS3) y JavaScript de avanzada. Con estas funcionalidades presentes, HTML5 marca la transformación de esta tecnología desde un "lenguaje de definición de páginas" a un estándar de desarrollo de aplicaciones. (Días, 2016)

Como ejemplos del potencial y de las características presentes en HTML5 son: componentes de interfaz de usuario avanzados, accesos a diferentes medios, servicios de GPS y disponibilidad offline. Al emplear diferentes características ya existentes y otras que aún se encuentran desarrollándose, se pueden crear aplicaciones usando tecnologías basadas en la Web. Algunas empresas mejoran la UI creando un sitio Web que se asemeja a una aplicación nativa y se puede ejecutar a partir de un acceso directo. (Días, 2016)

HIBRIDAS

Este enfoque de desarrollo combina las dos tecnologías vistas anteriormente, los desarrolladores escriben la mayor parte de la aplicación basándose en tecnologías Web lo cual las permite ser ejecutadas en la mayor parte de sistemas operativos además mantienen el acceso a las APIs nativas cuando necesitan hacer uso de estas. La parte nativa de estas aplicaciones

emplea APIs para crear un motor de búsqueda HTML incorporado que funciona como un puente entre el navegador y las APIs del dispositivo, permitiendo de esta forma que la aplicación híbrida aproveche todas las características que ofrecen los dispositivos modernos, este desarrolladores puente puede ser codificado directamente por los desarrolladores o bien usar soluciones ya construidas y probadas, como PhoneGap, que es una biblioteca de código abierto que provee una interfaz JavaScript uniforme para funcionalidades distintos tipos de dispositivos móviles, que son iguales en todos los sistemas operativos. . (Días, 2016)

2.3. TECNOLOGÍAS DE DESARROLLO

FLUTTER

Es un SDK de código abierto, además es un framework para Dart desarrollado por Google, está compilado con librerías de ARM C/C++15, trabaja con componentes preconstruidos que podemos ir adaptando en forma de contenedores para realizar el diseño de la aplicación que posteriormente se compilará y se lanzará como una aplicación nativa para los sistemas operativos, haciendo que la aplicación sea realmente una App nativa y no una web disfrazada. Estos componentes con los que trabaja reciben el nombre de widgets, los cuales son utilizados por Dart y Flutter para construir la interfaz de usuario, además se los puede usar para crear nuestros propios widgets personalizados y por tanto más completos. (Martinez, 2019). Flutter es un lenguaje declarativo ya que cada que un estado de la interfaz cambia la vuelve a reconstruir para poder cambiar dinámicamente partes de ese componente.

FIREBASE

Es una plataforma desarrollada por Google cuya función es agilitar el desarrollo de aplicaciones proporcionando un servidor back-end, Firebase puede ser usado en varias

plataformas: Android, IOS y Web. No solo brinda soluciones efectivas a los problemas de desarrollo, sino que también brinda soluciones a medida que aumenta el número de usuarios del servidor mientras crece la base de usuarios de la aplicación. (García, 2017)

Entre sus funciones tenemos el servicio de autenticación, Base de datos en tiempo real, almacenamiento de archivos, reparación de errores, funciones de back-end, obtener y medir las estadísticas recopiladas de los usuarios. (García, 2017)

Firebase tiene diferentes funciones, básicamente se puede dividir en tres categorías: desarrollo, crecimiento (Grow) y monetización (Earn), además podemos agregar una más que sería análisis. (Lopez, 2020)

A. **Desarrollo**. -Incluye los servicios necesarios para desarrollar aplicaciones tanto web como móvil. Posee características que ayudan a acelerar el proceso, porque algunas actividades son atendidas por Firebase, mientras que otras pueden optimizar todos los aspectos para lograr la calidad deseada. (Lopez, 2020)

B. **Real time DB.** - La base de datos en tiempo real es una de las herramientas más destacadas y esenciales de Firebase. Los datos se almacenan en la nube, no son SQL y se los almacena con formato JSON. Permiten almacenarlos y obtenerlos en tiempo real además guarda la información de la aplicación, se pueden mantener actualizados incluso si el usuario no realiza ninguna operación. Cuando los datos cambian Firebase enviará automáticamente el evento a la aplicación y almacenará los nuevos datos en el disco. Incluso si no hay conexión de usuario, los usuarios restantes pueden usar sus datos, y una vez que se restablezca la conexión, los cambios realizados se sincronizarán. (Lopez, 2020)

C. **Almacenamiento en la nube.** - Otra funcionalidad con la que cuenta Firebase es un sistema de almacenamiento, donde los usuarios podrían guardar los archivos de

sus aplicaciones (y vinculándolos con referencias a un árbol de ficheros para mejorar el rendimiento de la aplicación) y sincronizarlos. Es una herramienta personalizable mediante determinadas reglas.

Este almacenamiento es de gran ayuda para tratar archivos de los usuarios (por ejemplo, fotografías que hayan subido), que se pueden servir de forma más rápida y fácil. También hace la descarga de referencias a ficheros más segura. (Lopez, 2020)

D. **Cloud Mesagging.** - Su principal utilidad es la de poder enviar notificaciones a los usuarios ya sea desde Firebase o desde un servidor desarrollado.

SERVICIOS WEB – API REST

Los servicios web basados en REST (Representational State Transfer) son servicios web mucho más ligeros que los servicios basados en wsdl usando URL's como su elemento principal. Podemos decir que estos servicios consisten en URL's a las que podemos acceder, por mediante diferentes protocolos de internet siendo uno de ellos el protocolo HTTP, para obtener información o realizar alguna operación. El formato en el cual se realiza el intercambio de la información lo decidirá el desarrollador del servicio. (Introducción a los Servicios Web RESTful Dept. Ciencia de la computación, 2017)

JBoss

Es un servidor de aplicaciones de código abierto, preparado para la producción y certificado J2EE 1.4, ofrece una plataforma cuya principal característica es el alto rendimiento en aplicaciones de e-business. Combina una arquitectura orientada a servicios SOA, tiene una licencia GNU JBoss AS puede ser descargado, utilizado y distribuido sin restricciones por la licencia. (Marc Fleury, 2011)

JEE

Es una tecnología construida sobre la plataforma Java SE junto con jdk. Java EE proporciona un API y un entorno para poder desarrollar y ejecutar aplicaciones empresariales de gran escala, estas aplicaciones tienen la característica de ser multicapa, escalables, fiables y seguras. Estas aplicaciones se denominan Enterprise ya que se diseñan y desarrollan para resolver algunos problemas que presentan grandes empresas además este tipo de aplicaciones brindan características que benefician también a individuos y organizaciones pequeñas que quieran hacer uso de un mundo cada vez más conectado en red y que tienen previsto expandirse, algunas de estas características son características son potencia, fiabilidad o seguridad, también las hacen complejas a la hora de diseñar. La plataforma Java EE, está diseñada para reducir esta complejidad permitiendo a los desarrolladores concentrarse en la funcionalidad. (Mestras, 2014).

CAPÍTULO 3. MARCO METODOLÓGICO

3.1. Flutter generalidades

Flutter es un SDK móvil de código abierto, el cual ha sido desarrollado por Google, para ser usado por el lenguaje Dart. Al usar dicho lenguaje de programación, se estará "compilado con librerías de ARM C/C++15, que trabaja con componentes preconstruidos que podemos ir adaptando a forma de contenedores, para realizar el diseño de la aplicación, que posteriormente se compilará y se lanzará en el sistema nativo", siendo así esta una de las principales ventajas a la hora de desarrollar esta aplicación ya que no es una aplicación web disfrazada de una aplicación móvil, sino termina siendo realmente una aplicación Nativa.

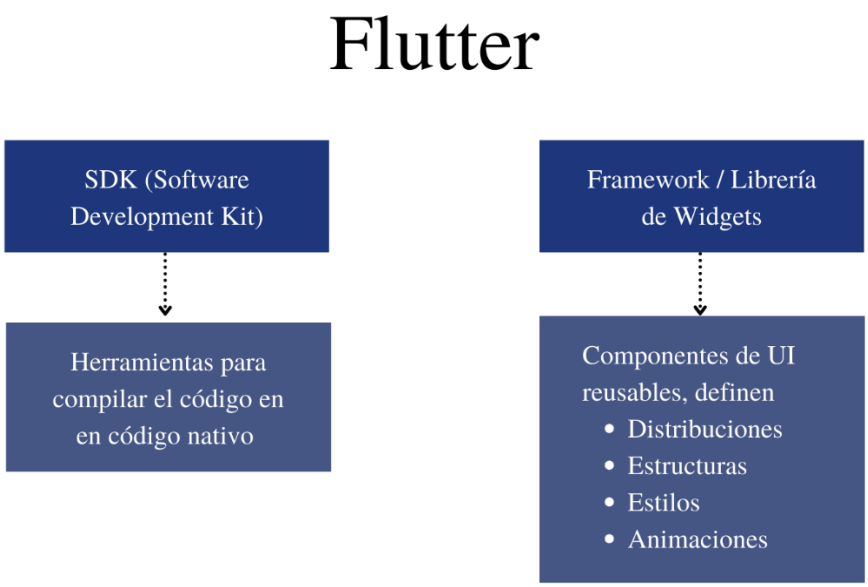

Ilustración 1. - Que es Flutter

Analizando más de cerca de Flutter podemos decir que es una combinación dos términos principales. El primero es el de SDK (Software Development Kit), lo que se entiende como una colección de distintas herramientas que permite escribir o usar una base de código de un

determinado lenguaje en este caso Dart, ya que incluye herramientas para compilar dicho código que no se ejecutarían en las plataformas de iOS o Android normalmente, es por ello por lo que debe compilarse en código de maquina nativa de cada plataforma.

Por otro lado, se lo considera como Framework o una librería de Widgets para el lenguaje Dart, lo que permite con sus componentes la creación de aplicaciones visualmente atractivas.

Lo que resulta que Flutter sea realmente beneficioso ya que hace posible cumplir con la frase “Write once, and deploy everywhere”, es decir con un solo código podemos desplegar nuestra aplicación en diferentes plataformas movibles como Android, iOS e incluso en entorno Web y entorno escritorio, siendo la última aún en fase Beta. Esta característica es a la hora de desarrollar nuestra aplicación o cualquier otro resultado hoy en día, entre las mejores alternativas; ya que por un lado se acortan los tiempos de desarrollo y se puede abarcar mayor número de usuarios independientemente del entorno en donde ejecuten dicha aplicación.

A lo largo de estos cortos años, Flutter en conjunto con Dart han ido teniendo mayor acogida en los desarrolladores, a su vez, estos han ido evolucionando con el fin de mejor su estructura, ya que, se encuentran en constante crecimiento a la par de su comunidad. Actualmente Flutter se encuentra en la versión 2.2, la cual ha tenido cambios significativos a sus predecesoras.

3.1.1. Dart y porque Flutter lo usa

Dart es un lenguaje desarrollado y mantenido por Google, el cual tiene entre sus principales características “Just-in-time” (JIT) y “Ahead-of-time” (AOT), y se centra más en la creación de interfases de usuario “front-end”, con una sintaxis que se asemeja a una mezcla entre JavaScript, Java y C#.

AOT: compilación que convierte a Dart en un rápido y eficiente código nativo, lo que hace que Flutter tenga una ganancia tanto para el desarrollador como para el usuario. También, hace que casi todo el código de Dart sea totalmente personalizable.

JIT: compilado a una velocidad excepcional de desarrollo, lo que incluye el "Hot Reload" y "Hot Restart", lo que da una facilidad a la hora de desarrollar la aplicación.

Dart, es un lenguaje orienta a objetos, predecible y productivo haciéndolo fácil de aprender, y resulta muy familiar para aquellos que viene de otros lenguajes. Ya que, tanto Flutter como Dart son desarrolladas por Google, se puede asegurar una muy buena compatibilidad entre ellas, lo que hace posible la creación de animaciones y transiciones que corran a 60fps (frames per second).

3.1.2. Conceptos básicos.

Se considera a Flutter como un motor de renderizado completo, ya que crea una UI juntando grupos de componentes o widgets. Todo componente de una aplicación es un Widget, los cuales a breves rasgos son clases de Dart que describe una vista como componente de interfaz gráfica, así como también la estructura, distribución de los componentes, estilos, animaciones o todo lo que construya un componente de interfaz para el usuario. Todo es un widget dentro de otro widget, algunos de estos pueden tener diferentes estados, como la cantidad de su valor, cuando el valor de dicho estado cambia el framework recibe una alerta para comparar el estado anterior con el nuevo para cambiar o redibujar el widget según su nuevo valor de su estado. Por ende, una aplicación desarrollada con Flutter se compone de un árbol gigante de Widgets, que dictan la configuración estructura y datos de estos.

3.1.3. Estructura de un proyecto

Cuando creamos un nuevo proyecto con Flutter, nos encontramos con un gran directorio, las cuales muchas de ellas al final no serán de mayor uso, ya que en su mayoría son principalmente como está constituida la aplicación para cada una de las plataformas de manera nativa.

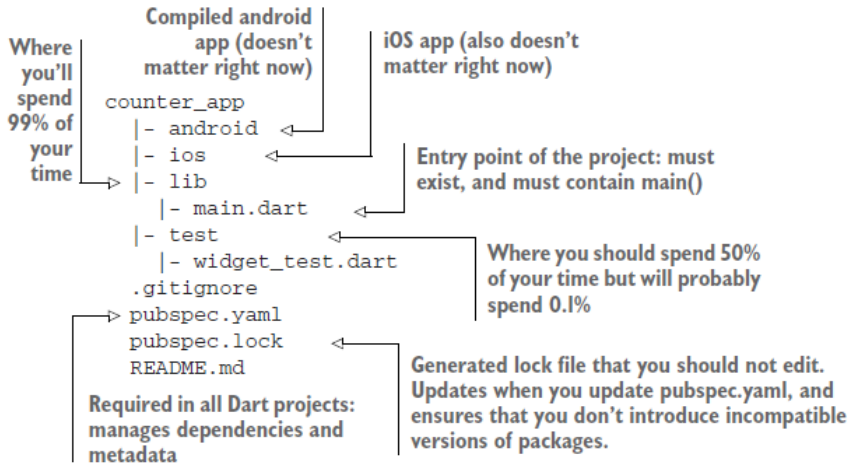

Ilustración 2. - Flutter project structure. (Windmill & Rischpater, 2020)

Como indica Windmill, en su libro la mayoría de los directorios de la estructura de un proyecto no son usados en la mayoría de los casos, como se aprecia en la ilustración el directorio "lib" es el de mayor uso, ya que este es donde se encontrará todo el código de la aplicación, tanto la parte gráfica como lógica de esta. Otra parte principal de la estructura es la de "test", donde se realizarán diferentes pruebas al código, como se observa se debería tomar más atención a esta parte, pero en muchos de los casos es una parte que se deja de lado.

Ilustración 3. - Estructura actual de un proyecto con Flutter 2

Como apreciamos en la imagen de un proyecto creado actualmente con la versión de Flutter 2, observamos que el directorio “web” ya está presente, lo que nos indica que ya está estable para correr en entorno web. Este directorio como “Android” e “iOS” generalmente no son usadas, salvo que las funcionalidades de la aplicación así lo requieran. Ya que estos hacen referencia a la aplicación reconstruida en código nativo respectivamente para su plataforma, de los cambios que se podrían hacer, a estos incluyen como requisito de permisos que debe tener una aplicación para cumplir cierta función, como el acceso al GPS, también y lo más importante; en estos directores es donde podremos encontrar los archivos necesarios para correr algún código nativo necesario y adicional para la aplicación.

Los programas desarrollados con Dart usan la función `main` como punto inicial para ejecutar dicho programa y una aplicación creada con Flutter; no es la excepción, es por esto por lo que Flutter envuelve en un nivel superior a esta función en el método `runApp`, en donde además de llamar a la función main, en ella se podrán realizar más acciones en el caso de que una aplicación lo requiera debido a su complejidad. Recordando que todo en Flutter es un widget, el main no es ni un objeto ni clase especial, ya que; lo que retorna es un widget.

3.2. Widgets

“En la mayoría de los otros frameworks, especialmente en la web, los widgets se denominan componentes, … es una clase que define un elemento específico de UI” (Windmill & Rischpater, 2020), así es como Windmill nos hace una referencia rápida a lo que es un widget, teniendo en cuenta, que a diferencia de los otros frameworks en donde los componentes solo representan partes de UI, los widgets de Flutter pueden definir diferentes características o

aspectos. Incluso el tema de la aplicación en donde se define colores, estilos, y entre otros aspectos, también se denominan widget.

Entre los widgets predefinidos por Flutter, existen aquellos que definen la distribución de estos tales como “Row”, “Column” entre otros, que ayudan a una distribución ordenada de cada componente de la aplicación. También, hay Widgets abstractos, que tienen definida una estructura inicial, entre estos encontramos a botones o cajas de texto.

Todo el conjunto de widgets de una aplicación ya sea widgets de que definen estructura, distribuciones, estilos y de más características conforman lo que se denomina **“Árbol de Widgets”**, siendo una forma clara de cómo está conformada la estructura de la aplicación, donde a partir de un nodo principal se ramifican el resto de los nodos, entendiéndose a nodo como un widget.

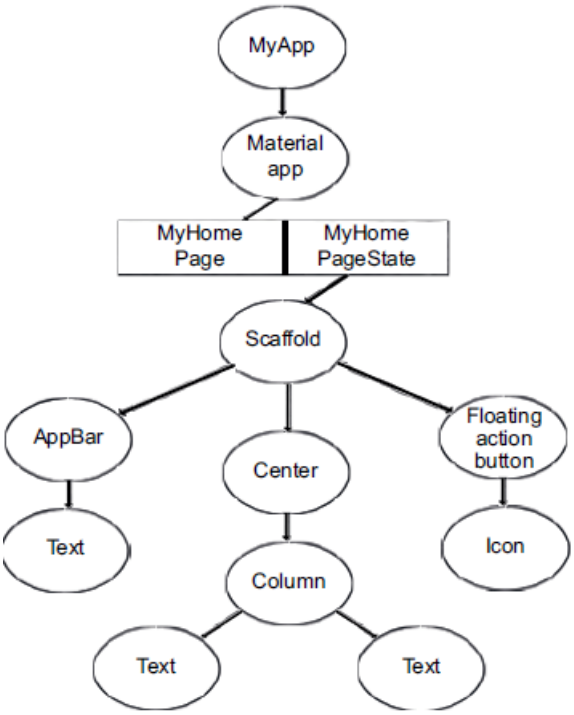

Ilustración 4. – Árbol de widgets. (Windmill & Rischpater, 2020)

En la ilustración 4, observamos la representación clara de un árbol de widgets, misma que hace referencia a la estructura de widgets de la aplicación, por defecto que viene cuando se crea un nuevo proyecto de Flutter; en la cual se aprecia como las relaciones que tienen cada nodo como una relación padre – hijo, y todos son ramificaciones de un nodo principal. Hay que tener en cuenta, que no todos los widgets pueden tener a otros widgets como hijos.

3.2.1. Tipos de Widgets

Flutter ya trae consigo un vasto número de widgets integrados en su librería, es por lo que se dice, que es un Framework bastante completo. Estos widgets están clasificados en dos tipos: “StatelessWidget” y “StatefullWidget”, siendo clases abstractas que se diferencian principalmente en que: el primero es indiferente al estado y sus cambios, mientras que el segundo; está enfocado totalmente a ello.

StatelessWidget

Como su nombre indica, no tiene ningún estado que pueda cambiar durante su ciclo de vida. Es un widget inmutable, es decir; que su configuración o valor inicial no cambiara y esta puede ser pasada desde su padre, o configurada desde dentro del mismo widget al inicio. Es por esto, que se lo considera como un widget estático que carece de algún tipo de método o lógica que le permita directamente que este se pueda actualizar por sí mismo.

StatefullWidget

A diferencia del StatelessWidget, este si posee un estado interno, pero aún más importante es que permite administrar el mismo. Sin embargo, comparte la característica de

que también es inmutable, pero logra cambiar su estado a pesar de ello ya que tiene un objeto asociado al mismo, y este es de tipo “State”, se pude asumir como al StatefullWidget y a su objeto State como una sola entidad. Este objeto si pose la carterista mutable, el cual puede mantener el estado incluso cuando Flutter redibuja al componente.

