Diseño e Implementación de una Aplicación Móvil Multiplataforma Dirigida a la Comunidad Estudiantil de la Universidad Industrial de Santander

Edward Javier Parada Silva, Gianfranco Estevez Ruiz y Jose Jaime Silva Martinez

Trabajo de Grado para optar al título de Ingeniero de Sistemas

Director

Gabriel Rodrigo Pedraza Ferreira

Doctor en Ciencias de la Computación

Codirector

Danny Felipe Vergel Paba

Especialista en Gerencia de Proyectos

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería de Sistemas e Informática

Bucaramanga

2023

Dedicatoria

Edward Parada

A la Universidad Industrial de Santander y a sus trabajadores.

Gianfranco Estevez

A mi Schnauzer Tobita, por acompañarme y ser el apoyo emocional en cada día de esta etapa de mi vida.

A mí mismo, por no renunciar y seguir adelante a pesar de las adversidades y momentos de debilidad.

José Silva

A mi familia, por estar ahí cuándo más los necesité.

A mi mismo, por demostrarme de lo que soy capaz y las ganas que tengo de salir adelante.

Agradecimientos

Al profesor Gabriel Pedraza por su orientación durante el desarrollo del proyecto de grado.

Al Ingeniero Danny por su paciencia en el acompañamiento realizado.

Al profesor Gilberto Diaz, quien consolidó la idea para la realización de este proyecto.

A nuestro lider de equipo Adrián González, quien fue parte fundamental para la realización de este proyecto.

A los profesores Jathinson, Lola y todos los demás quienes hicieron parte de nuestro aprendizaje.

Edward Parada

A mi padre, mi madre, y mi abuela, quienes estuvieron conmigo durante todo el camino.

A Carmencita del decanato de fisicomecánicas, quien me empujó a desarrollar todo mi potencial,

Gianfranco Estevez

A mi madre por todo el apoyo brindado para poder desarrollarme adecuadamente en esta etapa.

Al profesor Jathinson por el apoyo, la compañía y la ayuda invaluable que me brindó cuando más lo necesité.

José Silva

A mi madre, y mis hermanos, quienes estuvieron acompañándome y apoyándome durante todo el proceso.

A mi tía Martha, quien me dio mucha motivación y disciplina durante todo mi proceso de formación.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Resumen

Título: Diseño e Implementación de una Aplicación Móvil Multiplataforma Dirigida a la Comunidad Estudiantil de la Universidad Industrial de Santander [*]

Autores: Edward Javier Parada Silva, Gianfranco Estevez Ruiz y Jose Jaime Silva Martinez [**]

Palabras Clave: App, Carnet, Multiplataforma, Estudiantes.

Descripción: La aplicación estudiantil actual de la Universidad Industrial de Santander presenta varios desafíos que afectan la experiencia de los estudiantes. Esto se debe a una interfaz poco atractiva y la limitada accesibilidad a través de tiendas de aplicaciones populares. Además, el carnet físico de estudiante utilizada para la identificación y el acceso a servicios representa un costo significativo para la universidad y puede causar complicaciones a los estudiantes. La falta de control en el acceso al campus y la suplantación de identidad también ha generado preocupaciones de seguridad. Para abordar estos problemas, este proyecto propone el diseño e implementación de una aplicación estudiantil multiplataforma que integre módulos de información académica y una alternativa digital al carnet físico de estudiante. La aplicación tiene como objetivo mejorar la experiencia académica de los estudiantes y proporcionar una herramienta valiosa para su desarrollo profesional. La solución propuesta ofrece una plataforma moderna y accesible para la comunidad de la Universidad Industrial de Santander para acceder a servicios estudiantiles esenciales y mecanismos de identificación seguros.

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Gabriel Rodrigo Pedraza Ferreira

Abstract

Title: Design and Implementation of a Multi-Platform Mobile Application for the Industrial University of Santander's Student Community [*]

Authors: Edward Javier Parada Silva, Gianfranco Estevez Ruiz and Jose Jaime Silva Martinez [**]

Keywords: App, ID card, Multi-platform, Students.