Al ser una clase abstracta que retorna un widget, ya tiene integrado varios métodos destinados a la administración del estado, los cuales obviamente se pueden sobrescribir para realizar alguna función específica, entre los métodos principales del Objeto State y más usados encontramos:

setState (): a este método se llama así cuando; se sabe que el estado de este cambia o va a cambiar, por lo que, a la aplicación se le indica que redibuje al widget que retorna y cuando es redibujado lo hará con el nuevo estado. Cabe recalcar, que no puede ejecutar una operación asíncrona, se requerir hacerlo se deberá ejecutar el método asíncrono antes de llamar al setState.

initState (): a este método se lo denomina automáticamente cuando; el widget es construido, por ende, cuando es agregado al árbol de widgets, en este método podemos inicializar cualquier dato necesario considerando como el estado inicial del widget, el cual lo establecerá antes de ser dibujado en pantalla.

3.2.2. Método Build y BuildContext

Por un lado, tenemos al método `Build`, el cual está presente en cada widget que se crea dentro de una aplicación, ya sea un StatelessWidget o StatefullWidget, este método tiene como objetivo retornar otro widget. Para el caso de los StatefullWidget, encontramos a este método en la clase que extiende del objeto State, tal *t* como lo vimos previamente en su definición.

Como menciona Windmill en su libre, `BuildContext`, “es otro concepto fundamental de para la creación de las aplicaciones y tiene todo lo relacionado con el seguimiento de todo

el árbol de widgets, específicamente en su ubicación dentro del mismo" (Windmill & Rischpater, 2020). Gracias a esto es como podemos acceder a toda la información de la aplicación que está en el context, el cual como nos dice la teoría, este tiene la administración del árbol de widgets, razón por la cual el método Build, tomo como argumento al BuildContext. Como vimos este método regresa un widget, a su vez que regresa, este es agregado al árbol de widgets.

3.3. Entorno de desarrollo y creación de la aplicación.

Para el desarrollo de un proyecto de Flutter, no es necesario un sistema operativo en específico, salvo el caso de que se requiera compilar para dispositivos iOS, en tal caso se requiere su sistema operativo como base de desarrollo. A continuación, se detallará la configuración de Flutter y del entorno de desarrollo para que funcione adecuadamente en entorno Linux, específicamente en Ubuntu 20.

3.3.1. Prerrequisitos

- Java

  Completamente requerido para este entorno de desarrollo es tener el paquete JDK instalado, muchas versiones de Linux, viene incluido OpenJJDK, el cual también es útil y de no tener el mismo se lo puede instalar ya que es más fácil, sin embargo, se ha escogido trabajar con el paquete JDK (JDK – 11.0.7) propio de Java Oracle. Para lo cual deberemos descarga de su página oficial, escoger un directorio destinado en el sistema, y generar las variables globales, lo cual conseguimos con los siguientes comandos.

```
sudo update-alternatives --
install /usr/bin/java java /opt/JDK/jdk-11.0.7/bin/java 1

sudo update-alternatives --
install /usr/bin/javac javac /opt/JDK/jdk-11.0.7/bin/javac 1

sudo update-alternatives --
install /usr/bin/jar jar /opt/JDK/jdk-11.0.7/bin/jar 1

sudo update-alternatives --set java /opt/JDK/jdk-
11.0.7/bin/java

sudo update-alternatives --set javac /opt/JDK/jdk-
11.0.7/bin/javac

sudo update-alternatives --set jar /opt/JDK/jdk-11.0.7/bin/jar

update-alternatives --set javaws /opt/JDK/jdk-
11.0.7/bin/javaws

update-alternatives --set mozilla-javaplugin.so /opt/JDK/jdk-
11.0.7/jre/lib/amd64/libnpjp2.so
```

Posterior a la generación de las variables globales, se realizan las respectivas verificaciones de una instalación correcta, para este caso podemos usar el siguiente comando:

```
java --version
```

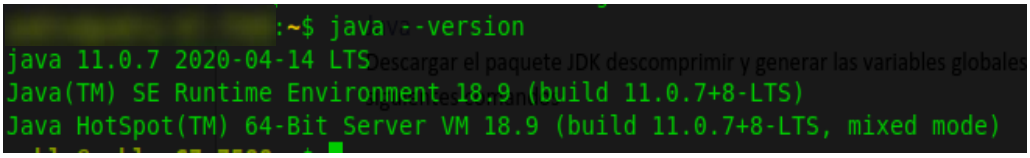

Ilustración 5. - Entorno, comprobación de JAVA

- Android SDK

  Descargamos el paquete de las fuentes oficiales y lo ubicamos de la misma manera en un directorio destinado para ello. Para generar las variables de entorno de `ANDROID_HOME`, y cual otra que sea requerida, se edita el archivo `profile`, en donde a su vez agregaremos a la variable `PATH`.

```
sudo nano /etc/profile
      export ANDROID_HOME=/home/user/Android/SDK
      export PATH=$PATH:$ANDROID_HOME/platform-tools
sudo source /etc/profile
```

- Flutter SDK

Obviamente deberemos realizar la instalación y configuración pertinente para el SDK de Flutter, teniendo en cuenta que este ya integra todo lo que compete con el SDK de Dart, así que no es necesario una instalación adicional para el mismo. En primera instancia, debemos tener presente que tiene ciertas librerías que se debe tener instaladas para su correcto funcionamiento, tales como: curl, bash, mkdir, which, etc. Mismas que en muchos casos ya vienen preinstaladas con el sistema operativo, sin embargo, hay tener pendiente.

Una vez verificado los requisitos previos, descargamos la última versión estable de la página oficial de Flutter, y de la misma forma extraemos la información y la ubicamos en el directorio destino para Flutter en nuestra máquina, para poder agregar las variables de entorno, aplicamos las siguientes instrucciones:

```
sudo nano /etc/profile
      export PATH="/opt/flutter/bin:$PATH"
sudo source /etc/profile
```

Antes de finalizar, comprobamos en primera instancia la ubicación de las variables y que las mismas estén disponibles.

```
which flutter
```

Ilustración 6. - Entorno, comprobación de variable de entorno Flutter

```
which flutter dart
```

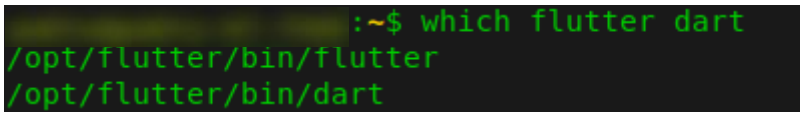

Ilustración 7. - Entorno, comprobación de variable de entorno Dart

3.3.2. Configurar Flutter

Es importante desde el inicio tener la seguridad de una correcta configuración, de nuestro entorno ya que es nuestra base durante todo el desarrollo de la aplicación. El SDK de Flutter integra una herramienta que es de gran utilidad para verificar si el mismo está en óptimas condiciones y sin ninguna dependencia faltante, y esta es `flutter doctor`, comando que lo podremos ejecutar en la consola de comandos, para visualizar toda la información.

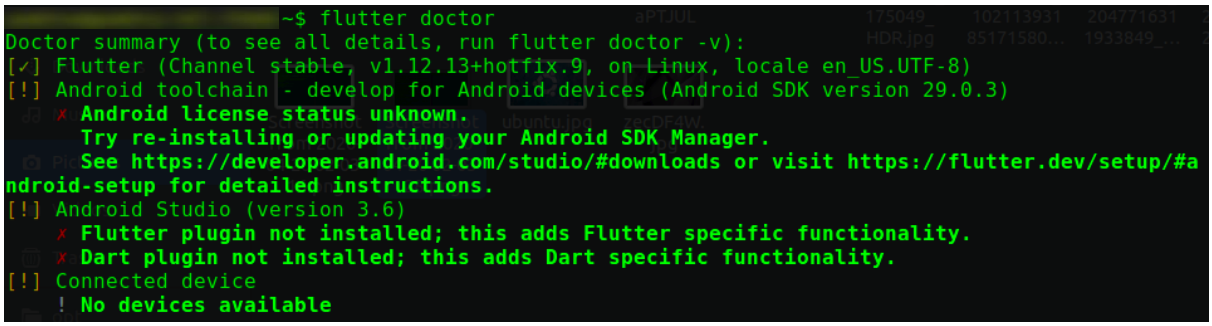

Ilustración 8. - Flutter Doctor

Como se aprecia en la Ilustración 8, observamos la configuración actual de Flutter y su relación con sus dependencias más importantes, nos muestra como errores o advertencias según el caso. Para nuestro caso, debemos solucionar la advertencia de Android License, el cual se nos presenta como advertencia y podríamos pasar por alto, sin embargo, después tendremos problemas a la hora de firmar la aplicación o usar APIs externas, por esto es por lo que presentamos la solución a este inconveniente:

```
Flutter doctor --android-licenses
```

de no funcionar, actualizar Flutter.

```
flutter upgrade

flutter doctor --android-licenses
```

3.3.3. IDE de desarrollo.

Se ha escogido, Visual Studio Code, como IDE de desarrollo por sus grandes ventas a la hora de codificar, además que tiene una gran compatibilidad con librerías de Flutter, así como snippets que nos facilitara mucho el desarrollo de la aplicación.

Para configurar Dart y Flutter, en este editor debemos instalar las extensiones propias de las mismas. Buscamos "Flutter" en el campo de búsqueda de extensiones, seleccione Flutter en la lista y haga clic en Instalar.

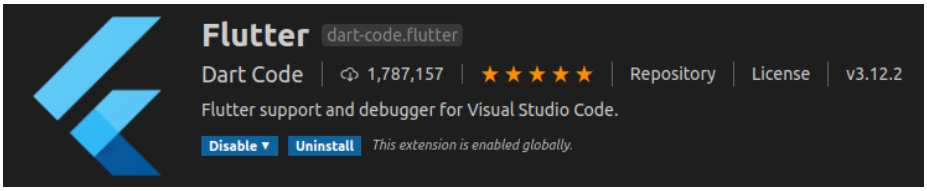

Ilustración 9. - IDE de desarrollo, extensión Flutter

Se realiza la misma extensión para instalar el complemento de Dart requerido.

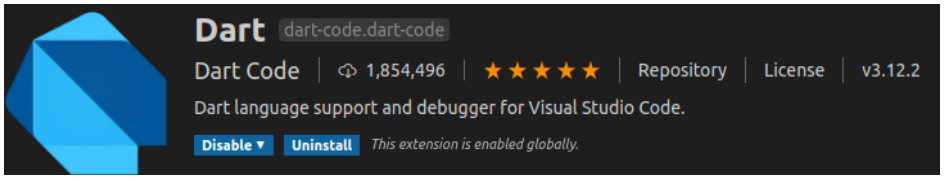

Ilustración 10. - IDE de desarrollo, extensión Dart

Como recomendación para facilitar ciertas opciones se instala el complemento siguiente.

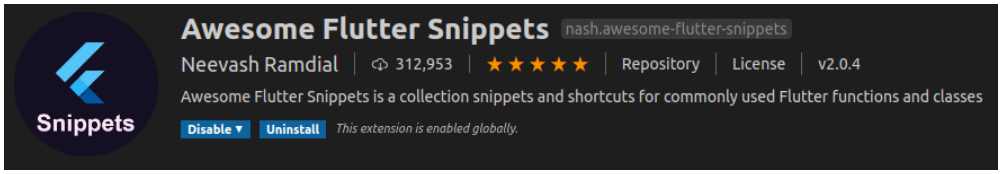

Ilustración 11. - IDE de desarrollo, extensión Awesome Flutter Snippets

Para finalizar, realizamos una comprobación con el comando flutter doctor, el cual nos valida que el IDE, esta apto para crear, codificar, y correr aplicaciones de flutter.

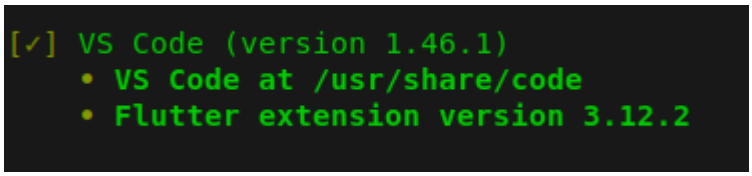

Ilustración 12.- Flutter Doctor, verificación IDE

Otras extensiones de gran ayuda dentro de Visual Studio Code:

- Bracket Pair Colorizer 2
- Material Icon Theme
- Paste JSON as Codes
- Terminal

3.3.4. Gestor de Versiones

Al desarrollar cualquier proyecto, se sabe que una buena práctica, incluso considera como requisito indispensable por muchos es el uso de Git o un gestor de versiones, y por obvias razones este proyecto no es la excepción. Para configurar Git hacemos uso del comando `git config`. Específicamente, debemos proporcionar nuestro nombre y nuestra dirección de correo electrónico, debido a que Git inserta esta información en cada confirmación que hacemos. Podemos añadir esta información escribiendo lo siguiente:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@domain.com"
```

La información que introduce se almacena en su archivo de configuración de Git. Tendrá la opción de modificarlo con un editor de texto de la siguiente manera:

```
nano ~/.gitconfig
```

Para la conversión de un proyecto existente en un entorno de espacio de trabajo, lo que se deberá hacer una vez que todos los archivos están en su espacio de trabajo dentro de un directorio determinado, se deberá indicar a Git que se desea usar su directorio actual como un entorno de git.

```
git init
Initialized empty Git repository in /home/user/git/testing/.git/
```

Para crear una nueva Branch usamos:

```
git checkout -b develop
```

Una vez que haya inicializado su nuevo repositorio vacío, puede agregar sus archivos. Lo siguiente por hacer será agregar todos los archivos y directorios al repositorio recién creado.

```
git add .
```

Ahora que se tiene configura correctamente un gestor de versiones, debemos tener presente que este es de manera local, así que también deberíamos hacer uso de uno de manera remota y así poder trabajar con todos los miembros del equipo. Para este caso se ha escogido GITHUB, como servidor de nuestro gestor de versiones remoto.

El primer paso para poder enviar código a un servidor remoto es proporcionar la URL donde reside el repositorio y darle un nombre, para lo cual haremos lo siguiente:

```
git remote add origin ssh://git@git.domain.tld/repository.gi user@host
git remote -v
```

CAPÍTULO 4. DISEÑO DE LA APLICACIÓN

4.1. Levantamiento de requerimientos.

Como cualquier otro proyecto de desarrollo, se da inicio con el levantamiento de los requerimientos para esclarecer funcionalidades, atributos o cualquier requerimiento de sistema.

Esta aplicación al estar destinada al uso de la comunidad universitaria se ha hecho mayor énfasis en las necesidades de los estudiantes y también priorizando su mayor interacción con las funcionalidades de página web institucional para que las mismas estén plasmadas en la aplicación.

Una vez que se ha definido todos los requerimientos se puede establecer de manera más precisa parámetros funcionales del sistema tal y como es su arquitectura para que de esta forma solventar cada uno de los requisitos de manera idónea y eficaz.

4.1.1. Identificación de stakeholders.

Entendiéndose como stakeholders a todo tipo de individuo que esté relacionado de alguna manera con el proyecto, es como definimos a dos grupos principales de los mismos, los cuales son:

Usuarios

Todo aquel individuo que hará uso de la aplicación como tal, así mismo indefiníamos a los posibles usuarios a todo miembro de la comunidad universitaria, entre ellos tenemos a los docentes, colaboradores y a estudiantes, siendo a estos últimos a los cuales estarán destinada las mayores funcionalidades.

Desarrolladores

Para este proyecto, estarán a cargo dos desarrolladores los cuales serán los responsables de la culminación de este cumpliendo con las funcionalidades inicialmente planteadas.

4.1.2. Requisitos Funcionales

Estos requerimientos definen que es lo que hace el sistema como tal, es decir cuáles son las funcionalidades que cumple el mismo para con el usuario. Tras el proceso de levantamiento de los requerimientos antes mencionados se han definido que la aplicación deberá contar con las siguientes características.

1º. Control de sesión de los usuarios por medio de las credenciales universitarias
2º. Sistema de mensajería instantánea.
3º. Autogeneración de grupos según materias matriculadas (estudiante) o asignadas en los distributivos (docentes).
4º. Servicio de notificaciones push.
5º. Servicio de comunicados.
6º. Visualización de noticias y eventos como parte del portal universitario
7º. Visualización de información institucional.
8º. Presentación de estados de solicitudes de estudiantes.
9º. Visualización de las calificaciones de estudiantes.
10º. Visualización del récord académico de estudiantes.
11º. Estados de cuenta de estudiantes.
12º. Horario de materias matriculadas o asignaturas asignadas en distributivo.
13º. Carnet digital por QR.

Para visualizar la descripción y mayores detalles de cada uno de estos requerimientos, véase el Anexo 1.

4.1.3. Requisitos no funcionales

Aquí definimos aquellos requisitos que indican a la aplicación como realizar determinada acción o trabajo, se hace referencia a temas de eficiencia, interfaz de usuario.

En este apartado, nos apegamos principalmente a la línea grafica de la Universidad como institución, la cual tiene sus normas de diseño como el uso de sus logos.

En cuanto al logo, se lo debe usar con fondo blanco puro y si se da el caso de que el fondo es de algún otro color, el logo deberá ser en que está en blanco y negro.

Se respeta la codificación de los colores institucionales, los cuales están presentes en la aplicación y se ha jugado con la variación de la opacidad de estos.

4.2. Definición de la arquitectura del sistema.

Posterior a la definición de los requerimientos de la aplicación se ha considerado implementar para la solución una arquitectura distribuida en N-capas, en razón a las distintas funcionalidades que tiene que realizar la misma para cumplir con los requerimientos y al uso de diferentes servidores como parte del desarrollo, dicha arquitectura la podemos visualizar en la ilustración 13.

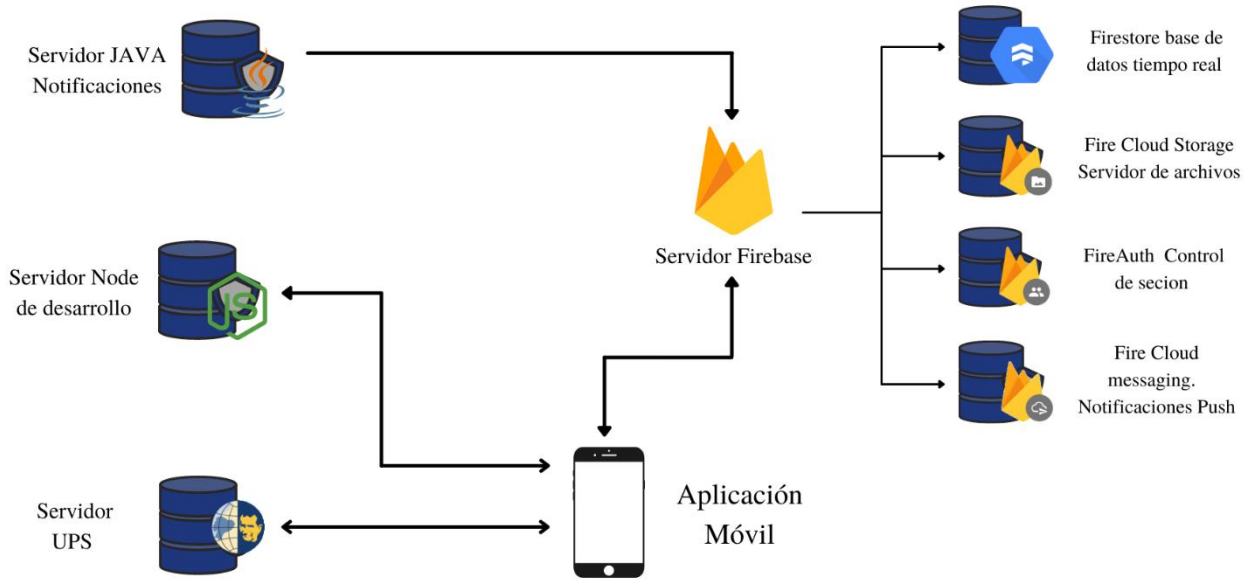

Ilustración 13. - Arquitectura del sistema.

Como se aprecia en la ilustración 13, ha sido necesario el uso de servidores tanto externos como Firebase, y desarrollados por el equipo de desarrollo de este proyecto, así como también otros por el personal del departamento de desarrollo de la universidad para brindar información institucional, los cuales todos en conjunto conforman el Backend de esta aplicación.

La aplicación tiene una interacción directa con los servicios de Firebase, como al registro de usuarios registrados en la aplicación, pero sobre todo en almacenar información correspondiente al Chat Universitario, lo que compete a texto y archivos multimedia, en donde también se hace uso del apartado de las Push Notifications. Por otro lado, tenemos al Servidor UPS y Servidor de Desarrollo, los cuales nos brindan información institucional, ya sea organizativa como especifica de un usuario.

4.2.1. Servidor UPS

Este servidor aloja todos los servicios web que competen con la información interinstitucional y los datos de los usuarios. Dichos servicios han sido desplegados en un entorno de pruebas de Swagger desarrollados por el departamento de la universidad, puesto que se ocupa información confidencial, por ende, es necesario una autenticación tipo “Basic Authorization” para hacer uso de estos. Los datos que nos devuelven estos servicios competen a la siguiente información estructurada por categorías:

Portal (Información Noticias y Eventos Universidad)

Este servicio, nos devuelve la información del portal web comunicacional de la universidad, en lo que compete a las noticias y ventos que se desarrollan en la mismas, de manera que podamos mostrar al usuario esta información e incluso aplicar filtros para segregación del contenido.

- Noticias. - nos devuelve una colección de las noticias de toda la universidad, en este mismo servicio podemos agregar parámetros para que el resultado sea infamación filtrada, en este caso por cada una de las diferentes sedes.
- Eventos. - al igual que el anterior nos devuelve información en forma de una colección de objetos, en este caso sobre los eventos.

SIGAC (Información Financiara Estudiante)

Estos tres servicios, hacen referencia a información económica de un usuario tipo estudiante, el cual podrá visualizar su información detallada sobre su situación económica con la institución.

- Facturas. - Colección de información sobre todas las facturas realizadas por un estudiante, la cual le servirá para corroborar los pagos de estas.
- Deudas. - Colección de deudas, si no tiene ninguna deuda pendiente, pues este servicio devolver un array vacío.
- Pre-facturas. - Colección de los datos de las prefecturas generadas que estarán pendientes de pago.

SOL (Información de Solicitudes de Estudiante)

Servicio web que no permite obtener la información de todas las solicitudes de un estudiante en específico que ha realizado por el sistema de solicitudes de la universidad, pudiendo constatar el estado de estas.

PED (Información Académica Estudiante)

Esta categoría hace referencia información netamente académica, nos devuelve colección de información correspondientes a las asignaturas de la malla de un estudiante, y del récord académico es decir de las asignaturas aprobadas.

ORG (Información Organizacional Universidad)

Información institucional, este servicio nos devuelve para del organigrama de la institución, lo que compete a la colección de las sedes y las carreras que encontramos en cada una de ellas.

INS (Información Personal Estudiante)

Información referente a los usuarios de tipo estudiante, esta categoría de servicio nos devuelve información personal de cada estudiante según su correo institucional.

- Usuarios. - servicio encargado de proporcionar información personal, la cual será utiliza a lo largo del sistema
- Inscripciones. - servicio que devuelve las carreras inscritas por un estudiante pudiendo ser más de una.

GTH (Información Personal Colaborador)

Información referente a los usuarios de tipo docente o colaborador, esta categoría de servicio al igual que la anterior, nos devuelve información personal de cada personal de la institución según su correo institucional.

4.2.2. Servidor NodeJS

Servidor REST con NodeJS, el cual nos devuelve información estática sobre datos organizacionales de la universidad, así también como de usuarios predefinidos como estudiantes y colaboradores. Lo encontramos dentro de esta arquitectura puesto que al final del desarrollo de la aplicación aun consume ciertos de estos servicios, ya que no se fue posible contar con la totalidad de servicios alojados en los servidores de la universidad.

Este servidor, se encuentra alojado en un dominio de HEROKU, el cual es un prestador de dominios gratuitos ideal para la funcionalidad que cumple, como servidor de pruebas de desarrollo.

4.2.3. Servidor JAVA

Para el testeo de funcionalidades como él envió de notificaciones y almacenamiento en la base de datos de la nube además para el módulo de comunicaciones se creó un servidor en JAVA con arquitectura JEE usando JBOSS en el IDE APACHE, por lo cual se tuvo que instalar la versión 14.0.1 que era la versión más estable para el uso de las librerías de FIREBASE propias para java, las pruebas del servidor se las realizaron en ambientes locales para su correcto funcionamiento y testeo, una vez que se concluyeron las pruebas locales se pasó a un ambiente remoto temporal constatando que el servidor creado funciona correctamente y posteriormente poderlo pasar a producción.

4.2.4. Firebase

Se ha hecho uso de los servicios que Firebase nos presta para el desarrollo de proyectos aprovechando sus benéficos y ventajas, este servidor termina siendo nuestro almacenamiento de los datos de la aplicación, en especial para las funcionalidades de mensajería instante. Además, que aprovechamos las funcionalidades de Messaging la cual nos permite la creación del servicio de Notificaciones push en los dispositivos móviles, así como también como repositorio de los archivos enviados por el servicio de chat.

4.3. Creación de la aplicación.

Como ya se ha visto previamente, de las ventajas más grandes de Flutter es su característica de multiplataforma, es decir una vez se crea un nuevo proyecto, el SDK de Flutter ya nos ayuda creando todos los archivos necesarios para correr en cualquier entorno móvil. Con la actualización a Flutter 2, ahora ya se crea por defecto el proyecto para ejecutarlo en entorno Web, a más de las móviles como Android y iOS. Para crear de un proyecto, podemos ayudarnos

de las opciones del IDE de desarrollo, caso contrario lo podremos realizar desde una terminal de comandos ejecutando la siguiente instrucción:

```
flutter create app_ups
```

Con este comando crearemos el directorio del proyecto de nuestra aplicación, la cual contendrá los directorios y archivos necesarios para ser ejecutados en las diferentes plataformas, como ya se hizo mención previamente, además viene integrada con una aplicación simple de demostración, la cual eliminaremos para crear desde cero nuestro proyecto.

De las utilidades más impresionantes de Flutter es el Hot Reload, el cual nos permite ir visualizando los cambios que vayamos haciendo en el código reflejados en la ejecución de un dispositivo, para lo cual una vez creada nuestra aplicación la procederemos a ejecutar de la siguiente manera:

```
flutter devices
```

Este comando nos ayuda a verificar los dispositivos disponibles que estén en nuestro computador para ejecutar nuestra aplicación, mismos que pueden ser dispositivos emulados o dispositivos físicos que estén conectados a nuestro computador de desarrollo. La salida de este comando es un listado de estos dispositivos, para lo cual escogeremos el o los deseados para ejecutar, ya que como otra ventaja es que podremos ejecutar nuestra aplicación en diferentes dispositivos a la vez, e incluso en diferentes plataformas, claro que dependerá directamente de la potencia y características del pc de desarrollo; para ejecutar la aplicación corremos el siguiente comando:

```
flutter run
o
flutter run  d all
```

Cabe recalcar que de esta forma se corre la aplicación en un modo de depuración, lo cual nos permite tener un desarrollo más eficaz gracias al Hot Reload, lo que genera en el rendimiento de esta un sobrecargo de procesamiento produciendo que nuestra aplicación en ocasiones se observen las animaciones con un aspecto de lentitud. Para ejecutar la aplicación en modo release o producción, como producto final, usaremos el siguiente comando que nos permitirá dicha configuración en la ejecución:

```
flutter run --profile
```

4.3.1. Librerías externas

Para el desarrollo de las aplicaciones es común usar, librerías o paquetes externos ya sea para evitar crear lo que ya este hecho o para usar métodos que ya han sido testeados por la comunidad e incluso validados por Flutter mismo, tal como paquetes de Firebase, manejadores de estado, etc. Paquetes que se hablaran más en específico en lo posterior.

Entre los archivos iniciales de un proyecto de Flutter, encontramos el archivo `pubspec.yaml`, el cual se lo podría tomar como un archivo de configuración para la aplicación de Flutter, en donde se especifica con que versión del SDK de flutter se compilara, además que en este archivo es donde se especificará los paquetes externos que deberá descargar para su posterior uso. También se indica rutas internas a archivos fuera del código de Dart, como pueden ser archivos multimedia o de configuración.

Las principales librerías externas que han sido usadas en este proyecto son las siguientes:

- Manejadores de estado:
    - provider: ^5.0.0
    - flutter_bloc: ^7.0.0
- Verificación estado de red:

    - connectivity: ^3.0.3
    - data_connection_checker: ^0.3.4
- Almacenamiento interno:
    - flutter_secure_storage: ^4.0.0
    - sqflite: ^2.0.0+2
- Peticiones HTTP:
    - dio: ^4.0.0-beta7
- Firebase
    - firebase_auth: ^1.0.1
    - firebase_core: ^1.0.1
    - cloud_firestore: ^1.0.1
    - firebase_storage: ^8.0.0
    - firebase_messaging: ^9.0.0
- Utilidades
    - pdf_previewer: ^0.1.0
    - flutter_cached_pdfview: ^0.4.0-nullsafety
    - flutter_local_notifications: ^5.0.0-nullsafety.1

Además de esas, también se usarán otras las cuáles, no son de carácter indispensable para el funcionamiento como tal de la aplicación, pero si aportan grandes características ya sea en la robustez de la aplicación como también en la parte visual de la misma, estas son:

- Utilidades:
    - path_provider: ^2.0.1
    - url_launcher: ^6.0.2

    - share: ^2.0.0
- Date Format
    - intl:
- UI
    - animate_do: ^2.0.0
    - pull_to_refresh: ^1.6.4
    - pretty_qr_code: ^1.0.1
    - qr_flutter: ^3.2.0
    - file_picker: ^3.0.0
    - after_layout: ^1.0.7+2
    - flip_card: ^0.4.4

4.3.2. Manejador de estados como lógica de negocio.

El estado de los componentes de una aplicación Flutter es fundamental, para su funcionamiento, por ello existen diferentes tipos de manejadores de estado que se pueden implementar, entre los más conocidos y usados encontramos a Flutter BloC, Provider, redux, flutter_meedu o Getx. Para esta aplicación se han usado los dos primeros antes mencionados, siendo el principal manejador del estado de nuestra aplicación Flutter Bloc.

Se dice que Flutter es de carácter declarativo, es decir, que construye la interfaz que se muestra al usuario para reflejar el estado en el que se encuentra en ese momento la aplicación; por lo que Flutter en vez de modificar sus componentes de UI los reconstruye, parecería que en primea instancia tomaría mayor tiempo o esfuerzo de procesamiento, pero no es así, ya que como sabes Flutter compila renderiza directamente los componentes de UI en las plataformas

de forma nativa, esto hace que sea lo suficientemente rápido para construir nuevamente el comente, incluso en cada frame por segundo.

Cuando el estado de la aplicación o de alguno de sus componentes llega a cambiar, ya sea por algún parámetro lógico o por una interacción del usuario, existe una modificación del estado lo que desencadena una configuración distinta para la UI, tal y como nos dice la documentación oficial “No hay ningún cambio imperativo de la propia interfaz de usuario (como por ejemplo widget.setText) - cambias el estado, y la interfaz de usuario se reconstruye desde cero” (Google, n.d.).

Flutter BloC

Se ha elegido a Flutter BLoC como el manejador principal de la aplicación, por su robustez, a pesar de ser considerado entre los más complejos manejadores de estados, se consideró que nos brinda mayares beneficios en la aplicación en especial en el tema de escalabilidad, dejando una estructura inicial sólida, adaptable y sobre todo entendible.

Se considera a un BLoC a una clase avanzada ya que está constituida o basada en la ejecución de un evento el cual puede o no desencadenar un cambio al estado por medio de funciones.

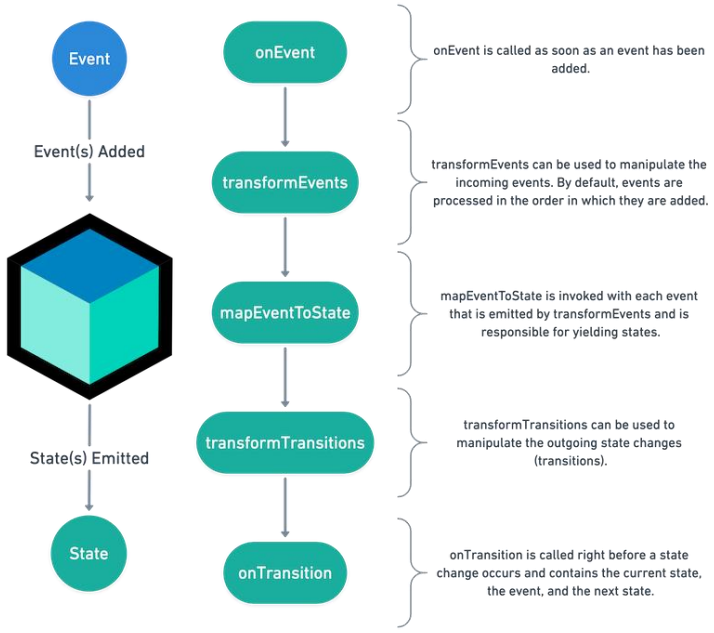

Ilustración 14. - Funcionamiento de manejo de estado con BLoC (Angelov, n.d.)

Como observamos en la ilustración 14, el funcionamiento lógico de un BLoC por el cual manejaremos el estado de la aplicación. Dentro de un bloc podemos manjar varios eventos que serán como las llamadas a la ejecución de una función. De esta forma ya sea por interacción del usuario, o de alguna acción lógica, se podrá hacer una llamada a un evento, de los ya definidos dentro del BLoC, para ser mapeados dentro del método `mapEventToState`, en donde identificaremos el evento accionado el cual podrá ejecutar ciertas instrucciones que pueden desencadenar en un cambio de un estado a otro o la modificación de uno mismo.

Ilustración 15. - Interacción de un BloC (Angelov, n.d.)

"Elaboración propia"

El uso del bloc nos ayuda a mantener separada la lógica de la interfaz, como buena práctica de programación, haciendo que el código del proyecto sea más entendible para otros programadores e incluso para nosotros mismos, facilitando la escalabilidad y mantenimiento.

Es como vemos en la ilustración 15, la interacción de la UI, con el bloc y los datos, como ejemplo práctico en nuestra aplicación, en el apartado del portal web, al inicio estará en un estado de cero noticias, llamando al evento del bloc para obtener las mismas; el bloc mapea dicho evento para ejecutar la función determina que este caso es hacer un llamado al repositorio de las noticias, este nos responde con el listado de las misas que son devueltas al bloc para que emita el cambio al estado, el cual ya tendrá la lista de noticias para ser mostradas al usuario, como componente de UI.

Para realizar todas las acciones antes mencionas es importante tener en claro ciertos conceptos sobre, ya con los cuales estaremos creándoles, usándoles, escuchándolos y demás funciones esenciales para si uso, tales como BlocProvider, BlocBuilder, BlocListener y BlocConsumer, y tal como la teoría nos dice los elementos mencionados también son widgets.

BlocProvider

El BlocProvider es un widget, que proporciona un bloc a manera de inyección de dependencias como una única instancia de este dentro del árbol de widgets, para que sus hijos puedan tener acceso a dicho bloc.

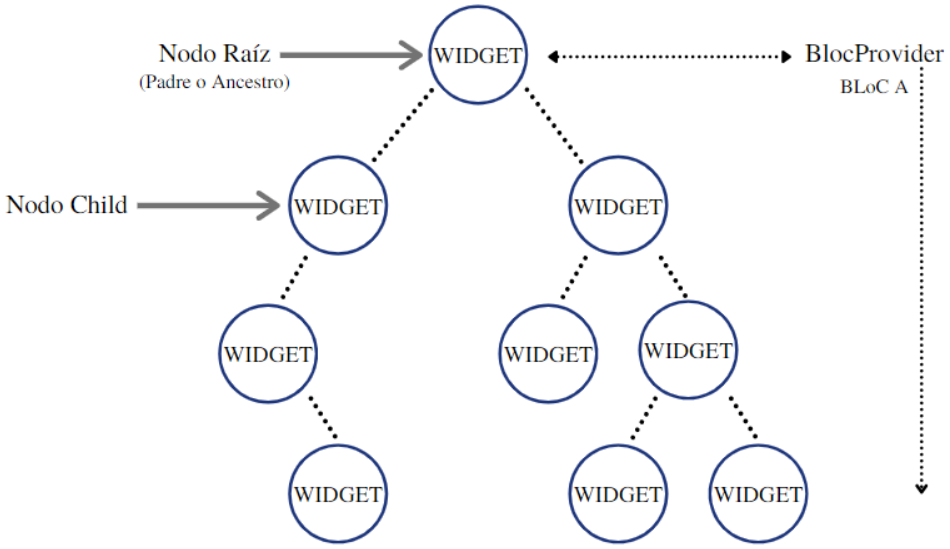

Ilustración 16. - BlocProvider creación de Bloc inserción al árbol de widgets.

Vemos como en un nodo raíz hacemos uso del widget BlocProvider para insertar al Bloc en el árbol de widgets, con la finalidad de que todos sus hijos puedan tener acceso al mismo, saber su estado y en lo posterior redibujarse por el cambio del estado.

```
BlocProvider (
  create: (BuildContext context) => BlocA(),
  child: ChildA(),
);
```

BlocBuilder

Este widget tiene gran importancia ya que es el encargado de construir o reconstruir un widget en respuesta al cambio del estado de un bloc, ya que este escucha el stream del bloc estando siempre pendiente a sus cambios.

```
BlocBuilder<BlocA, BlocAState>(
 builder: (context, state) {
  // retorna un widget específico
 }
)
```

El BlocBuilder para su creación requiere indicar el bloc que será usada y su estado, tiene como función principal `builder`, la cual es la encargada de retornar el widget que se construye y a su vez la misma función adiciona dicho widget al árbol de widgets. Su mayor importancia es que este puede envolver al widget necesario y justo para redibujar, ese decir que se centrara en reconstruir un widget en concreto de toda una pantalla y la misma, reduciendo drásticamente el tiempo de procesamiento, ya que se focaliza la acción en el elemento que realmente se requiere.

BlocListener

Tiene estructura similar al BlocBuilder, ya que este también es un widget que está a la escucha de cualquier cambio del estado del bloc, con la diferencia que este no tiene la función builder, es decir no se encarga de crear o reconstruir un widget especifico, si no que al contrario ejecuta una función void llamada `listener`, la cual es llamada una sola vez por cambio de estado, excluyendo al estado inicial del bloc.

```
BlocListener<BlocA, BlocAState>(
 listener: (context, state) {
  // aquí se realiza alguna acción
 },
 child: Widget(),
)
```

Usamos al listener generalmente para ejecutar funciones tales como mostrar un dialogo, o navegar a otra ventana. En la aplicación desarrollada encontramos esta estructura, para estar a la escucha de los cambios del estado del "Internet Cubit", mismo que validad la conexión a internet; si existe algún cambio de este, el listener ejecutara una acción de notificación al usuario mediante un snackbar de información.

BlocConsumer

Este widget facilita el uso de BlocBuilder o BlocListener, ya que de hecho es una combinación de los dos, tiene una estructura muy similar a los anterior, salvo que este tiene las dos funciones integradas en el `builder` y `listener`; hace que el código sea más entendible por lo que mejora el mantenimiento de este.