Description: The current student application of the Industrial University of Santander presents several challenges that affect the students' experience. This is due to an unattractive interface and limited accessibility through popular app stores. Additionally, the physical student ID card used for identification and access to services represents a significant cost for the university and can cause complications for students. The lack of control over campus access and identity impersonation has also raised security concerns. To address these issues, this project proposes the design and implementation of a multi-platform student application that integrates modules for academic information and a digital alternative to the physical student ID card. The application aims to improve the academic experience of students and provide a valuable tool for their professional development. The proposed solution offers a modern and accessible platform for the Industrial University of Santander community to access essential student services and secure identification mechanisms.

---

[*] Bachelor Thesis

[**] Faculty of Physicomechanical Engineering. School of Systems and Computer Engineering. Director: Gabriel Rodrigo Pedraza Ferreira

Introducción

Una aplicación móvil es una aplicación capaz de ejecutarse en dispositivos móviles, notablemente teléfonos inteligentes. La asociación de estos últimos con el acceso a Internet, un diseño portable, la facilidad de manejo; además de la naturaleza personal de un número telefónico, han dado lugar al uso de aplicaciones móviles como una solución ante la problemática del acceso seguro a datos confidenciales y/o personales, como puede evidenciarse en las aplicaciones bancarias, carteras virtuales y cuentas de correo, entre otros.

Hoy en día, la Universidad Industrial de Santander cuenta con su propia aplicación móvil para la comunidad estudiantil. Esta app permite el acceso a información académica y a solicitudes relacionadas con el trabajo de grado. Sin embargo, se han identificado varios problemas que afectan su utilidad y accesibilidad para los estudiantes. Por ejemplo, la aplicación carece de funcionalidades amigables con el usuario, como la capacidad *offline* para acceder a los módulos de la aplicación sin conexión a Internet. También se ha señalado la falta de una alternativa al carnet físico estudiantil, así como la limitada disponibilidad y distribución de la aplicación en tiendas de aplicaciones en una cantidad significativa de dispositivos móviles. Otro problema es la inapropiada difusión de la aplicación, lo que limita su alcance. Además, la aplicación no ofrece funcionalidades que ayuden al estudiante a planificar su semestre y cuenta con estándares de diseño antiguos, lo que resulta en una experiencia de usuario pobre y limitada. Estos son algunos de los problemas

identificados por los autores del proyecto que requieren una solución para mejorar la calidad de la aplicación móvil actual con la que cuenta la universidad.

Del mismo modo, se ha encontrado que varias (si no todas) las funciones del carnet físico estudiantil pueden ser reemplazadas por una alternativa virtual, removiendo varias de sus desventajas, entre las que se encuentran su costo anual de producción para la universidad, la necesidad de un tiempo de expedición y la posibilidad de perdida u olvido. Esta alternativa virtual también busca mitigar uno de los problemas más grandes que atraviesa la universidad en su historia reciente, como lo es la inseguridad dentro del campus. Donde no se tiene un control de las personas ingresan al mismo, así como también se evidencia un problema de suplantación de credenciales/identidad, proliferando así el ingreso de personas malintencionadas que no pertenecen a la comunidad universitaria y creando una sensación de inseguridad dentro de la misma.

Este proyecto se enfocó en diseñar e implementar una aplicación basada en la propuesta mencionada anteriormente. El documento abarcará el alcance de la propuesta, su arquitectura, su implementación y las validaciones realizadas.

1. Planteamiento Y Justificación Del Problema

La aplicación estudiantil actual de la Universidad Industrial de Santander presenta varios desafíos que afectan negativamente la experiencia de los estudiantes. En primer lugar, está construida sobre tecnologías antiguas, lo que dificulta su uso y le confiere un aspecto poco atractivo, lo cual es preocupante dados los estándares de diseño actuales. Además, la aplicación se distribuye como un archivo instalable para *Android* y no está disponible en tiendas de aplicaciones populares como *Play Store* o *App Store*, lo que puede disuadir a los estudiantes de utilizarla y limitar su capacidad para acceder a información importante (Delia, 2015).