```
BlocConsumer<BlocA, BlocAState>(
 listener: (context, state) {
  // aquí se realiza alguna acción
 },
 builder: (context, state) {
  // retorna un widget específico
 }
)
```

BLoCs y Providers de nuestra aplicación

A lo largo de toda la vida de la aplicación, es decir, desde que el usuario la ejecuta hasta que la cierra, la lógica de negocio de la aplicación estará manejada por los manejadores de estado siendo Bloc nuestro manejador de estado primordial y también de Provider que será destinado a ciertas funcionalidades específicas como es el caso del módulo de chat.

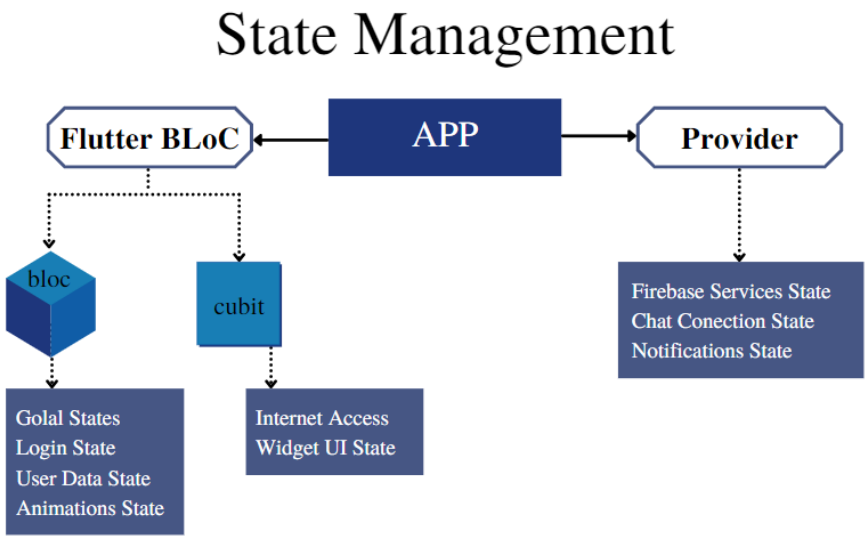

Ilustración 17. - State Management de la aplicación

Como se observa en la ilustración previa, se define el rol que cumplirá cada uno de los tipos de manejadores de estado que están presentes en la aplicación, siendo BloC el más representativo ya que con los distintos blocs y cubit manejamos gran parte de la lógica de negocios de la aplicación, mediante el uso de repositorios de datos tanto externos, es decir; a través de servicios REST y repositorios de datos internos que están almacenados en el dispositivo móvil; a su vez hay bloc que también manejan estados para cada pantalla que a más de mantener el estado con los datos e información necesario, se encarga de manejar los estados de ciertos widget para la ejecución de animaciones que se han considerado de carácter complejo, ya que tienen relación directa con la existencia de datos necesarios los cuáles han sido previamente validados por el mismo bloc; un claro ejemplo de esto es el BuilAppBLoC del cuál se hablará en la siguiente sección. Por otro lado, para el manejo de estados de widgets de UI se ha hecho uso de Cubits, los cuales al ser más simple son ideales para manejar el estado de los mismo, así mismo por su sencillez de configuración y lectura, se ha configurado un cubit para manejar el estado de la aplicación con respecto a la conexión de internet denominado “InternetCubit”, el cual está presente a lo largo de la aplicación para indicar el estado de conexión y según el mismo realizar una u otra acción permitida.

Por otro lado, tenemos a los manejadores de estado bajo Provider, los cuales están encargados de manejar el estado en relación con la comunicación con Firebase, siendo su principal funcionalidad la del chat, pendiente de los cambios que se puedan ejecutar en los mismos, estados de envío y recepción de los datos.

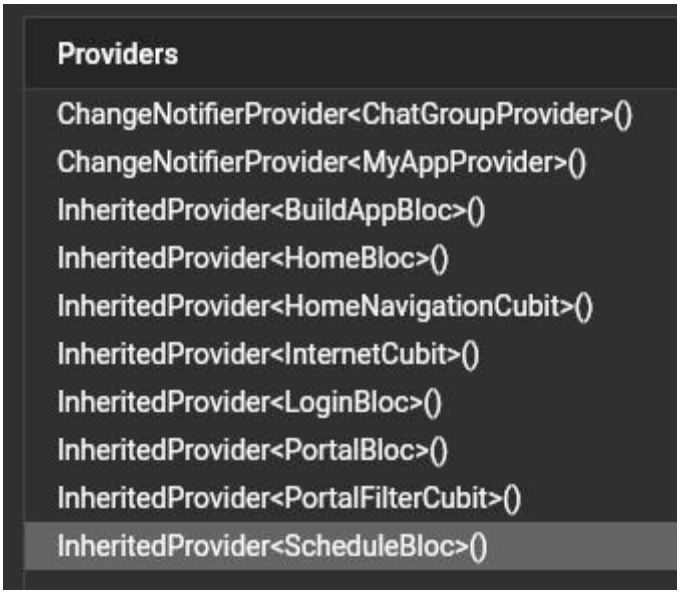

Ilustración 18.- Listado de manejadores de estado de la aplicación

4.3.3. BuildApp BLoC.

El BuilAppBLoC es el manejador de estado de mayor importancia dentro la aplicación, ya que administra la información prioritaria establecida en sus estados, además de que se encarga de validaciones primordiales para el acceso a la misma aplicación, de esta forma logra tener dicha información disponible a lo largo de la aplicación, dicho en otras palabras, es la parte principal de la lógica de negocio de la aplicación.

Una vez que la aplicación es iniciada en el dispositivo móvil, debe cargar cierta información que será útil a lo largo de toda la aplicación, tanto datos de sistema como información prioritaria del usuario registrado, para cual pasa por distintas validaciones antes de la que la aplicación esté disponible para el uso, durante todo este proceso se presenta pantallas de carga o loading al usuario.

Como vimos previamente el funcionamiento de un BLoC, está constituido tanto por eventos que serían las llamadas a las acciones a ser ejecutadas, como por estados que se establen tras ser completas dichas instrucciones; para este caso del BuildApp BloC, tenemos dos eventos y cuatro eventos, siendo el estado inicial "`BuildAppStateInit`".

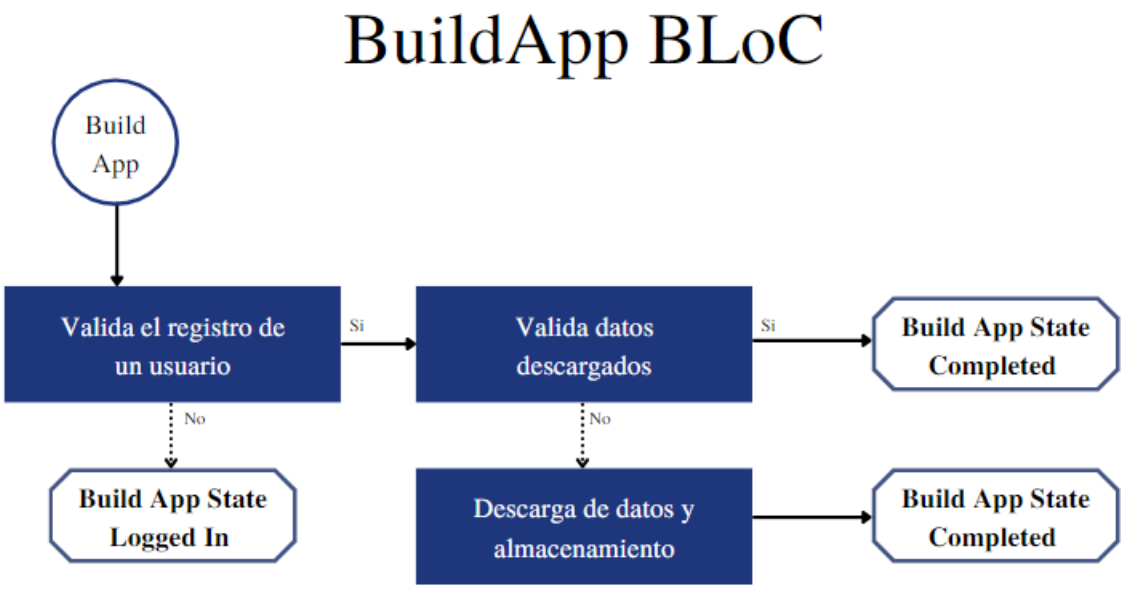

Ilustración 19. - Build App BLoC evento Build App

El primer evento `OnBuild`, corresponde al evento encargado de lanzar la aplicación previo a ciertas validaciones las cuales determinaran el primer paso posterior a la pantalla de carga. Como vemos en la ilustración número 19, la primera acción es validar si un usuario ya ha iniciado sesión en la aplicación, mediante la validación de los tokens almacenados localmente; en el dispositivo tanto con el servidor de pruebas de Node, como el servicio Firebase, por otro lado, también se comprueban la existencia de los datos del usuario, si alguna de esta información no existiera o fue alterada de laguna forma quedando incompleta o ilegible para los métodos establecidos de lectura de datos internos se establece el estado de "`BuildAppStateLoggededIn`" para el bloc, el cual mediante un BlocListener que está a la

escucha de los cambios del mismo hará que la aplicación navegue hasta la pantalla de registro de usuario. En tal caso, una vez que el proceso de registro se haya completado, regresa a la pantalla de carga donde nuevamente pasa por el método de validación del usuario registrado en la aplicación, para evitar cualquier tipo de fallo de información incompleta.

Después de la primera validación, se comprueba que en el dispositivo exista cierta información almacenada en la base de datos interna la cual es consumida por el repositorio de información que nos entrega estos datos, los cuáles son referentes a datos institucionales, mismos que son usados a través de diferentes procesos de la aplicación, si esta información existe y está completa e integra en cuanto a su formato se establecerá para este bloc el estado de "`BuildAppStateCompleted`", el mismo que contiene toda esta información recuperada del repositorio y estará disponible para toda la aplicación, además de que al ser estableció invoca la navegación a la pantalla de "Home" mediante la escucha del BlocListener previamente mencionado.

Por último, en el caso de que la información institucional no exista o este incompleta se procederá a descargar la misma, haciendo uso de la misma dependencia para proveer los datos, pero con distinta implementación, puesto que, en vez de leer los datos de la base interna, se hace uso de un servicio REST para consumir el servicio que nos devolverá los datos en cuestión, la cual será almacena en el dispositivo para futuros usos. Finalmente, del mismo modo que el caso anterior se estable el estado de "`BuildAppStateCompleted`", donde el bloc alberga toda esta nueva información descargada.

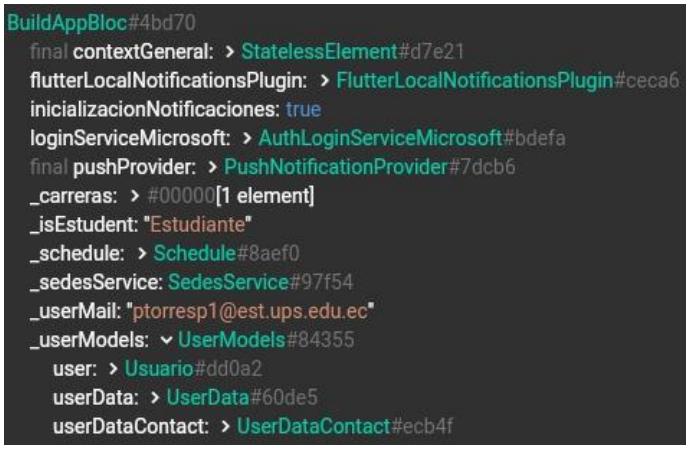

Ilustración 20. BuildApp Bloc Complete State (componentes)

Todos estos procesos de validación, lecturas son de vital importancia para el correcto funcionamiento de la aplicación, ya que es completamente necesario que tenga desde un inicio cierta información tanto de la institución como del usuario registrado, asegurando la posibilidad de que la aplicación funcione de manera Off Line, siendo esta una característica importante de la misma, brindado seguridad y disponibilidad al usuario final.

4.3.4. UI/UX - Tema y diseños.

El tema del diseño de una aplicación va mucho más allá de solamente pensar en colores o estilos de letras, tanto que se ha convertido en parte fundamental de la misma. Puesto que, si un aplicativo móvil no es agradable a la vista de un usuario, difícilmente va a ser que este lo use de manera constante y más aún si a la experiencia de usuario, es decir, la usabilidad de las diferentes acciones y funciones no son intuitivas o por el en contrario son tan complejas que generan un efecto negativo al usuario final.

Como primer paso, se definió un mockup de la aplicación para tener una idea inicial y clara tanto de la interfaz de la aplicación como de la funcionalidad, transición de las distintas pantallas de esta.

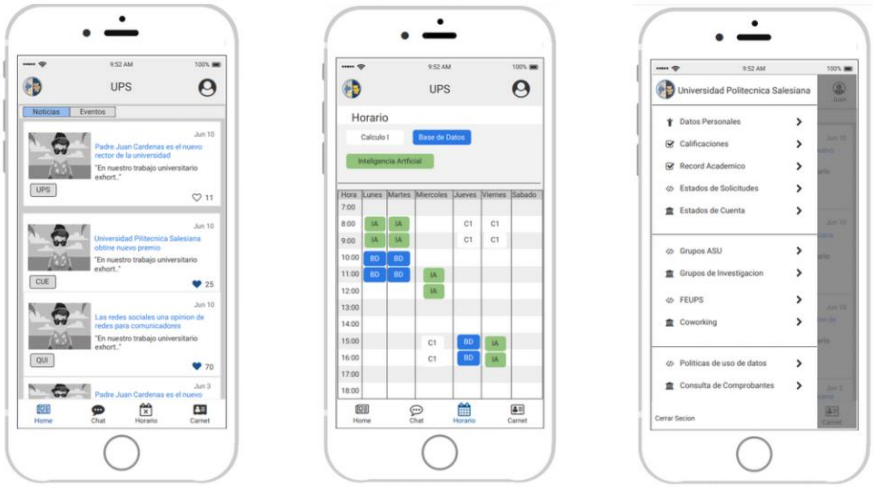

Ilustración 21. – UI - Mockup de la Aplicación

Posterior al prototipo, dejamos en claro la codificación de los corles a ser usado en la aplicación, en este caso en especial ya que nos alineamos a las políticas de imagen de la universidad como tal. Para seguir una misma línea; en cuanto al uso de su imagen en lo que compre al software. Como podemos ver en la ilustración 22 la evolución que tuvo el mockup de la aplicación en contraste con la versión final de la aplicación, en donde ya se han aplicado las diferentes técnicas tanto de UX como de UI, así como también las políticas institucionales de la imagen, siendo los más evidentes el uso de los logos, según la situación corresponda.

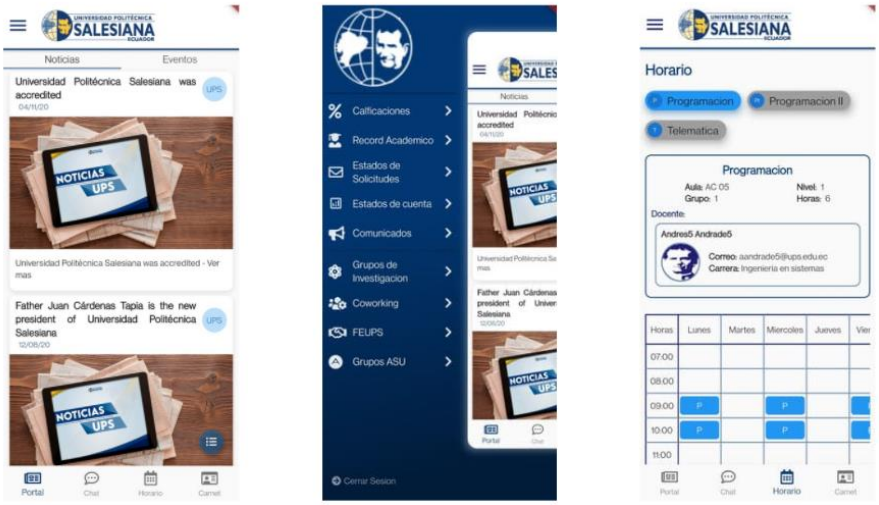

Ilustración 22.- UI - Pantallas finales de la aplicación

Interacción en Flutter

En cuanto a Flutter se reviere, es decir, al código como tal, en la clase main se define el tema global de la aplicación, mediante el widget "`ThemeData`" en el cual se puede especificar los estilos que tomaran por defecto cada uno de los componentes de que puedan estar en la aplicación, tales como cajas de texto, los mismos textos, botones, enfocándose en la propiedad de estos, como tamaños, colores, fuentes etc.

Adicional al widget general para el tema, tenemos dos clases que tiene objetos estáticos que competen a los diferentes códigos de colores que estarán en la aplicación, así como también a los diferentes estilos de textos tales como títulos, subtítulos, definiendo el tamaño, grosor y hasta el color provenientes de los ya definidos previamente; para con todos estos objetos tener de cierta manera centralizados estor atributos de importancia en lo que respecta a la interfaz de usuario.

Por otro lado, para cubrir el tema de las proporciones de cada widget en pantalla, se ha generado una clase denominada "`Responsive`", la cual es de vital importancia, en razón a que hace que todos nuestros componentes se adapten a la pantalla según las dimensiones del

dispositivo. Esta clase recibe como atributo el context, que como ya vimos tiene toda la información tanto del árbol de widgets como especificaciones del dispositivo y en este caso, de las dimensiones de este, lo cual nos permite trabajar sobre porcentajes logrando definir el tamaño de nuestros componentes de una manera idónea según el dispositivo. También tenemos la información de la orientación en el que se encuentra el dispositivo, es decir, en Portrait o Landscape, con esto se logra tener diferentes distribuciones de los componentes adaptados a la orientación del dispositivo.

Ilustración 23. - UI – menú bajo Dev Tools de Dart.

En la ilustración previa vemos como una de las ventajas que Flutter con Dart nos ofrece, y es una de las funcionalidades de las Dev Tools, la cual nos permite me en tiempo de ejecución diferentes líneas guía en la pantalla del dispositivo, mismas que nos indican entre tamaños, límites y direcciones del distinto widget presentes en la pantalla. Con ello se puedo realizar un trabajo cada vez más eficiente a la hora de hacer a los componentes responsivos a los distintos tamaños de pantalla, así como también la distribución que emplean en el layout.

4.4. Creación de la base de datos.

4.4.1. Almacenamiento remoto Firebase

Para la realización de este proyecto, se optó por almacenar datos de la aplicación en la nube de Google (Firebase) cuya base de datos en tiempo real es Firestore, se tomó esta decisión ya que Google brinda una serie de ventajas como se explicó en el marco teórico de este documento, estas ventajas nos ayudaran en lo posterior al desarrollo de la aplicación así también como él envió de las notificaciones que es uno de las principales funciones para el proyecto y chat que es uno de los módulos presentes en la aplicación.

Como primer paso para la usar Firestore creamos un proyecto nuevo en Firebase, al momento de crear el proyecto e ingresar a la plataforma Firebase nos muestra una pantalla como muestra la figura 16

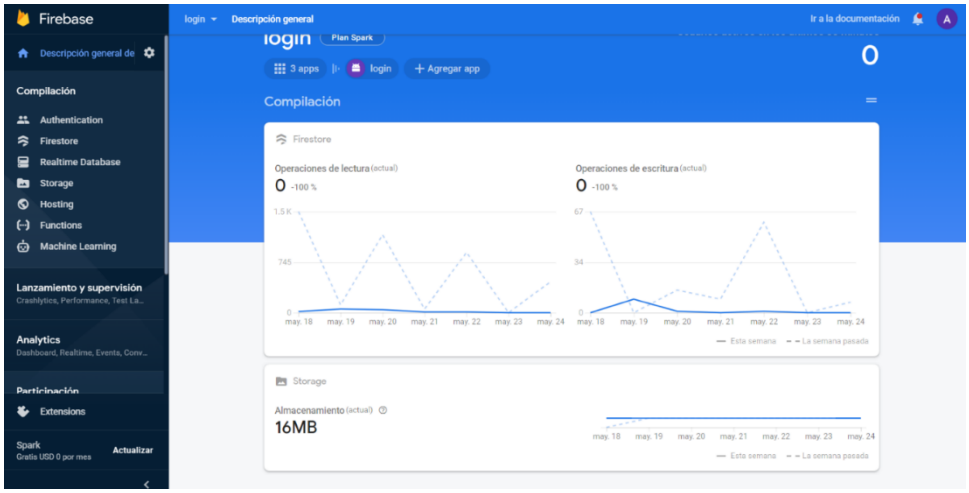

Ilustración 24. - FB - panel de configuración principal.

En la misma pantalla nos dirigimos al apartado de general aquí bajamos completamente hacia el final y nos encontramos con las aplicaciones, aquí debemos crear una aplicación para Android, luego de creada la aplicación nos permitirá descargarnos el archivo google-services.json el cual es un documento en formato JSON que nos da un resumen de todos los

datos necesarios para crear una aplicación móvil que ocupe los servicios de Firebase en este caso como ya se comentó anteriormente los servicios usados son Firestore como base de datos y FCM para las notificaciones

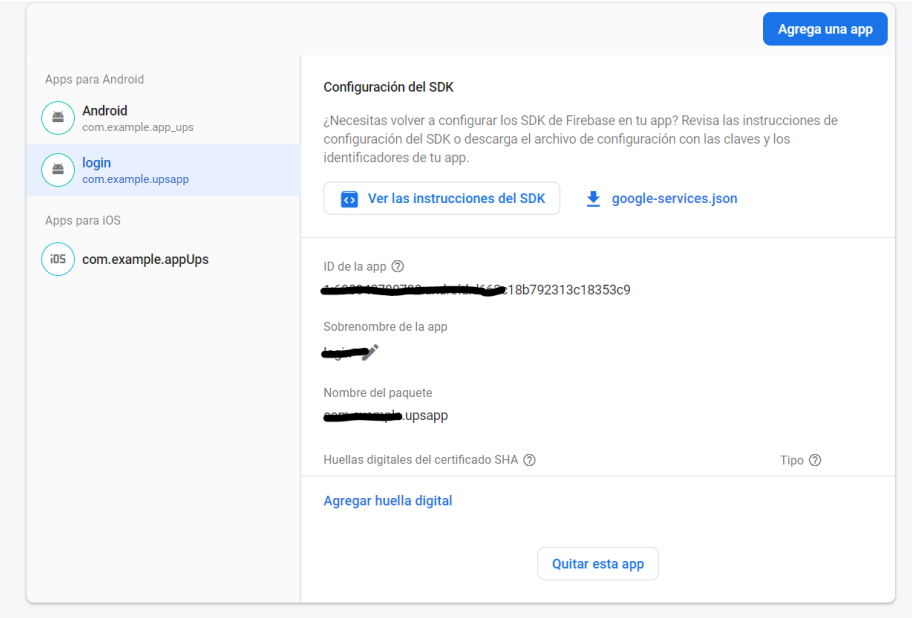

Ilustración 25. - FB - creación de la aplicación

Una vez creado el proyecto de Android para las notificaciones y Firestore, podemos comenzar a crear distintas colecciones de datos, para lo cual nos dirigimos al apartado Firestore dentro del menú lateral izquierdo

Una vez en esta pestaña debemos configurar las reglas que va a tener la base de datos, como por ejemplo las acciones que van a poder realizar los clientes tales como lectura y escritura de datos.

Ilustración 26. - FB - Reglas Acceso

Cabe recalcar que para desarrollo y pruebas de nuestra aplicación se dejaron estas reglas en modo público, en lo posterior cuando ya se pase a producción estas se cambiaran y se realizaran algunos cambios para preservar la seguridad de acceso tanto de escritura como lectura de los datos, a continuación, en la siguiente imagen podremos visualizar como va cambiando las reglas implementadas.

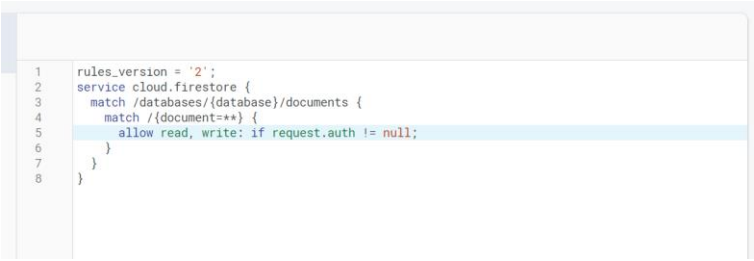

Ilustración 27 Regla segura para la aplicación

Además, tenemos un apartado importante que nos indicara el uso que le hemos dado a Firestore lecturas, escrituras, eliminaciones en la base de datos.

Este apartado es el más importante que encontramos en este submenú Firestone, ya que nos muestra con exactitud las operaciones realizadas a la base de datos, cabe recalcar que Firebase luego de un determinado número de operaciones en la base de datos cobra por cada operación extra realizada al mes.

A continuación, se presenta una imagen en la cual podremos visualizar la manera que tiene Firestore para realizar la facturación.

EE.UU. (multirregión) | Los Ángeles | Salt Lake City | Virginia del Norte | Las Vegas | Montreal | Más

EE.UU. (multirregión)

	Precios posteriores a la cuota gratuita
Lecturas de documentos	$0.06 por 100,000 documentos
Escrituras de documentos	$0.18 por 100,000 documentos
Eliminaciones de documentos	$0.02 por 100,000 documentos
Datos almacenados	$0.18 por GiB al mes

Ilustración 28. - FB - cobros

En la ilustración 18, podemos visualizar la cantidad de operaciones que se ha realizado en la base de datos.

Ilustración 29 . - FB - operaciones

Estas mediciones están realizadas en un ambiente de desarrollo por lo tanto para saber la cantidad de operaciones exactas de lectura y escritura se tendrá que realizar una aproximación para tener datos más reales.

La aplicación se va a lanzar a las 3 sedes de la Universidad Politécnica Salesiana lo que nos daría una cantidad de usuarios de aproximadamente 25 000 descargas, según datos estadísticos la cantidad de usuarios activos por día representa el 10% de las descargas totales de la aplicación dándonos un total de 2 500 usuarios activos por día, además de esto debemos también tomar en cuenta el tamaño de los documentos que es relativamente pequeño, a continuación en la siguiente tabla se podrá visualizar el tamaño en disco que ocupan los archivos.

Colección	Tamaño del documento (en tránsito)	Tamaño del documento (en disco)*
usuarios	1 KB	3 KB
grupos	0.5 KB	1.5 KB
mensajes	0.25 KB	0.75 KB

Ilustración 30 Tamaño de los archivos en disco

En la siguiente imagen podemos visualizar un costo estimado para una media de 50 000 descargas de la aplicación y sus respectivos cálculos.

Por 50,000 instalaciones de la app (5,000 usuarios activos por día): $ 12.14 por mes					
Costos de lecturas y escrituras					
Costo total mensual = $ 11.10	Total de 400,000 lecturas diarias	=	50,000 lecturas GRATIS + (350,000 lecturas a $ 0.06 por 100,000)	=	3.5 * $ 0.06
	$ 0.21 por día * 30 = $ 6.30				
	Total de 100,000 escrituras diarias	=	20,000 escrituras GRATIS + (80,000 escrituras a $ 0.18 por 100,000)	=	0.8 * $ 0.18
	$ 0.14 por día * 30 = $ 4.20				
	Total de 100,000 eliminaciones diarias	=	20,000 eliminaciones GRATIS + (80,000 eliminaciones a $ 0.02 por 100,000)	=	0.8 * $ 0.02
	$ 0.02 por día * 30 = $ 0.60				
Costos de almacenamiento y red					
Costo total mensual = $ 1.04 por mes	20 KB por DAU de salida diaria * 5,000 DAU	=	100 MB de salida diaria * 30	=	3 GB de salida de red mensual
	3 GB de salida GRATIS = GRATIS[1]				
	15 KB de almacenamiento diario de mensajes por DAU + 3 KB de almacenamiento por instalación[2]	=	45 KB de almacenamiento por DAU * 5,000 DAU	=	225 MB de almacenamiento diario por DAU * 30 = 6.75 GB de uso de almacenamiento mensual
	1 GB de almacenamiento GRATIS + (5.75 * $ 0.18) = $ 1.04 por mes				

Ilustración 31 Costo para un aproximado de 50 000 descargas

A continuación, se explicará cada una de las tablas o colecciones de datos creados y usados por la aplicación.

Una vez explicados las opciones que nos encontramos en el submenú Firestore, a continuación, veremos la estructura de datos creados para el módulo de chat y de envío de notificaciones de la aplicación. En la imagen 23, se puede identificar todas las colecciones de datos que usamos en la aplicación.

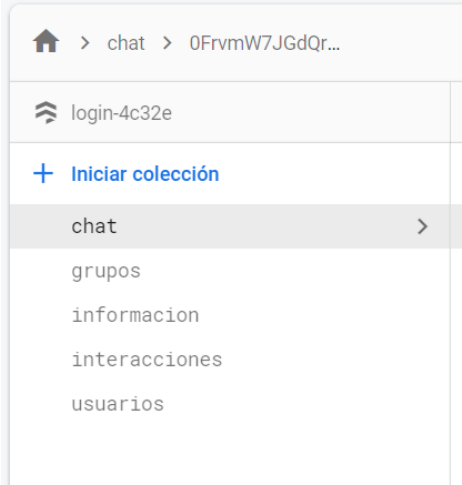

Ilustración 32. - FB - esquema base de datos

A continuación, explicaremos que tiene cada colección y su función dentro del módulo de chat

Chat

En esta colección almacenaremos los mensajes enviados ya sea de manera grupal o manera individual para mostrarlos en tiempo real dentro de la pantalla del chat en la aplicación.

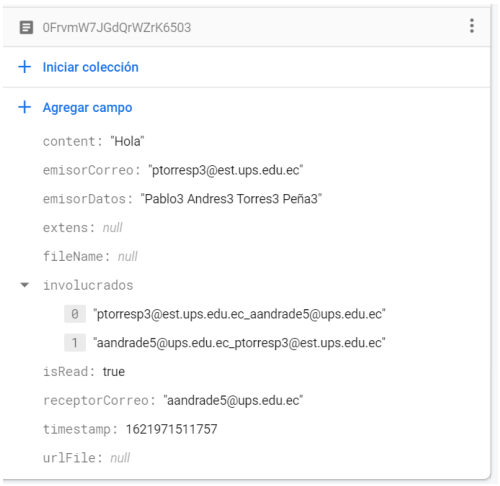

Ilustración 33. - FB - colección Chat

Grupos

En esta colección de datos almacenamos la información que corresponde a los grupos creados para el chat entre profesores y estudiantes.

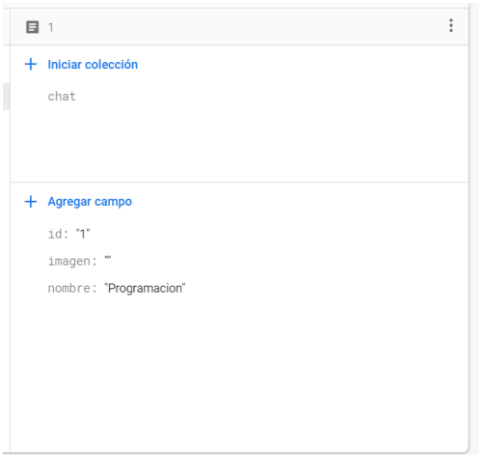

Ilustración 34. - FB - colección Grupos

Información

En esta colección de Firestore guardamos los datos correspondientes al módulo de comunicados que tiene nuestra aplicación, dichos comunicados son enviados por los directores

de carrera además llega como una notificación push contando también con una pantalla que muestra los diferentes comunicados que ha recibido el estudiante.

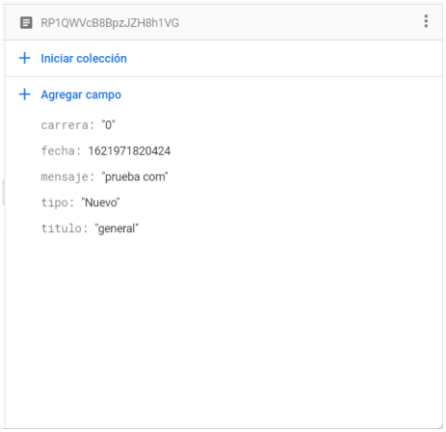

Ilustración 35. - FB - colección Información

Interacciones

En esta colección se almacena los datos de todas las interacciones que se realizan en el chat, aquí podemos ver quien ha enviado el mensaje además a quienes se los envió.

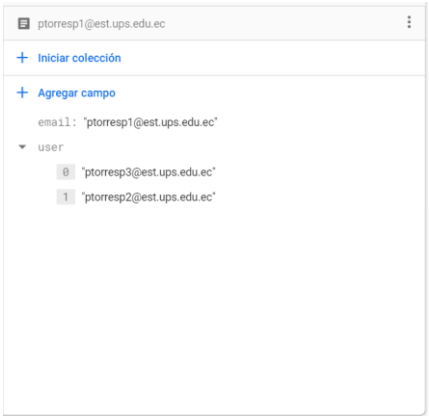

Ilustración 36. - FB - colección Interacciones

Usuarios

En esta colección; como su nombre lo indica guardamos los usuarios que tienen descargada y han iniciado sesión dentro de la aplicación, aquí se mantiene datos de los usuarios como nombres, carrera, campus, entre otros campus concernientes a la Universidad Politécnica Salesiana además se tiene el token de usuario dado por FCM para él envió individual de notificaciones push.

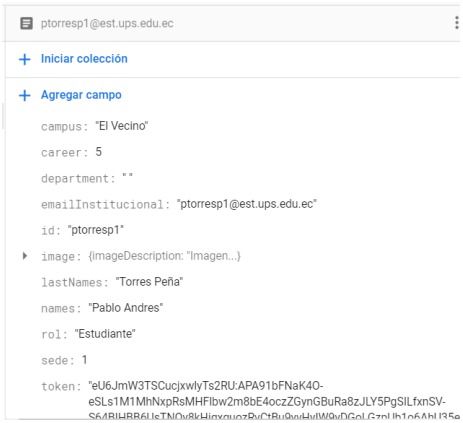

Ilustración 37. - FB - colección Usuarios

4.4.2. Almacenamiento local.

Al ser una de las funcionalidades de la aplicación, el hecho de que pueda ser usada cuando el dispositivo no disponga de internet, es decir, de manera Off Line, es correcto pensar ciertos datos deberán ser almacenados de manera local, para lo cual se ha analizado y escogido dos maneras de guardar dichos datos.

Flutter Secure Storage .

Esta opción esta destina a datos que cortos y de rápido acceso, que serán almacenados en el dispositivo a manera “Shared Preferences”, tal y como es el caso de información

concerniente al inicio de sesión de la aplicación referente a datos del usuario. Para esto se ha usado el paquete externo “Flutter Secure Storage”, ya que este a diferencia de las Shared Preferences tradicionales, no guarda la información en texto plano o legible de algún modo, sino que agrega cierta seguridad y encriptación de los datos, debido a que serán datos cruciales sobre el registro del usuario, mismo que luego servirán a la lógica de negocio de la aplicación pueda realizar validaciones y usos pertinentes. Toda esta información será eliminada permanentemente al cierre de sesión.

SQFlite

Por otro lado, se ha implementado la base de datos SQLITE, la cual es una base relacional, en donde estará almacenada tanto la información institucional como información del usuario. En lo que compete a la información institucional, tenemos a los datos de las sedes, campus y carreras, misma brinda la estructura organizativa de la universidad. Sobre la información de los usuarios se estará almacenando datos referentes a datos personales y el horario, siendo este último tipo de dato de gran importancia; ya de esta información depende la creación de los grupos de chat a los cuales un usuario estará agregado automáticamente por el sistema.

4.5. Análisis de Performance.

Tanto el desarrollo como los diferentes análisis se hicieron en dispositivos Android, unos emulados y otros en dispositivos físicos. Las capturas siguientes con respecto a los datos de análisis corresponden a la ejecución en un dispositivo OnePlus 8, el cual cuenta con Android 11, además que se tomaron en modo Debug, lo que significa en Flutter que este modo tiene

mayor carga de procesamiento que en un dispositivo final al ser descargado en cualquier tienda de aplicaciones.