Por otra parte, el actual carnet físico es utilizado por los estudiantes para identificarse en la universidad y acceder a servicios como comedores y préstamos de libros en la biblioteca. Sin embargo, la emisión de estos carnets representa un costo significativo para la universidad cada semestre, y en caso de extravío, olvido o robo, puede causar complicaciones y retrasos a los estudiantes, además de afectar la capacidad de la universidad para proporcionar un servicio eficiente.

Además, la universidad ha experimentado problemas de seguridad recientemente debido a la falta de control en el acceso al campus y la suplantación de identidad. Esto ha permitido el ingreso de personas ajenas a la comunidad universitaria, posiblemente con intenciones malintencionadas, lo que ha generado una sensación de inseguridad dentro de la institución.

Estos problemas ponen de manifiesto que tanto los estudiantes de la Universidad Industrial

de Santander como la propia universidad enfrentan un problema crítico: la falta de una aplicación estudiantil actualizada que integre módulos de información académica y verificación de identidad. La necesidad de encontrar una solución para este problema se ha vuelto esencial para mejorar la experiencia académica de los estudiantes y proporcionarles una herramienta valiosa para su desarrollo profesional. Por lo tanto, se propone diseñar e implementar una aplicación multiplataforma, accesible y acorde a los estándares de diseño y desarrollo actuales, que incluya módulos de información académica inspirados en la aplicación estudiantil actual y una alternativa digital al carnet físico con las respectivas medidas de seguridad.

2. Marco de Referencia

2.1. Marco Conceptual

El *frontend*, también conocido como interfaz de usuario o capa de presentación, se refiere a la parte visible y accesible de una aplicación o sitio web con la que los usuarios interactúan directamente (Lemonaki, 2022). Es la parte del sistema que se encarga de mostrar la información y proporcionar una experiencia de usuario intuitiva y atractiva. Este se construye utilizando tecnologías web como HTML, CSS, y *JavaScript*. La evolución del *frontend* ha llevado al desarrollo de marcos de trabajo (*frameworks*) y bibliotecas que simplifican y agilizan el proceso de construcción de interfaces de usuario. Algunos ejemplos populares de estos son *Angular*, *React* y *Vue*.

Por otra parte, el *backend* es la parte de una aplicación o sistema que se encarga de procesar y gestionar los datos y la lógica de negocio (Pahl & Jamshidi, 2016). Es responsable de todo lo que sucede detrás de escena en una aplicación, incluyendo la gestión de la base de datos (la cual almacena la información usada en el negocio) la autenticación y autorización de usuarios, y la integración con otros sistemas y servicios. En el contexto de la arquitectura de relacionada a servicios, el *backend* se divide en varios servicios que se comunican entre sí para realizar tareas específicas (Newman, 2015). Cada servicio se encarga de una función específica, lo que permite una mayor flexibilidad y escalabilidad en el desarrollo y mantenimiento de la aplicación. En la figura 1 se puede observar un diagrama mostrando la relación entre el *frontend*, *backend* y la base

de datos.

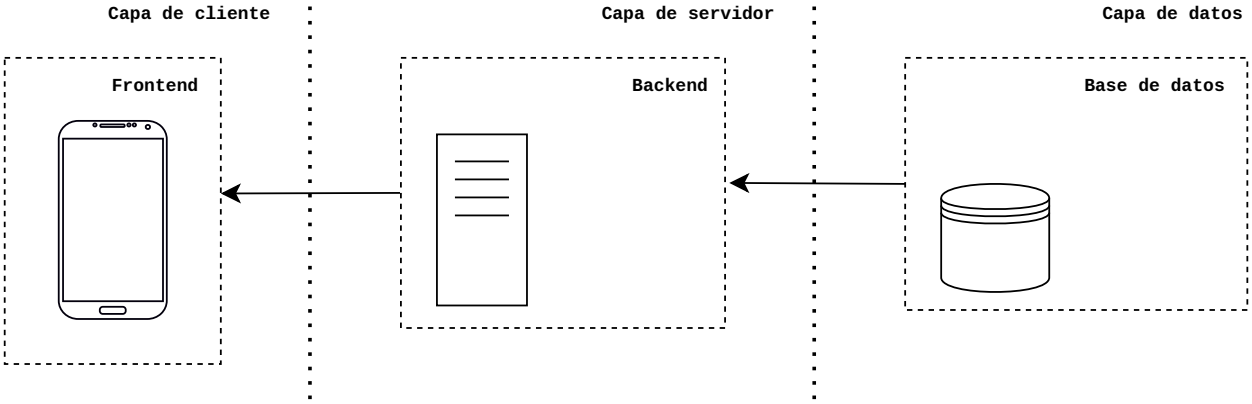


Figura 1. Diagrama básico de una aplicación con backend, frontend y base de datos.