```
{
  "engineEnterTimestampMicros": 440150025951,
  "timeToFrameworkInitMicros": 140609,
  "timeToFirstFrameRasterizedMicros": 194518,
  "timeToFirstFrameMicros": 187725,
  "timeAfterFrameworkInitMicros": 47116
}
```


Ilustración 38. - Proceso de levantar la aplicación

Como apreciamos los datos de la ilustración previa, notamos que el tiempo que trascurre para que la aplicación este a disposición de uso, es realmente bajo, y esto se debe principalmente a que Flutter compila el su código a código nativo y lo hace de una forma realmente eficiente.

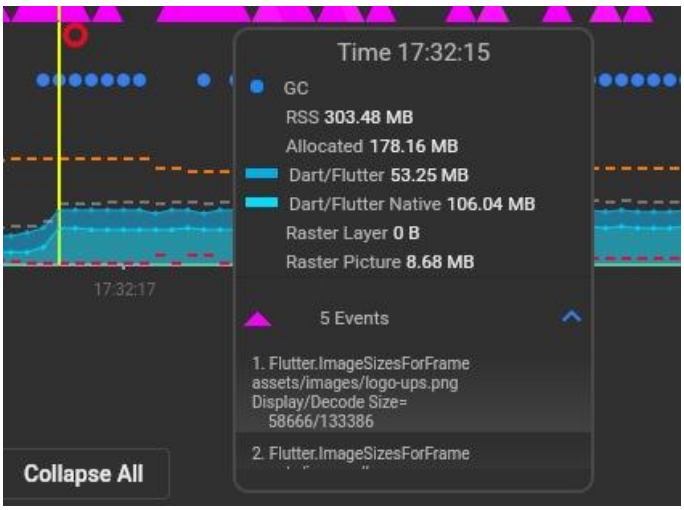

Ilustración 39. - Métrica de rendimiento de la aplicación por Dev Tools

Así mismo las Dev Tools, como herramientas de desarrollo del mismo lenguaje Dart, nos provee mucha información con respecto a la aplicación en ejecución. En la ilustración 36,

notamos un uso de 106.04mb del dispositivo, en uno de los picos más altos del time-lapse de rendimiento. Lo cual nos da una idea clara del rendimiento de la aplicación ya ene ejecución.

CAPÍTULO 5. MÓDULOS DE LA APLICACIÓN

5.1. Información institucional.

Al ser la aplicación de carácter institucional destinada para la Universidad, tendrá su apartado fundamental que es el de exponer al usuario la información organizacional, por tal razón se ha seleccionado datos prioritarios sobre distintos departamentos y asociaciones que más interacción tienen con la comunidad universitaria.

Por ende, la información que estará a disposición será sobre: la Secretaría de Ecosistema de Emprendimiento e Innovación haciendo énfasis en los Grupos de Investigación y en Coworking StartUPS; por otro lado, se expone información sobre las distintas agrupaciones estudiantiles, como son FEUPS y Grupos ASU.

Ya que la información ha sido recopilada de distintas fuentes oficiales de la universidad tales como: la página web de esta, página web de Investigación UPS y la página web de Coworking StartUPS; toda esta información se encuentra de manera estática en la aplicación debido que por el momento no se cuenta con métodos que nos provean de dichos datos, como sería el caso de un servicio web.

Cada apartado de información está expuesto en una pantalla distinta, por lo que se la ha realizado bajo una misma línea gráfica para de esta forma generar una UX dentro de este apartado. Así como también el tema de contralar la distribución de los distintos componentes gráficos, para lo cual nos ayudamos de las Dev Tools de Dart, asegurando simetrías entre las pantallas y sobre todo los componentes sean responsivos ante distintas dimensiones.

Ilustración 40. - UI - información institucional creación de vista

5.2. Portal Noticias y Eventos.

Este apartado también hace referencia a información institucional; en este caso se hace énfasis al Portal Institucional, es decir, las noticias y eventos que se publican en la página web de la universidad. Esta información, se presenta como pantalla inicial de la aplicación una vez registrado un usuario en la misma.

La idea es tener diferenciado las noticias de eventos en componentes gráficos independientes, pero los cuales siempre mantienen las mismas características de UI para mostrar estos datos, mismos que se presentan para ambos casos en manera de listado, generando la posibilidad también para acceder a cada elemento ya sea noticia o evento para navegar a una pantalla independiente donde se mostrara el detalle completo del mismo.

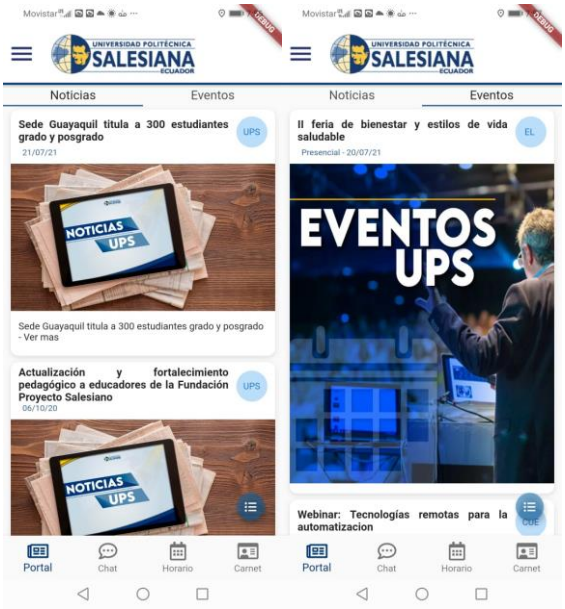

Ilustración 41. - UI – tabs bar Noticias y Eventos

Ilustración 42. - UI – ventanas independientes para detalle de una noticia o evento.

5.2.1. Portal Bloc

Para manejar la lógica de negocio de este apartado se ha generado un BLoC el cual se lo ha denominado “PortalBloc”, donde su característica principal es la de obtener los datos

correspondientes a noticias y eventos para establecer dicha información en estado del bloc y exponerlo al uso de los widgets para la muestra al usuario.

Puesto que la información tanto de las noticias y eventos, están destinas a trabajarse dentro de la aplicación de una manera muy similar, se ha optado por generar un solo bloc para administrar la misma peor con la peculiaridad de que existen dos incastas del PortalBloc, una para eventos y otra para las noticias.

Para cada instancia del bloc, lo que se realiza al ser llamado sus eventos, es el de obtener los datos referentes a los elementos deseados, esto es mediante un consumo de servicios web, el cual nos devuelve los datos de tal manera que la podemos trabajar de manera eficiente ya que nos brinda además la posibilidad de aplicar filtros a la información, siendo esto último clave para dar la funcionalidad de presentar al usuario las noticias o eventos según una sede especifica.

5.2.2. Substream InternetCubit.

Ya que la ventada del Portal en la aplicación actúa como ventana inicial o principal, se le ha dado mayor funcionalidad a la misma, como es el caso de estar pendientes a los cambios del InternetCubit, el cual nos avisa si sobre una desconexión o reconexión a internet del dispositivo móvil. Lo cual se logra mediante un Substream, que es una manera de subscribirse a un Stream, en este caso al stream del InternetCubit, es decir, el PortalBloc se subscribe al InternetCubit, estando siempre a la escucha de su stream para identificar si este ha cambiado su estado.

Con esta funcionalidad agregada, podemos a través de validaciones controlar el estado del PortalBloc ya que en una reconexión a internet automáticamente se volverá a consumir los

servicios que nos provén los datos ya sea de noticias o eventos, para dar la impresión de actualización.

5.3. Carnet digital.

Este módulo provee de una alternativa al carnet físico tradicional, el cual es el único identificador oficial de un miembro de la universidad. La idea es plasmar la misma información del carnet físico y recrearlo en una pantalla especifica añadiendo un código QR para que pueda ser leído y utilizado como objeto de validación en diferentes actividades dentro de la universidad.

Ilustración 43. - UI - pantalla Carnet digital

5.4. Información institucional académica de usuario.

Para todo este apartado de información académica de un estudiante, se ha destinado que se muestre en pantallas diferentes una para cada tema, por ende, cada uno de estas tiene un

BloC específico para el manejo de la lógica de negocio, manejando el estado de la información que estará a disposición por el bloc.

Todos lo blocs de esta sección, guardan mucha similitud, sin embargo, se mantienen independientes uno del otro para de esta forma garantizar una mejor escalabilidad a futuro y también un mejor mantenimiento del código. La manera de obtener la información es a través del Servidor UPS, para clo cual en las peticiones REST que se efectúan en este apartado, incluyen configuración adicional de autenticación para el uso de los servicios.

Hay que recordar que, para hacer uso del Servidor UPS, deberemos estar en el entorno de pruebas de este para lo cual se hizo uso adicionalmente de la VPN de Cisco proporcionada por el departamento de TIC de la universidad.

5.4.1. Récord académico

En este módulo se muestra el récord académico de un estudiante, dando a conocer las materias y su calificación respectiva de las materias que hayan sido aprobadas por el mismo. Además de gráfico, el cual indica una comparativa entre las materias ya aprobadas con las que le falta por cursar para culminar su carrea correspondiente.

Ilustración 44. - UI - récord académico.

5.4.2. Estado de solicitudes

En este módulo se mostrara en forma de listado las solicitudes que el estudiante ha realizado por medio de la portal institucional además mostrara el estado de las mismas, al dar clic sobre una de las solicitudes mostrara tanto el estado actual como las reasignaciones que ha tenido dicha solicitud a los colaboradores de la Universidad Politécnica Salesiana mostrando así su detalle, en dicho detalle se verá información como fecha de reasignación, responsable, numero de solicitud y alguna observación que se ha dado a dicha solicitud.

Ilustración 45. - UI – estados de solicitudes

5.4.3. Estados de cuenta

Esta sección muestra detalles del estado de cuenta del estudiante, teniendo en cuenta que existen tres tipos de información que podemos encontrar en este apartado los cuales son: Facturas realizadas, Deudas pendientes y Pre-facturas. Brindando al usuario tener toda esta información de rápido accesos, y presentados los tres tipos de elementos bajo una misma línea gráfica y estructura de los widgets.

Ilustración 46. - UI - estados de cuenta

5.5. Comunicados.

Uno de los servicios primordiales de la aplicación, es este apartado en concreto, el de los Comunicados. El cual como su nombre indica, sirve para realizar notificaciones a los usuarios de la aplicación ciertos mensajes globales, manteniendo una comunicación efectiva.

Para el manejo del estado de los widgets y la lógica del negocio de este apartado, se lo ha realizado con Provider, puesto que se hace uso de los servicios de Firebase, tales como es el de Cloud Firestore y Fire Cloud Messaging.

Ya que se ha destinado a Provider para todo lo relacionado con Firebase, aprovechamos la ventaja ya la facilidad del ChangeNotifierProvider, lo que nos permite emitir cambios al estado al enviar o recibir datos desde los servicios de Firebase, y también permite estar siempre a la escucha de los cambios que se puedan dar en los datos almacenados en Firestore.

Los comunicados se los ha clasificado en “Generales” y de “Carrera”, los cuales están destinado a que usuarios pueden recibir los mismos. En el primer caso son comunicados globales, es decir, todos los usuarios registrados en la aplicación pueden recibir dicha información, con la peculiaridad de que estos pueden ser de diferente tipo; entiéndase por tipo a diferentes departamentos o agrupaciones que podrían emitir comunicados, ver ejemplos en la siguiente ilustración.

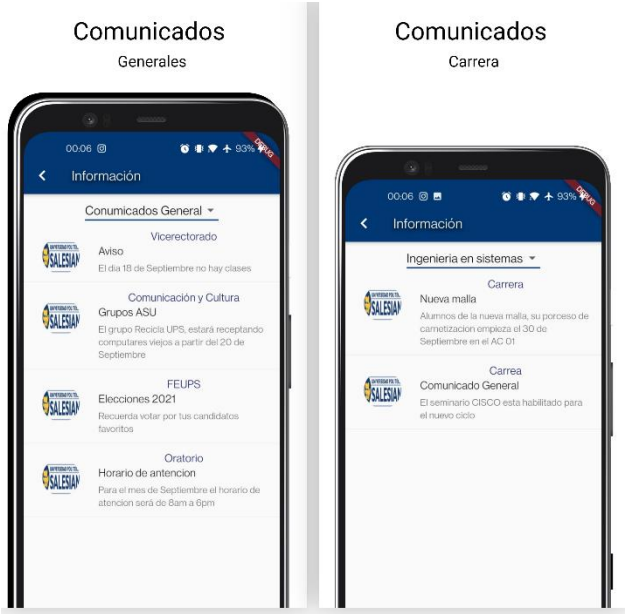

Ilustración 47. - UI - comunicados

Para poder enviar los comunicados desde software satélites se desarrolló un servicio en java con tecnología JEE, dicho servicio cuenta con el siguiente método:

- Enviar comunicado
    - Parámetros de entrada
        - Topic: Código de la carrera
        - Correo: Correo del estudiante para el envío de la notificación
        - Título: Titulo de la notificación
        - Mensaje: Cuerpo de la notificación

```java
@GET
@Path("push")
public Response enviarComunicado(String topic, String correo, String titulo, String mensaje) throws IOException,

    if (topic == "%") {
       ReglasNegocio r =new ReglasNegocio();
       r.enviarPushUsuario(titulo, mensaje, correo);
       return Response
           .ok("ok")
           .build();
    }else
    {
       ReglasNegocio r =new ReglasNegocio();
       r.enviarPushTopic(titulo, mensaje, topic);
       return Response
           .ok("ok")
           .build();
    }
}
```

Ilustración 48.- Método envió Comunicado

Para poder hacer la conexión hacia Firebase tanto para enviar notificaciones como al Firestore desde pom.xml se importó las siguientes librerías

```xml
        <dependency>
        <groupId>com.google.truth</groupId>
        <artifactId>truth</artifactId>
        <version>1.0.1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.google.cloud</groupId>
        <artifactId>google-cloud-firestore</artifactId>
        <version>1.35.0</version>
    </dependency>
    <dependency>
        <groupId>com.google.firebase</groupId>
        <artifactId>firebase-admin</artifactId>
        <version>6.14.0</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.6</version>
    </dependency>
```

Ilustración 49.- Archivo pon.xml

Una vez importados los paquetes necesarios para la conexión con Firebase se procedió a realizar el desarrollo para el envío de notificaciones y almacenar datos en Firestore. En la siguiente imagen se muestra el método usado para la conexión con Firebase

```java
public Conexion(String nombreProyecto) throws IOException{
    path = "ups.json";
    InputStream service_account = this.getClass().getResourceAsStream(path);
   // System.out.println("Archivo:  " + this.getClass().getResource(path));
    ruta = System.getProperty("java.class.path");
    GoogleCredentials credentials = GoogleCredentials.fromStream(service_account);
    FirebaseOptions options = new FirebaseOptions.Builder()
        .setCredentials(credentials)
        .build();
    app = FirebaseApp.initializeApp(options);
    this.db = FirestoreClient.getFirestore(app);
}
```

Ilustración 50.- Método para la conexión hacia Firebase

5.6. Chat institucional.

Este módulo a su vez consta de dos módulos internos chat individual y chat grupal, el chat grupal servirá para que los docentes de las diferentes materias puedan enviar comunicados a los estudiantes que se encuentran cursando dichas materias, este chat se creara automáticamente al momento de que inicie sesión ya sea un estudiante o un profesor perteneciente a la universidad, el momento que exista el registro exitoso de un usuario, se obtendrá mediante un servicio web el horario del estudiante el cual nos servirá para la creación de salas de chat para el estudiante basado en el nombre del grupo Enel que se encuentre matriculado, este módulo está ligado a Firebase ya que usamos sus dos principales características que son almacenamiento en tiempo real y las notificaciones push que le alertaran al estudiante cunado un mensaje le llegue, otra funcionalidad importante que se usó de Firebase es el almacenamiento de archivos para el envío y recepción de los mismo por medio del chat.

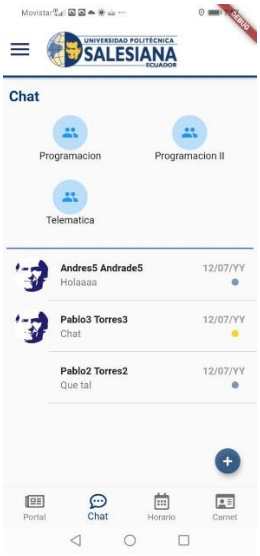

Ilustración 51. - UI - pantalla chat institucional

5.6.1. Grupos de chat

Como se lo explico anteriormente este módulo fue creado especialmente para que los estudiantes de las diferentes materias se puedan comunicar entre si e inclusive con el docente encargado de la materia.

Al momento de iniciar sesión se consume un servicio rest expuesto por la universidad que nos devuelve el horario del estudiante o el docente, se usa dicho horario para crear los diferentes asignaturas y grupos para las salas de chat del estudiante.

Como se puede ver en la imagen anterior en una misma pantalla se tiene tanto las salas de chat grupal basadas en el horario además se tiene los chats individuales creados e iniciados por el estudiante o el docente con los demás estudiantes y docentes, cabe recalcar que se podrá iniciar un chat individual siempre y cuando la otra persona tenga descargada la aplicación y haya iniciado sesión.

Ilustración 52. - UI – grupos chat

5.6.2. Carga de imágenes y archivos multimedia.

Una característica fundamental con la que cuenta el módulo de chat es la posibilidad de enviar archivos de imágenes, pdf, Word ya que de esta forma los docentes podrán enviar directamente los deberes o trabajos tanto al chat individual como grupal para una mejor

comunicación entre docentes y estudiantes además permitiendo que trabajos o deberes les puedan a llegar a todos los estudiantes que tengan la aplicación descargada.

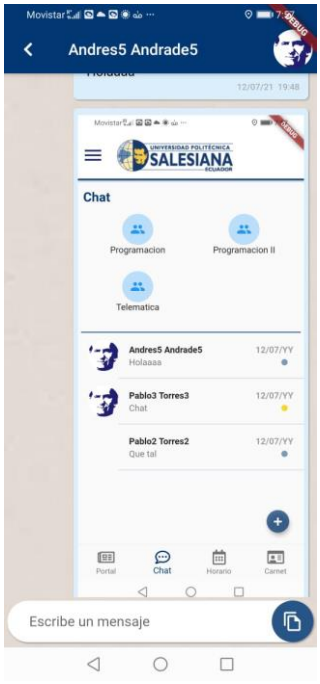

Ilustración 53.- UI - carga archivos multimedia

5.7. Análisis de resultados de la aplicación.

- Se ha logrado un nuevo canal de comunicación para los miembros de la universidad, de manera eficiente ya que un usuario tendrá de automáticamente grupos de chat que corresponden a sus grupos del periodo académico vigente.
- Dentro del mismo apartado de chat se logra un envió de archivos multimedia, imagen, pdf de manera eficaz, sin embargo, hay que tener en cuenta el posible sobre almacenamiento de archivos.
- Como otro canal de comunicación, es el de los Comunicados, los cuales se pueden realizar de manera general o por una sola carrera, llegando a todos los miembros de estas cuenten con la aplicación, de esta forma se logra informar a los usuarios de una mejor manera.

- Un usuario estudiante podrá acceder de manera más rápida a su infracción académica, la cual está a su disposición a no más de 3 interacciones (clics o touch) con la pantalla del dispositivo.
- El Carnet digital, brinda una forma rápida de identificación, mismo que abre la puerta a muchos usos posteriores dentro y fuera de la institución.
- Registro de la aplicación por medio de; servidor Azure para las funciones de Office 365 que tiene la Universidad, generando un mismo ecosistema de interacción para el usuario final.

En resumen, se cumple con el objetivo general de este proyecto de una aplicación informativa y de apoyo a los miembros de la institución, por medio de la creación de nuevos canales de comunicación indispensables en el marco académico.

Se ha realizado una investigación sobre el lenguaje Dart y Flutter, tanto de la documentación oficiales de las mismas como de diferentes autores, lo cual nos ha permitido plasmar los conceptos dentro del marco teórico, pero aún más importante la interacción de la teoría con la práctica, en cada parte de la construcción de la aplicación, así como también para definir de una manera idónea los requisitos del proyecto; con lo que se da cumplimiento al primer objetivo específico.

Para el segundo objetivo específico, La interacción con distintos departamentos de la universidad nos ha dado las directrices para el diseño de la interfaz de la aplicación, la cual está acorde a los diferentes parámetros institucionales. Además de ser una aplicación amigable e intuitiva para el usuario final, ya que se ha sigo convenciones de UI/UX.

En relación con el objetivo tres, se estructuro el proyecto de la aplicación de tal forma que se pueda cumplir con cada uno de los requerimientos, la cual fue una estructura basada en la arquitectura de N-capas, lo que aporto gran valor ya que hemos podido usar distintos

servidores para nuestro back end, e incluso uno de pruebas el cual para cierta información al final se lo sigue usando. En cuanto al proyecto de desarrollo de la aplicación como tal, se ha usado principalmente BloC como manejador de estado, así como también sus convenciones para el manejo de directorios, la cual no ayuda a la edición y adicción de nuevas características sin mayor conflicto. Toda esta estructura tiene gran importancia ya que vuelve al proyecto entendible, mantenible y escalable.

Finalizamos con la elaboración de la documentación, tanto manuales de instalación como de usuario. Referirse a los anexos.

- Anexo 2 – Manual de instalación y configuración de despliegue
- Anexo 3 – Manual de usuario.

CONCLUSIONES Y RECOMENDACIONES

- El SDK de Flutter como herramienta para el desarrollo de aplicaciones, nos brinda grandes ventajes en comparación ante sus similares, el desarrollo como tal es más rápido principalmente por el "Hot Reload" lo que agrega una mayor eficacia a un desarrollador, además de que es realmente multiplataforma ya que compila el proyecto a código nativo. Hay que tener que cuenta que está en conste actualización por parte de Google, por lo que hay que estar pendientes a los cambios, sin embargo, existe una gran comunidad la cual facilita el entendimiento y acoplamiento a los mismos.
- Se concluye que el proyecto de la aplicación es escalable a nuevas características principalmente por la arquitectura de N-Capas, lo que nos permite consumir servicios de diversos servidores o fuentes siendo operaciones que el usuario final no lo nota se puede realizar cualquier cambio sin mayor inconveniente. Por otro lado, la aplicación sola también cumple dicha característica de escalabilidad, en este caso es debido al manejador de estado principal, el cual es el Flutter BloC, con el cual se logra tener una correcta administración en cuanto al estado de la UI como al manejo de la lógica de negocio de la aplicación.
- Se concluye que el uso de Firebase como uno de los servidores para la aplicación es muy acertado ya que permite una administración del almacenamiento de los datos y archivos multimedia de manera eficaz, así como también para el control de sesiones y envió de notificaciones push, mismo que son escalables en su complejidad para agregar diferentes funcionalidades más complejas.
- Las bondades de Firebase son altas, y debido a esas mismas ventajas lo convierten en un punto de quiebre a la hora de tomar la decisión de quedarse con este como

servidor, ya que debido a su creciente uso se entrará en una fase de pago la cual significa costos adicionales a la institución, mismos que dependerán a la cantidad de operaciones tanto de lectura como de escritura de los datos. Por este motivo se recomienda, el desarrollo de un servidor propio que cuente con estas características, en donde sea indisponible una base de datos a tiempo real y la funcionalidad de envió de notificaciones push.

REFERENCIAS BIBLIOGRÁFICAS

Angelov, F. (s.f.). *Bloc*. Obtenido de https://github.com/felangel/bloc/blob/master/packages/bloc/README.md

Días., J. L. (2016). Sistema web móvil para la gestión y el control entre usuarios.

Fayzullaev, J. (2018). *Native-like Cross-Platform Mobile Development Multi-OS Engine & Kotlin Native vs Flutter.* South-Eastern Finland: University of Applied Scienses.

flutter-dev. (s.f.). *Flutter*. Obtenido de https://flutter.dev/docs/

García, C. (2017). *Desarrollo de una aplicación Android de apuestas utilizando Firebase para la sincronización de datos.*

Google. (s.f.). *FlutterES*. Obtenido de https://esflutter.dev/docs/development/

Götz, F. M., Stieger, S., & Reips, U. D. (2017). Users of the main smartphone operating systems (iOS, Android) differ only little in personality. *PloS one,* 12.

Guerrero, C., & Bautista, R. (2017). IOT: Una aproximación desde ciudad Inteligente a Universidad, Parra-Valencia.

Introducción a los Servicios Web RESTful Dept. Cienca de la computación. (2017). Obtenido de http://www.jtech.ua.es/j2ee/restringido/cw/sesion11-apuntes.pdf

Lopez, S. (2020). *Funcionalidades de firebase : que es, para que sirve y ventajas.*

Marc Fleury, . S. (2011). JBoss® 4.0 The Official Guide.

Martinez, R. M. (2019). *Creación de una App de Gestión para Android con un Framework Actual,.*

Mestras, J. P. (2014). Aplicaciones Web/Sistemas Web.

Mi UG - Universidad de GuayaquilSmart Campus. (s.f.). Obtenido de http://www.ug.edu.ec/

Oliver, E. (2009). A survey of platforms for mobile networks research. *ACM SIGMOBILE Mobile Computing and Communications Review, ,* 12(4), 56-63.

Universidad de Oviedo, Servicio de Informática y Comunicaciones. (s.f.). Obtenido de https://sic.uniovi.es/informatica/appmovil

Windmill, E., & Rischpater, R. (2020). *Flutter in Action.* Shelter Island, NY, United States of America: M A N N I N G - SHELTER ISLAND.

ANEXO 1 – REQUERIMIENTOS

En esta sección se encuentran detallados cada uno de los requerimientos funcionales establecidos previamente.

Estos requerimientos definen que es lo que hace el sistema como tal, es decir cuáles son las funcionalidades que cumple el mismo para con el usuario. Tras el proceso de levantamiento de los requerimientos antes mencionados se han definido que la aplicación deberá contar con las siguientes características.

1º. Control de sesión de los usuarios por medio de las credenciales universitarias.

Cod:	RF 001
Descripción:	Control de sesión de los usuarios por medio de las credenciales universitarias.
Descripción detallada:	Para acceder a la aplicación es necesario una autenticación, la cual debe ser realizada con las credenciales de la universidad, por medio del dominio de Azure. Para ello se deberá validar que los correos ingresados sean únicamente con estos dominios: @est.ups.edu.ec @ups.edu.ec Se tiene dos métodos: • Server Pruebas • Azure
Precondiciones:	El dispositivo móvil deberá contar con acceso a internet.

Entradas:	Ventada de Login, se ingresa las credenciales.
Proceso:	Como se mencionó existe dos métodos para realizar el registro de un usuario en la aplicación. • Server Pruebas: El usuario ingresara credenciales directamente en la aplicación, para registrarse con el servidor de pruebas el cual nos devuelve un token de acceso, • Azure: Se abre un Web View propio de los servicios de Azure para Office365, la cual nos permite el registro con las credenciales institucionales. Una vez validado regresa a la aplicación con el token.
Salidas:	Si es correcto, pasará a una pantalla de carga que iniciará la aplicación. Si es incorrecto, se presentará un mensaje al usuario indicando el inconveniente.
Postcondiciones:	Si no se tiene acceso a internet o se pierde el mismo en momento de realizar la operación, se presentará un mensaje al usuario de alerta.
Roles involucrados:	Usuario final.
Verificación:	Si

2º. Sistema de mensajería instantánea.

Cod:	RF 002
Descripción:	Sistema de mensajería instantánea.
Descripción detallada:	Envió y recepción de mensajes a los diferentes usuarios en tiempo real.
Precondiciones:	Configurado un servidor en tiempo real para él envió de los datos, de los mensajes. Para esto se ha configurado el servidor de Firebase. Tener acceso a internet para él envió de un mensaje, Para visualizar los mensajes estos no requieren internet ya que se almacenan en la memoria chache.
Entradas:	Panta de chat ya sea conversación de un grupo o individual.
Proceso:	Usuario ingresa un texto deseado o en su defecto selecciona un archivo de su dispositivo.
Salidas:	Si el mensaje se envió exitosamente aparecerá en la pantalla de mensajes de la conversación correspondiente,
Postcondiciones:	Deberá tener acceso a internet para visualizar si el otro usuario recibió y abrió para ver el mensaje en concreto, siendo el chat directo con otro usuario, en conversaciones grupales no se tiene esa confirmación,

Roles involucrados:	Usuario final.
Verificación:	Si

3º. Autogeneración de grupos según materias matriculadas (estudiante) o asignadas en los distributivos (docentes).

Cod:	RF 003
Descripción:	Autogeneración de grupos de chat
Descripción detallada:	Autogeneración de grupos según materias matriculadas (estudiante) o asignadas en los distributivos (docentes). En base a lo detallado, cuando un usuario se registre en la aplicación, esta creara los grupos de chat según las especificaciones, de manera automática.
Precondiciones:	Acceso a internet, registro éxitos de un usuario.
Entradas:	-
Proceso:	Se descarga de los servidores la información pertinente, para con ella generar los grupos de chat, mismo que será subido al servidor Firebase en donde se almacenará.
Salidas:	-
Postcondiciones:	Tener acceso a internet durante todo este proceso, si falla automáticamente se volverá a ejecutar.
Roles involucrados:	Usuario final.

Verificación:	Si

4º. Servicio de notificaciones push.

Cod:	RF 005
Descripción:	Enviar y recibir notificaciones push
Descripción detallada:	Al momento que llega un comunicado o un mensaje se muestre una notificación push en el teléfono móvil en el cual tenemos instalada la aplicación.
Precondiciones:	Conexión a internet, tener instalada la aplicación móvil, haber iniciado sesión en la aplicación móvil.
Entradas:	Tokens a los cuales se les va a enviar la notificación.
Proceso:	Al momento de enviar un mensaje por el chat o un comunicado desde el módulo le llegara a al destinatario una notificación push que le alerte de la recepción del mensaje
Salidas:	Titulo y texto del mensaje en la notificación recibida.
Postcondiciones:	Tener internet en todo el proceso sino no le podrá llegar la notificación push.
Roles involucrados:	Usuario final
Verificación:	Si

5º. Servicio de comunicados.

Cod:	RF 006
Descripción:	Enviar y recibir comunicados.
Descripción detallada:	Tener la posibilidad de enviar y recibir comunicados pertenecientes a la Universidad o la carrera, esto se lo realizara en base a roles que tengan los usuarios.
Precondiciones:	Están inscrito en un Topic especifico, acceso a internet
Entradas:	Comunicado, Topic para él envió.
Proceso:	El docente creara un comunicado y enviara a cierto grupo de estudiantes, los estudiantes recibirán la notificación push indicando que ha recibido el comunicado y podrá verlo en la pantalla.
Salidas:	Comunicado
Postcondiciones:	Tener acceso a internet para recibir la notificación.
Roles involucrados:	Usuario final
Verificación:	Si

6º. Visualización de noticias y eventos como parte del portal universitario

Cod:	RF 007
Descripción:	Poder visualizar noticias y eventos en la aplicación.

Descripción detallada:	Desde la aplicación se podrá visualizar las noticias y eventos pertenecientes a la universidad, además estas podrán ser filtradas de acuerdo con las sedes a las cuales pertenecen estas noticias o eventos.
Precondiciones:	Tener internet
Entradas:	
Proceso:	Una vez que ingrese a la aplicación esta recuperara por secciones las noticias y los eventos pertenecientes a la universidad, estas luego podrán ser filtradas de acuerdo con las necesidades del usuario.
Salidas:	Noticas y eventos de la universidad.
Postcondiciones:	Tener acceso a internet
Roles involucrados:	Usuario final.
Verificación:	Si

7º. Visualización de información institucional.

Cod:	RF 008
Descripción:	Visualizar la información institucional del estudiante.
Descripción detallada:	Tendrá una pantalla en la cual se mostrará la información institucional del estudiante que ha realizado el login en la aplicación

Precondiciones:	Tener internet para descargarse la información por primera vez.
Entradas:	Datos del estudiante.
Proceso:	Al momento que ingresa a la pantalla se consumirá un servicio web el cual devolverá la información para poder mostrarla en la pantalla de la aplicación.
Salidas:	Información institucional.
Postcondiciones:	
Roles involucrados:	Usuario final.
Verificación:	Si

8º. Presentación de estados de solicitudes de estudiantes.

Cod:	RF 009
Descripción:	Presentar las solicitudes realizadas por los estudiantes.
Descripción detallada:	Se podrá visualizar las solicitudes realizadas por los estudiantes además de los estados que ha tenido dicha solicitud.
Precondiciones:	Tener internet y que el usuario registrado sea un estudiante.
Entradas:	Datos del estudiante.
Proceso:	Al momento de ingresar a la pantalla se consume un servicio web el cual nos devolverá como

	respuesta las solicitudes realizadas por los estudiantes con sus estados.
Salidas:	Solicitudes y estados de solicitudes.
Postcondiciones:	
Roles involucrados:	Usuario final tipo estudiante
Verificación:	Si

9º. Visualización de las calificaciones de estudiantes.

Cod:	RF 010
Descripción:	Ver las calificaciones del estudiante.
Descripción detallada:	Se podrá visualizar las calificaciones de los estudiantes en las diferentes materias que este cursando.
Precondiciones:	Tener internet y que el usuario registrado sea un estudiante.
Entradas:	Datos del estudiante.
Proceso:	Al momento de ingresar a la pantalla se consume un servicio web el cual nos devolverá como respuesta las calificaciones y las materias de los estudiantes.
Salidas:	Materias con sus calificaciones.
Postcondiciones:	
Roles involucrados:	Usuario final tipo estudiante
Verificación:	Si

10º. Visualización del récord académico de estudiantes.

Cod:	RF 011
Descripción:	Visualización del récord académico de estudiantes
Descripción detallada:	Una pantalla donde se podrá visualizar el récord académico de un estudiante, es decir, todas las materias aprobadas por el mismo, haciendo una comparativa con las materias que le falta por aprobar mediante un gráfico.
Precondiciones:	Tener conexión a internet y que el usuario registrado sea un estudiante.
Entradas:	Datos del estudiante.
Proceso:	Abrir el menú y entrar a la página de récord académico.
Salidas:	Récord académico
Postcondiciones:	-
Roles involucrados:	Usuario final tipo estudiante
Verificación:	Si

11º. Estados de cuenta de estudiantes.

Cod:	RF 012
Descripción:	Poder visualizar el estado de cuenta.
Descripción detallada:	Se podrá visualizar todas las deudas y los pagos realizados por el estudiante.

Precondiciones:	Tener internet y que el usuario registrado sea un estudiante.
Entradas:	Datos del estudiante.
Proceso:	Al momento de ingresar a la pantalla se consume un servicio web el cual nos devolverá como respuesta el estado de cuenta actual del estudiante.
Salidas:	Estado de cuenta actualizado del estudiante.
Postcondiciones:	
Roles involucrados:	Usuario final tipo estudiante
Verificación:	Si

12º. Horario de materias matriculadas o asignaturas asignadas en distributivo.

Cod:	RF 013
Descripción:	Mostrar el horario de las materias.
Descripción detallada:	Se deberá contar con la información sobre horarios de los usuarios, misma que deberá ser almacena en el dispositivo físico. • Estudiantes: descargar el horario de las materias a las cuales se le matriculo en el periodo académico activo. • Docentes: Descargar el horario de las materias que tiene asignado en su distributivo.

	Dicha información se descarga por primera vez, para que las siguientes veces el usuario pueda acceder a la misma sin necesitar estar conectado a internet
Precondiciones:	Tener internet por primera vez.
Entradas:	Datos del estudiante o datos de un colaborador docente.
Proceso:	Al momento haberse registrado en la aplicación, se hace un consumo REST para obtener la información pertinente, la cual será almacenada en el dispositivo físico. Si es un reingreso a aplicación deberá leer los datos almacenados; de no existir los mismo o que de un error, se procederá a descargar nuevamente los datos haciendo una previa limpieza de los mismo.
Salidas:	Horario digital de las materias cursadas o impartidas. Con dicha información se crean automáticamente los chats de grupos de las materias.
Postcondiciones:	Validaciones sobre actualizaciones
Roles involucrados:	Usuario final tipo estudiante
Verificación:	Si

13º. Carnet digital por QR.

Cod:	RF 014
Descripción:	
Descripción detallada:	Recrear el carnet físico institucional por uno digital el cual deberá estar identificado por un código QR único para cada usuario. Dicha información se descarga por primera vez, para que las siguientes veces el usuario pueda acceder a la misma sin necesitar estar conectado a internet
Precondiciones:	Tener conexión a internet por primera vez.
Entradas:	Datos del estudiante o datos de un colaborador docente.
Proceso:	Al momento haberse registrado en la aplicación, se hace un consumo REST para obtener la información pertinente, la cual será almacenada en el dispositivo físico. Si es un reingreso a aplicación deberá leer los datos almacenados; de no existir los mismo o que de un error, se procederá a descargar nuevamente los datos haciendo una previa limpieza de los mismo.
Salidas:	Carnet Digital
Postcondiciones:	-
Roles involucrados:	Usuario Final
Verificación:	Si

ANEXO 2 – MANUAL DE INSTALACIÓN Y CONFIGURACIÓN PARA DESPLIEGUE

Manual de instalación correspondiente a la configuración de Dart y Flutter como entorno de desarrollo y el correcto despliegue de la aplicación finalizada.

Flutter

Para usar el SDK de Flutter es necesario tener en cuenta los requerimientos mínimos del sistema para correr de una manera idónea.

Característica	Windows	Linux	MacOS
Sistema	Windows 7 SP1	Linux 64-bit	macOS
Espacio en disco	1.6 GB	600 MB	2.8 GB
Herramientas	PowerShell 5.0 Git	bash curl git mkdir	git Xcode

Tabla A2 1.- Requerimientos mínimos para SDK Flutter (flutter-dev, n.d.)

Para el desarrollo de un proyecto de Flutter, no es necesario un sistema operativo en específico, salvo el caso de que se requiera compilar para dispositivos iOS, en tal caso se requiere su sistema operativo como base de desarrollo. A continuación, se detallará la configuración de Flutter y del entorno de desarrollo para que funcione adecuadamente en entorno Linux, específicamente en Ubuntu 20.

- Java

  Completamente requerido para este entorno de desarrollo es tener el paquete JDK instalado, muchas versiones de Linux, viene incluido OpenJJDK, el cual también es útil y de no tener el mismo se lo puede instalar ya que es más fácil, sin embargo, se ha escogido trabajar con el paquete JDK (JDK – 11.0.7) propio de Java Oracle. Para lo cual deberemos descarga de su página oficial, escoger un directorio destinado en el sistema, y generar las variables globales, lo cual conseguimos con los siguientes comandos.