Generalmente, tanto el desarrollo del *backend* como el del *frontend* se lleva a cabo utilizando *frameworks*. Un *framework* es una estructura o conjunto de herramientas y componentes que proporcionan una base para el desarrollo de aplicaciones o sistemas. Según la definición de (Gamma, 1994), es un esqueleto o infraestructura reutilizable que puede ser extendido y personalizado para construir aplicaciones específicas.

Una API es una interfaz que permite la comunicación entre dos o más programas/aplicaciones. En esta se establecen las reglas que se deben seguir con el fin de que dichos programas/aplicaciones puedan comunicarse. Por otro lado, REST (*Representational State Transfer*) es una arquitectura de software que impone condiciones sobre como debe funcionar una API (Services, 2022). Se basa en los principios fundamentales del protocolo HTTP, utilizando los verbos GET, POST, PUT y DELETE para operar sobre recursos identificados por URLs (*endpoints*). Según Roy Fielding, uno de los autores de la especificación HTTP y creador del concepto REST, una

API REST es un conjunto de restricciones y propiedades que, cuando se aplican a los componentes de una arquitectura de software, mejoran la escalabilidad, la simplicidad y la interoperabilidad del sistema (Fielding, 2000).

El patrón *Modelo-Vista-Controlador* (MVC, por sus siglas en inglés: *Model View Controller*) es un diseño arquitectónico ampliamente utilizado en el desarrollo de software. Se basa en la separación de responsabilidades en tres componentes principales: el modelo, que representa la estructura de datos y la lógica de negocio; la vista, que maneja la presentación de la información al usuario y la interacción con él; y el controlador, que actúa como intermediario entre el modelo y la vista, gestionando las solicitudes del usuario y actualizando el modelo y la vista en consecuencia (Apple, 2023).

Por otro lado, el patrón de diseño *Repository* es una abstracción que se utiliza en el desarrollo de software para separar la lógica de acceso a datos de la lógica de negocio. Este patrón "mediana entre las entidades de dominio y la capa de acceso a datos, habilitando una separación de responsabilidades que aísla los detalles de almacenamiento de los objetos de dominio" (Fowler, 2002). El patrón *Repository* proporciona métodos para realizar operaciones de consulta, inserción, actualización y eliminación en la fuente de datos subyacente, ya sea base de datos, almacenamiento en un dispositivo, u otro mecanismo de almacenamiento. Su objetivo es fomentar la reutilización de código al ofrecer una abstracción coherente de las operaciones de almacenamiento de datos.

El desarrollo móvil multiplataforma se refiere a la creación de aplicaciones de software que

son compatibles con múltiples sistemas operativos (como *iOS y Android*) y dispositivos. Esta metodología permite a los desarrolladores maximizar la eficiencia y la reutilización de recursos al evitar la necesidad de desarrollar y mantener versiones separadas de la aplicación para cada plataforma. Según (Singh & Shobha, 2021), el desarrollo móvil multiplataforma ofrece ventajas significativas en términos de costos, tiempo de desarrollo y alcance del mercado, ya que las aplicaciones pueden llegar a una amplia variedad de dispositivos. Hoy en día, el desarrollo de aplicaciones móviles multiplataforma se ha vuelto más sencillo. Existen nuevas opciones disponibles, como se menciona en (Target, 2022):

- Desarrollo de aplicaciones móviles híbridas: Se desarrollan las aplicaciones utilizando tecnologías como HTML5 o JavaScript, y se ejecutan dentro de un *Web View*.
- Desarrollo rápido de aplicaciones móviles (RMAD): Permite a los usuarios empresariales crear y gestionar aplicaciones lo suficientemente buenas para resolver problemas específicos en el ámbito empresarial.
- Aplicaciones Web Progresivas (PWAs): Son sitios web que tienen apariencia y funcionalidad similar a las aplicaciones móviles. Las PWAs están diseñadas para aprovechar las características de los dispositivos móviles, pero no requieren que el usuario instale una aplicación nativa.