```
sudo update-alternatives --
install /usr/bin/java java /opt/JDK/jdk-11.0.7/bin/java 1

sudo update-alternatives --
install /usr/bin/javac javac /opt/JDK/jdk-11.0.7/bin/javac 1

sudo update-alternatives --
install /usr/bin/jar jar /opt/JDK/jdk-11.0.7/bin/jar 1

sudo update-alternatives --set java /opt/JDK/jdk-
11.0.7/bin/java

sudo update-alternatives --set javac /opt/JDK/jdk-
11.0.7/bin/javac

sudo update-alternatives --set jar /opt/JDK/jdk-11.0.7/bin/jar

update-alternatives --set javaws /opt/JDK/jdk-
11.0.7/bin/javaws

update-alternatives --set mozilla-javaplugin.so /opt/JDK/jdk-
11.0.7/jre/lib/amd64/libnpjp2.so
```

  Posterior a la generación de las variables globales, se realizan las respectivas verificaciones de una instalación correcta, para este caso podemos usar el siguiente comando:

```
java –-version
```

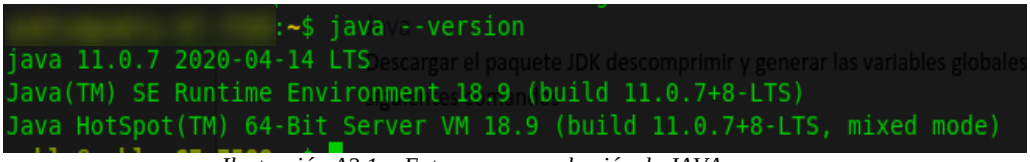

Ilustración A2 1. - Entorno, comprobación de JAVA

- Android SDK

  Descargamos el paquete de las fuentes oficiales y lo ubicamos de la misma manera en un directorio destinado para ello. Para generar las variables de entorno de `ANDROID_HOME`, y cual otra que sea requerida, se edita el archivo `profile`, en donde a su vez agregaremos a la variable `PATH`.

```
sudo nano /etc/profile
      export ANDROID_HOME=/home/user/Android/SDK
      export PATH=$PATH:$ANDROID_HOME/platform-tools
sudo source /etc/profile
```

- Flutter SDK

  Obviamente deberemos realizar la instalación y configuración pertinente para el SDK de Flutter, teniendo en cuenta que este ya integra todo lo que compete con el SDK de Dart, así que no es necesario una instalación adicional para el mismo. En primera instancia, debemos tener presente que tiene ciertas librerías que se debe tener instaladas para su correcto funcionamiento, tales como: curl, bash, mkdir, which, etc. Mismas que en muchos casos ya vienen preinstaladas con el sistema operativo, sin embargo, hay tener pendiente.

  Una vez verificado los requisitos previos, descargamos la última versión estable de la página oficial de Flutter, y de la misma forma extraemos la información y

la ubicamos en el directorio destino para Flutter en nuestra máquina, para poder agregar las variables de entorno, aplicamos las siguientes instrucciones:

```
sudo nano /etc/profile
      export PATH="/opt/flutter/bin:$PATH"
sudo source /etc/profile
```

Antes de finalizar, comprobamos en primera instancia la ubicación de las variables y que las mismas estén disponibles.

```
which flutter
```

Ilustración A2 2.- Entorno, comprobación de variable de entorno Flutter

```
which flutter dart
```

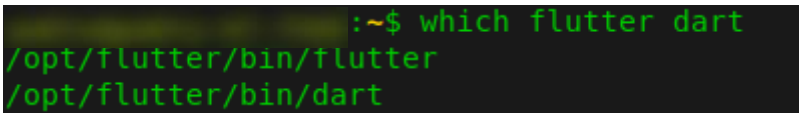

Ilustración A2 3.- Entorno, comprobación de variable de entorno Dart

Es importante desde el inicio tener la seguridad de una correcta configuración, de nuestro entorno ya que es nuestra base durante todo el desarrollo de la aplicación. El SDK de Flutter integra una herramienta que es de gran utilidad para verificar si el mismo está en óptimas condiciones y sin ninguna dependencia faltante, y esta es `flutter doctor`, comando que lo podremos ejecutar en la consola de comandos, para visualizar toda la información.

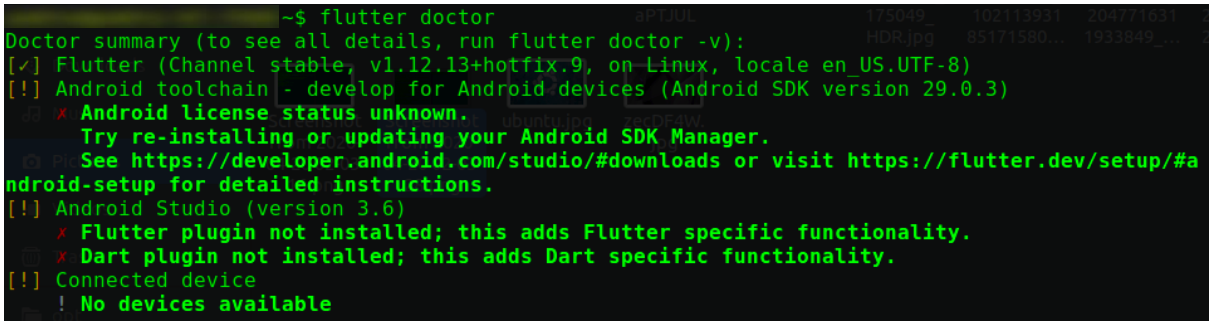

Ilustración A2 4.- Flutter Doctor

Como se aprecia en la Ilustración 8, observamos la configuración actual de Flutter y su relación con sus dependencias más importantes, nos muestra como errores o advertencias según el caso. Para nuestro caso, debemos solucionar la advertencia de Android License, el cual se nos presenta como advertencia y podríamos pasar por alto, sin embargo, después tendremos problemas a la hora de firmar la aplicación o usar APIs externas, por esto es por lo que presentamos la solución a este inconveniente:

```
Flutter doctor --android-licenses
```

de no funcionar, actualizar Flutter.

```
flutter upgrade
```

```
flutter doctor --android-licenses
```

IDE de desarrollo.

Se ha escogido, Visual Studio Code, como IDE de desarrollo por sus grandes ventas a la hora de codificar, además que tiene una gran compatibilidad con librerías de Flutter, así como snippets que nos facilitara mucho el desarrollo de la aplicación.

Para configurar Dart y Flutter, en este editor debemos instalar las extensiones propias de las mismas. Buscamos "Flutter" en el campo de búsqueda de extensiones, seleccione Flutter en

la lista y haga clic en Instalar. Se realiza la misma extensión para instalar el complemento de Dart requerido.

Para finalizar, realizamos una comprobación con el comando flutter doctor, el cual nos valida que el IDE, esta apto para crear, codificar, y correr aplicaciones de flutter.

Ilustración A2 - 5.- Flutter Doctor, verificación IDE

Otras extensiones de gran ayuda dentro de Visual Studio Code:

- Bracket Pair Colorizer 2
- Material Icon Theme
- Paste JSON as Codes
- Terminal

Aplicación y despliegue.

Para dar por finalizado un proyecto en Flutter para móviles, es necesario cierta configuración tanto estética como funcional en cara al despliegue en las distintas tiendas de aplicaciones.

Icono de la aplicación.

Como primer paso, será cambiar el icono de la aplicación, lo cual se puede hacer de forma manual tanto para Android como para iOS, así como usar cierto paquete que nos ahorra dicha configuración, este es `flutter_launcher_icons`, el cual se lo agrega en nuestro archivo pubspec.yaml del proyecto, el apartado de `dev_dependencies`.

Para indicar la ruta de la imagen y su configuración del paquete, se lo realiza posterior a su adición y lo haremos de la siguiente forma:

```
flutter_icons:
  android: “launcher_icon”
  ios:true
  image_path: “assets/icon/ups-icon.png”
  adaptive_icon_backgroud: “assets/icon/ups-icon.png”
```

Ya configurado el paquete en el archivo pubspec.yaml, procedernos a ejecutar el siguiente comando, en la terminal para que se ejecuta dicha configuración y agregue automáticamente los archivos necesarios a los directorios de Android y iOS.