Un *Web View* es un navegador incrustado que las aplicaciones nativas utilizan para mostrar contenido web (Chinnathambi, 2022). A diferencia del navegador tradicional, el contenido web

mostrado en el *Web View* no está limitado por las restricciones de seguridad del navegador, lo que permite que el *JavaScript* en el *Web View* llame a las APIs del sistema nativo (Adinugroho et al., 2015). La comunicación entre el código nativo de la aplicación y el código web se realiza a través de un puente o *bridge*.

JWT, acrónimo de *JSON Web Token*, es un estándar que establece un método compacto y autónomo para transmitir información de forma segura entre diferentes entidades, como objetos JSON (JWT, 2022). Esta información puede ser verificada y confiable, ya que está digitalmente firmada. Estas firmas se pueden generar utilizando algoritmos criptográficos o pares de claves pública y privada. Se recomienda el uso de *JSON Web Tokens* para autorización en aplicaciones, y para transmitir información entre entidades de forma segura.

La carga referida o *Lazy Loading* es una técnica de optimización utilizada en el desarrollo de software que consiste en retrasar la carga de recursos o datos hasta el momento en que sean necesarios. La carga referida reduce el tiempo inicial de carga y el tiempo de carga percibido (Shivakumar, 2020). Según el artículo de Mozilla Developer Network sobre *Lazy Loading* (Network, 2021), esta técnica es especialmente útil para acelerar la carga de imágenes y vídeos en páginas web con una gran cantidad de contenido multimedia. En lugar de cargar todas las imágenes y vídeos de una sola vez, la carga referida permite cargar solo aquellos elementos que son visibles en la pantalla del usuario en ese momento. A medida que el usuario se desplaza hacia abajo en la página, se cargan más elementos de manera selectiva.

Por otra parte, Los *tests unitarios* son una técnica de pruebas utilizada en el desarrollo de software para verificar la funcionalidad de unidades pequeñas de código. Estos tests se escriben junto con el código de la aplicación y ayudan a identificar y corregir errores tempranamente, mejorando la calidad del software (Beck, 2003). Los *tests unitarios* se escriben utilizando *frameworks* y herramientas específicas, como *JUnit* para *Java*, que proporcionan aserciones y métodos para verificar los resultados y automatizar la ejecución de los *tests unitarios* (Team, 2021).

2.2. Marco Tecnológico

2.2.1. Backend. Uno de los frameworks principales para el desarrollo *backend* es *Spring Boot*, el cual es un framework del lenguaje de programación Java que facilita la creación de aplicaciones web y servicios REST, este incluye una amplia variedad de módulos y bibliotecas integradas que facilitan la integración con otras tecnologías, como bases de datos, seguridad, mensajería, etc. También proporciona herramientas integradas para la gestión de dependencias y la generación de proyectos, lo que facilita y agiliza el proceso de desarrollo.

Entre las librerías integradas en Spring Boot, se encuentra *Feign* la cual simplifica la definición de interfaces de cliente para comunicarse con servicios REST y otorga las capacidades de intermediario al *backend* principal, esta suele ser necesaria para acceder a información de otras bases de datos o el uso de servicios transversales del proyecto RSI.

Por otro lado, cuando se va a recuperar información de la base de datos, se usa *Hibernate*, este es un framework de mapeo objeto-relacional (ORM) que permite la interacción con bases de

datos. *Hibernate* se basa en el modelo de datos orientado a objetos, lo que significa que estos se modelan mediante clases de *Java* y permiten su manipulación como si de instancias se tratase.

2.2.2. Frontend. Como *framework* de desarrollo web se utilizó *Angular*, el cual es utilizado para construir aplicaciones web de una sola página (*Single Page Applications*, SPAs) y aplicaciones móviles híbridas. *Angular* ofrece un enfoque basado en componentes para el desarrollo de aplicaciones web, lo que significa que la aplicación se divide en componentes reutilizables y cada componente se encarga de una parte específica de la interfaz de usuario (Angular, 2023).

Para desarrollar la aplicación en multiples plataformas, como *iOS* y *Android*, se utilizó el *framework* de desarrollo *Ionic*. Este permite desarrollar aplicaciones móviles multiplataforma utilizando tecnologías web estándar como HTML, CSS y *JavaScript*. *Ionic* es compatible con *frameworks* web como Angular, e incluye una variedad de componentes que están enfocados para funcionar en dispositivos móviles.

Por otra parte, se utilizó el *framework Capacitor*, desarrollado por el equipo de *Ionic*, el cual actúa como un puente entre el código web y las capacidades nativas de los dispositivos móviles. Es una capa de abstracción que permite a los desarrolladores acceder a las características y funcionalidades nativas de los dispositivos a través de un conjunto de API y plugins (Ionic, 2023).

Una de las principales ventajas de *Capacitor* es su capacidad para utilizar componentes y bibliotecas web existentes, lo que facilita el desarrollo de aplicaciones híbridas. Además, *Capacitor* proporciona una API unificada que permite a los desarrolladores acceder a características nativas

como la cámara, el GPS, los sensores del dispositivo, entre otros.

El flujo para el despliegue a nativo se ilustra en la Figura 2. Para compilar y correr la aplicación en *Android* se utilizó *Android Studio*. Este es el entorno de desarrollo integrado oficial (IDE) para el desarrollo de aplicaciones *Android*. Viene equipado con un amplio conjunto de características y herramientas para ayudar a los desarrolladores a escribir, depurar, probar y perfilar aplicaciones Android con eficiencia (Studio, 2023).

Por el lado de *iOS*, se utilizó *Xcode*, el cual es un conjunto completo de herramientas de desarrollo para crear aplicaciones para macOS, iOS, watchOS y tvOS. Xcode incluye herramientas como editor de código, un depurador completo, herramientas de análisis de rendimiento, simuladores para probar aplicaciones en diferentes dispositivos y muchas otras características (Documentation, 2023).

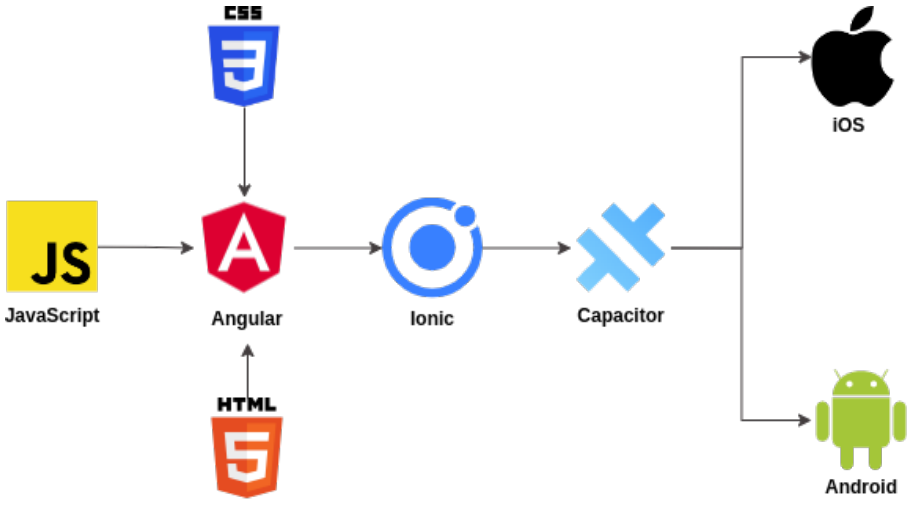


Figura 2. Flujo para el despliegue de la aplicación a Android e iOS nativo.

2.3. Estado del Arte

Considerando el interés a mejorar el servicio ofrecido por las universidades mediante los avances tecnológicos logrados en años recientes, ha aparecido el concepto del campus inteligente o *Smart Campus*, el cual se refiere a la utilización de tecnología avanzada para mejorar la calidad de vida de los estudiantes, profesores y personal administrativo en un campus universitario (Guo, 2018).