```
flutter packages pub run flutter_launcher_icons:main
```

Nombre de la aplicación.

Para Android, la configuración necesaria es el archivo “Manifes.xml”, en el cual encontramos la etiqueta “android:label” la cual es la que deberemos cambiar por el nombre deseado.

En el caso de iOS, el archivo donde deberemos hacer la modificación es el de “info.plist”, en este estará la llave:

```
<key>CFBundleName</key>,
<string>UPS</string>
```

En la cual podremos cambiar por el nombre deseado.

Ilustración A2 5.- Icono y nombre de la aplicación

Despliegue Android

1. Configurar el ID de la aplicación, el cual se encuentra el archivo “build.gradle” del directorio app. Este APPID deberá ser único para nuestra aplicación y no se puede repetir en todo el mundo. Adicionalmente deberemos cambiar por el mismo APPID en el archivo “manifest.xml”.
2. Especificar la versión que tendrá la aplicación en el mismo “build.gradle”.

   versionCode: indica el número de versión que deberemos incrementar cada vez que se despliegue una versión de la aplicación.

   versionName: es el nombre de la versión que vera el usuario final.
3. Especificar la versión del sistema Android mínima con la cual aplicación puede correr sin problema, este casi siempre dependerá de las diferentes funciones y/o paquetes que use la aplicación. Para nuestro caso se ha colocado como versión mínima el SDK 21 el cual corresponde a Android 5.0 Lolipop en adelante.
4. Firmar la aplicación:
   a. Creamos una llave con el siguiente comando provisto por la documentación oficial:

```
keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -
validity 10000 -alias key
```

Una vez ejecutado dicho comando, nos pedirá que ingresemos una contraseña, y seguimos la configuración que nos va presentando, hasta que se genere el archivo de certificado.

b. Ahora asignamos la llave previamente generada a nuestra aplicación, para lo cual en nuestra carpeta Andorid, crearemos un archivo con el nombre "key.properties", en el cual agregaremos lo siguiente con nuestra configuración:

```
storePassword=<password from previous step>
keyPassword=<password from previous step>
keyAlias=key
storeFile=<location of the key store file, such as /Users/<user name>/upload-keystore.jks>
```

Importante no subir este archivo algún repositorio para mayor seguridad, a menos que sea completamente privado.

c. Configurar la firma en el gradle.

Adicionamos la información de la llave del archivo de propiedades:

```
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')
if (keystorePropertiesFile.exists()) {
       keystoreProperties.load(new
       FileInputStream(keystorePropertiesFile))
 }
```

Buscamos la etiqueta "buildTypes" y lo remplazamos por:

```
signingConfigs {
       release {
           keyAlias keystoreProperties['keyAlias']
```

```
            keyPassword keystoreProperties['keyPassword']
            storeFile       keystoreProperties['storeFile']       ?
file(keystoreProperties['storeFile']) : null
            storePassword keystoreProperties['storePassword']
        }
    }
    buildTypes {
        release {
            signingConfig signingConfigs.release
        }
    }
```

5. Generar la versión de producción de la aplicación, para ello hacemos con el siguiente comando de Flutter, en la raíz del proyecto:

    a. APP bundle

```
flutter build appbundle
```

    b. APK

```
flutter build apk --split-per-abi
```

Google actualmente está soportando ambas versiones, ya sea APK o AppBundle, pero lo que sí es completamente necesario es que la aplicación sea compatible con dispositivos de 64bits, para el caso de APK abre que hacer cierta configuración adicional; para el caso de AppBundle ya viene por defecto la dicha compatibilidad, tanto armeabi-v7a (ARM 32-bit), arm64-v8a (ARM 64-bit), and x86-64 (x86 64-bit). (flutter-dev, Flutter Deploymeny, n.d.)

6. Subir el APK o AppBundle, a la PlayStore

Despliegue iOS

"Para desplegar aplicaciones en la Apple AppStore, es necesario pagar 100 dólares ANUALES, a diferencia de Google PlayStore que el servicio es de 30 dólares de por vida, Apple es una membresía de 100 dólares." (Herrera). Adicionalmente es completamente necesario el uso de una Mac.

1. En la página Apple Developer, ingresar a la cuenta y realizar la configuración necesaria para crear una nueva aplicación en la cuenta.
    a. Crear un AppID, con un nombre de la aplicación de referencia.
    b. Seleccionamos la opción de "Explicit AppID"
    c. Llenamos la casilla del Bundle ID, el cual es único para nuestra aplicación, el cual por convención suele ser el mismo nombre usado en la configuración de Andorid.
    d. Seleccionar los servicios que se requieran.

2. Crear la aplicación en AppStore Connect.
    a. Generáramos una nueva App
    b. Seleccionamos el BundleID, generado previamente.
    c. Agregamos el nombre de la aplicación el cual deberá ser único.
    d. Llenamos toda la información de detalle de la aplicación.

3. Preparar nuestro proyecto a través de Xcode.
    a. Abrimos el archivo "Runner.xcworkspace"
    b. Apartado Identity: Cambiar el BundleID, el cual deberá ser el mismo especificado en la página de Apple.

c. Apartado Signing: para firmar la aplicación en este caso, seleccionamos la opción “Automatically” y seleccionamos nuestra cuenta de Apple Developer.

4. Crear el archivo Build, el cual lo subiremos al AppStore, para ello ejecutamos el siguiente comando:

```
flutter build ios –release
```

Lista de Ilustraciones Anexo 2

Lista de Tablas Anexo 2

ANEXO 3 – MANUAL DE USUARIO

Ventana Home

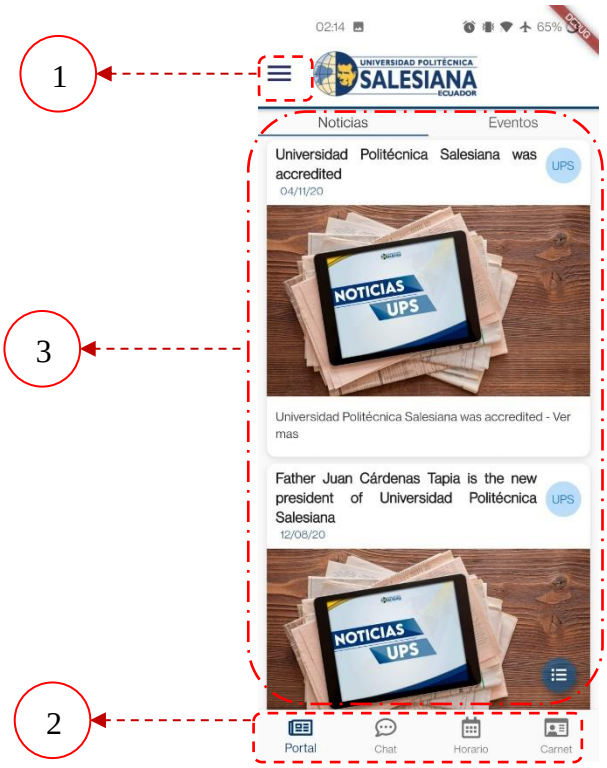

Ilustración A3 1.- UI Home

1) Barra de navegación para la ventana home.
2) Botón de menú opciones.
3) Zona de cada pantalla.

Ventana Home – Portal

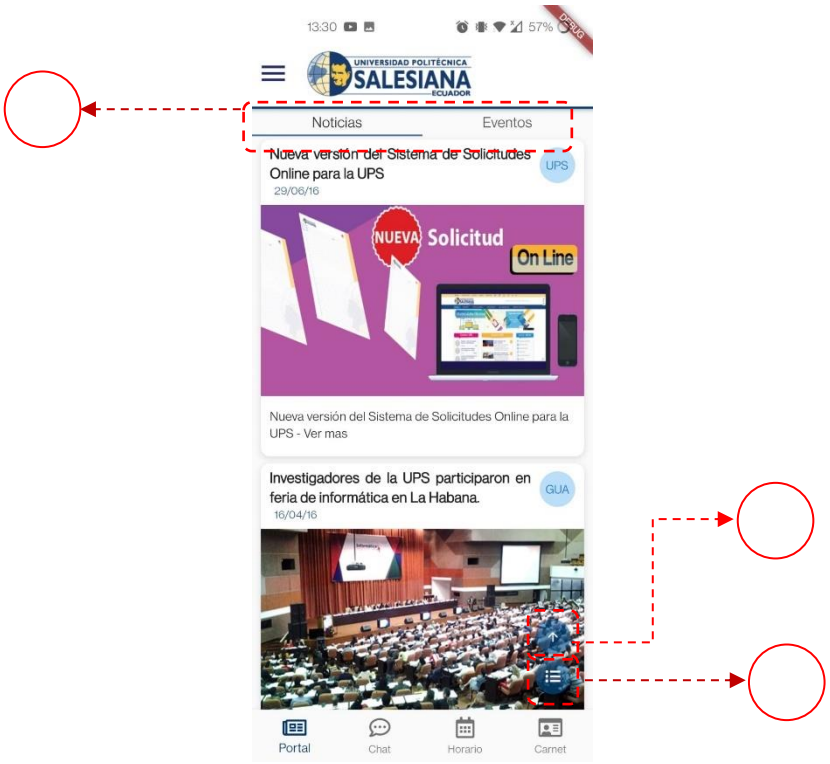

Ilustración A3 2.- UI Portal.

1) Tabs para Noticias y Eventos, se puede deslizar para cambiar entre ellas o hacer clic en las mismas.
2) Botón de filtro, se selecciona la sede en la cual se queira filtrar.
3) Botón para desplazar automaticamente hacia el incio de lista, este aparece una vez se haga un scroll verticcal hacia abajo y pase de cierto umbral.

Ventana Home – Horario

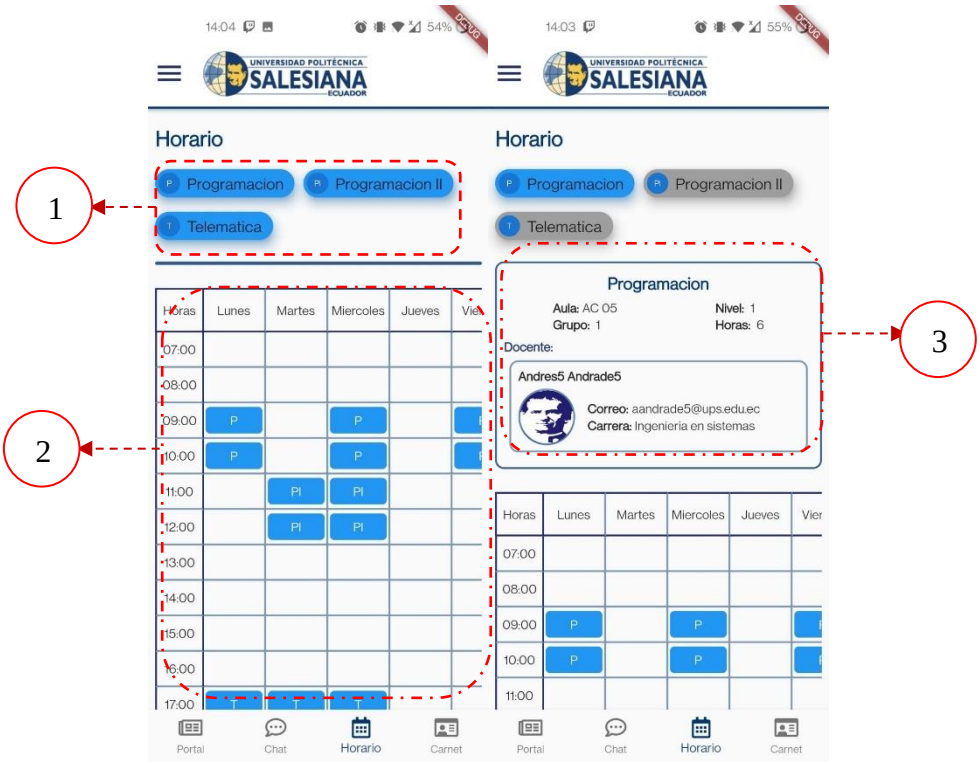

Ilustración A3 3.- UI Horario

1) Etiquetas para cada asignatura.
2) Horario de todas las asignaturas.
3) Área de infromación sobre una materia en concreto, incluye la información de el/los docentes que la imparten.

- Al momento de seleccionar una materia se presenta una animación que da lugar a al ifromación de dicha materia.

Ventana Home – Carnet

Ilustración A3 4.- UI Carnet

1) Zona de carnet digital Frontal.
2) Zona de carnet digital Posterior.
3) Indicación que se peude realizar una acción de Tap para girar y mostrar la parte posterior, sin embargo dicha función es valida al tocar cualqueir parte del carnet.

Menú de opciones

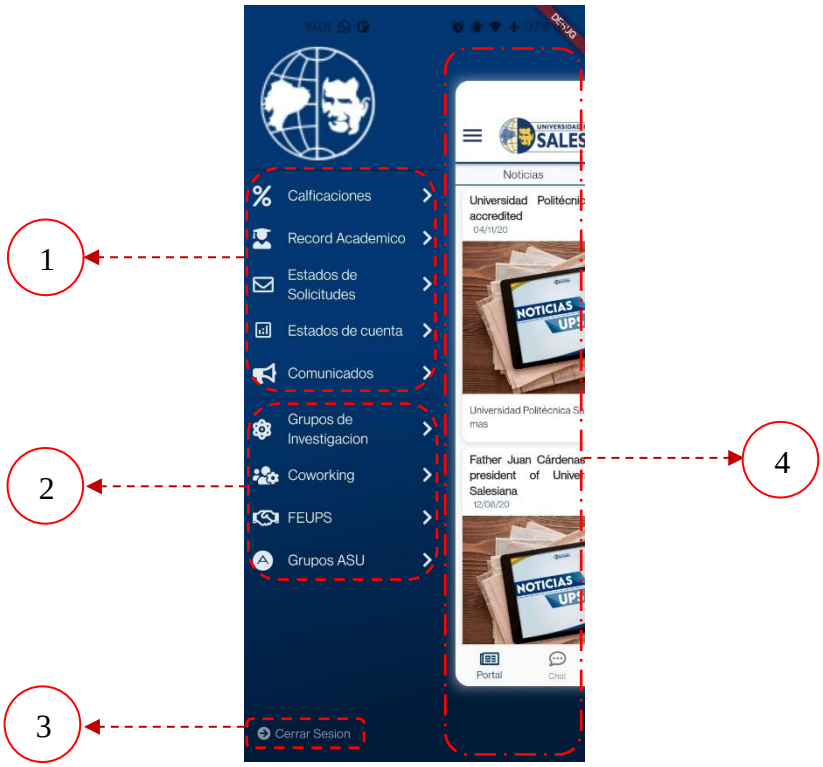

Ilustración A3 5.- UI Menú de opciones desplegado.

1) Botones de opciones según usuario, abren una nueva ventana.
2) Botones de opciones para ventanas de información institucional.
3) Botono de cerrar sesión.
4) Área que cierra el menú y vuelve a la pantalla Home.

Ventana de noticia

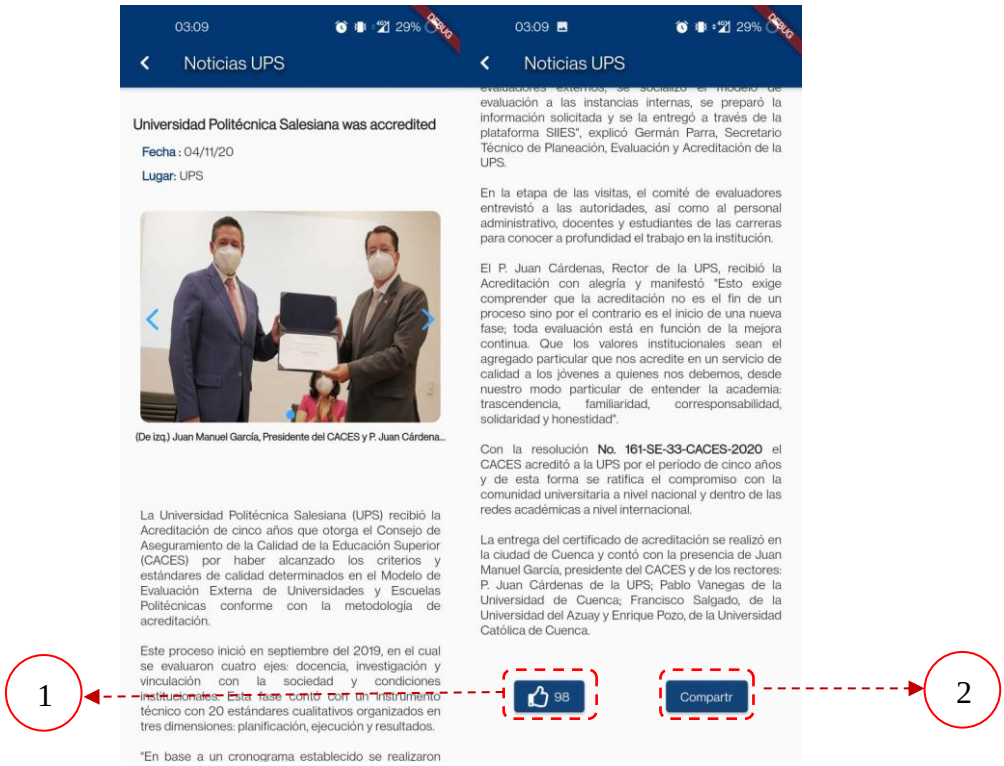

Ilustración A3 6.- UI Notica, misma distribución para Eventos

1) Botón de Like para la noticia o evento.

2) Botón para compartir,

- Abre ocpciones propias del dispositvo para compartir.

Venta información Institucional

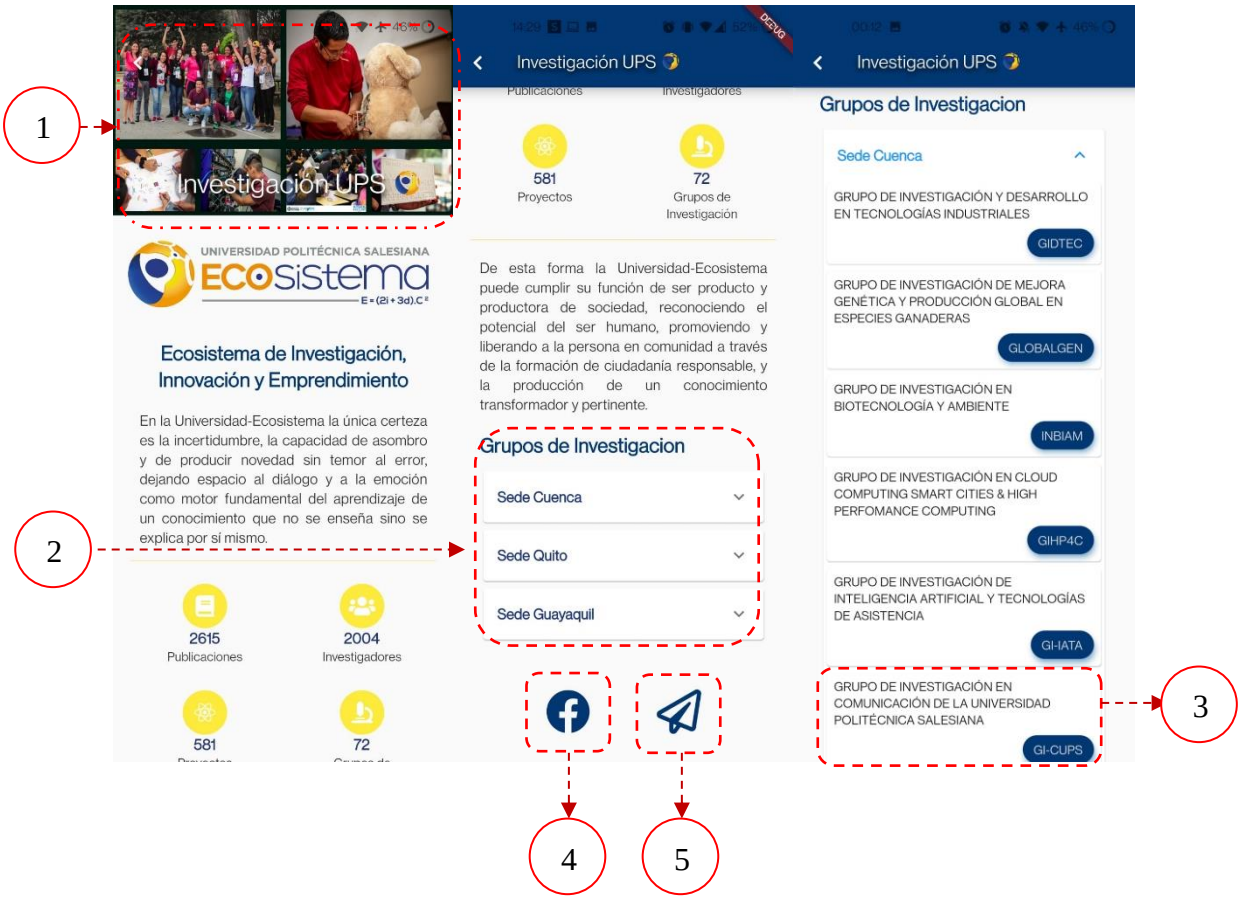

Ilustración A3 7.- UI Ventas de información institucional, misma estructura para las demás opciones.

1. App bar personalizado, muestra imagen general de la página.

Opciones desplegables de información de agrupaciones.

1) Infromacián de un grupo en concreto.
2) Botán de envío a redes sociales,
3) Botón de envío a pagina oficial de la pagina en concreto.

- Al dar clic en 3, se abre el navegador del dispositivo para ir a la pagina del grupo interno en concreto.

- Al dar clic en 4, se abre la aplicación de Facebook del dipositivo y la lleva la pagina del grupo, en el caso de no tener la apliacacion abrira el navegador.
- Al dar clic en 5, abre el navegador del dispositivo para ir a la pagina oficial del grupo.

Lista de Ilustraciones Anexo 3