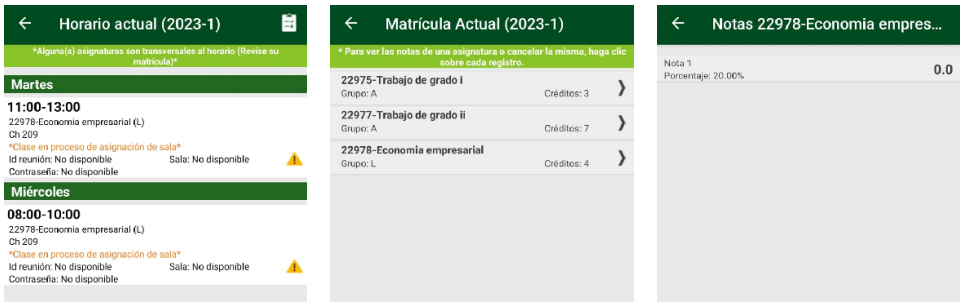
Entre los elementos que componen a un campus inteligente, Guo (2018) menciona la presencia de al menos un terminal o dispositivo que el usuario pueda utilizar para el acceso a los sistemas que cuenta la universidad y fomentar la transformación digital en esta. Del mismo modo (Zhang et al., 2013) menciona la importancia de la recopilación de datos mediante diferentes terminales para la toma de decisiones informadas en el campus.

Dicha terminal, en el contexto del presente proyecto, puede referirse a un dispositivo móvil con la aplicación estudiantil instalada, entre las ventajas que estas terminales ofrecen se encuentran una mayor conveniencia para los usuarios y aquellos encargados de mantener los sistemas de información, pues el diseño del sistema encargado del campus inteligente se enfoca en la integración orgánica de varias fuentes de información y la mejora en la velocidad de respuesta de los procesos (Zhang et al., 2013).

Por otro lado, se debe tener en cuenta la necesidad de preservar los recursos que la universidad utiliza para su operación, mientras que hace alrededor de una década las tecnologías

dominantes eran basadas en dispositivos de identificación por radiofrecuencia (RFID) (Voon et al., 2016), la aparición y rápida expansión de los dispositivos móviles ha generado interés en el uso de estos dispositivos debido a que cuentan con tecnologías como NFC (Roland, 2012) y permiten la generación de códigos QR (Satanasaowapak et al., 2021).

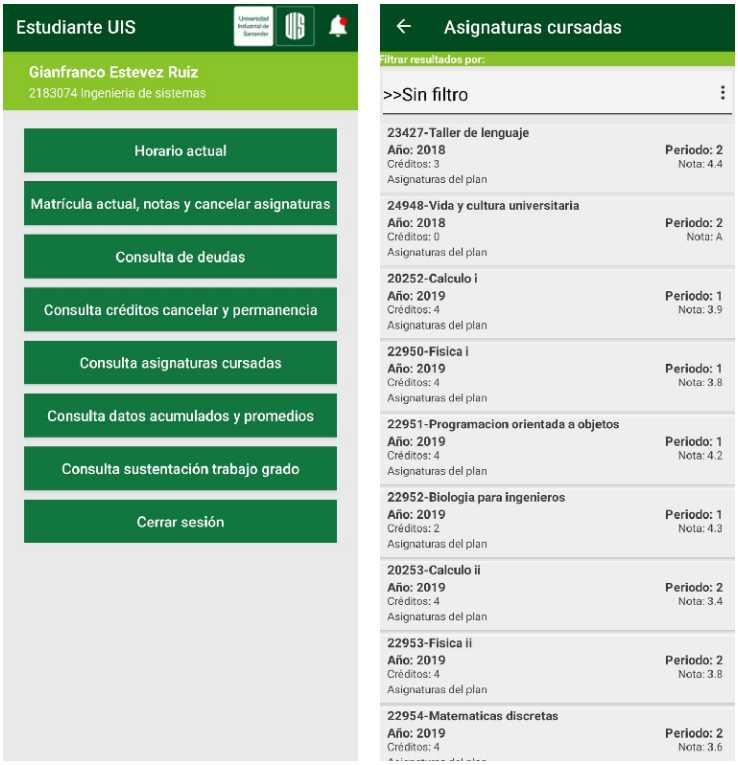
2.3.1. Aplicación oficial actual de académico estudiantes. La Universidad Industrial de Santander cuenta actualmente con una aplicación para estudiantes disponible en dispositivos *Android* con versión 3.0 o superior (UIS, 2022a). Esta aplicación proporciona acceso a funcionalidades como consultar el horario actual, realizar la matrícula actual, verificar deudas, revisar créditos, consultar asignaturas y promedios, así como acceder al proceso de matrícula y al trabajo de grado. Esta aplicación se realizó con Android nativo usando Java en su versión 1.8 y siguiendo los estándares de diseño de *Material*. Por la parte del *backend*, los servicios se realizan por medio del protocolo SOAP. Además cuenta con *ProGuard* como método de ofuscación.



(a) Vista horario (b) Vista principal notas (c) Vista secundario notas

Figura 3. Vistas del horario y notas de la aplicación oficial actual para estudiantes.

Nota. Aplicación Estudiantes UIS, Versión 10.0. (2023). Captura de pantalla. Recuperado de https://uis.edu.co/uis-app-estudiantes-es/



(a) Vista principal (b) Vista de asignaturas cursadas

Figura 4. Vistas de la pantalla principal e histórico semestral de la aplicación oficial actual para estudiantes.

Nota. Aplicación Estudiantes UIS, Versión 10.0. (2023). Captura de pantalla. Recuperado de https://uis.edu.co/uis-app-estudiantes-es/

3. Requerimientos

Los requerimientos establecen las funcionalidades y características que el sistema debe cumplir, así como los criterios de calidad y los estándares de rendimiento que deben ser satisfechos. La documentación adecuada de los requerimientos es esencial para garantizar que el proyecto se

desarrolle con éxito y se cumplan las expectativas de los usuarios. En consecuencia, se dedicó tiempo y esfuerzo para definir y documentar los requerimientos de manera adecuada y precisa, para que todos los interesados en el proyecto puedan entender claramente lo que se espera del sistema. A continuación, se presentarán los requerimientos de la aplicación, tanto funcionales como no funcionales, agrupados por categorías para facilitar la comprensión y lectura de la lista.

3.1. Requerimientos Transversales

Para el desarrollo de los módulos del presente proyecto se plantaron ciertos requerimientos que estarán presentes en todos los módulos. Dichos requerimientos enriquecen la experiencia de usuario y permite que la aplicación tenga más consistencia entre sus módulos. A continuación se presentan los requerimientos planteados.

3.1.1. Requerimientos Funcionales.

- Los módulos de la aplicación deben contar con un *cargador de esqueleto* para visualizar que los datos requeridos están siendo solicitados.
- Los módulos de la aplicación deben contar un *efecto de carga* personalizado y que se debe mostrar siempre de los cabeceros de los mismos.
- Los módulos de la aplicación deben contar con un *refrescador* el cuál al ejecutarse debe volver a realizar las peticiones al *backend* necesarias dependiendo del módulo donde se realiza.

- Los módulos de la aplicación deben contar con un cabecero. Este cabecero debe contener texto indicativo al módulo, un botón con acceso al menú lateral o un botón que redirija a la vista anterior (en caso de tener módulos con más módulos dentro) y dependiendo del módulo contará con un botón extra si se necesita información adicional.
- Los módulos de la aplicación si detectan un cambio en la conexión a Internet, debe desplegar un pie de página momentáneo el cual indique que el estado actual de la red.
- Los módulos de la aplicación si no hay datos disponibles para mostrar, deberá mostrar una pantalla de estado vacío que varía dependiendo de la razón de la falta de datos.

3.1.2. Requerimientos No Funcionales.

- Los módulos de la aplicación deben contar con traducciones en español e inglés y visualizarse dependiendo del idioma del dispositivo donde se despliegue la aplicación.

3.2. Inicio de sesión

3.2.1. Requerimientos Funcionales.

- El módulo de inicio de sesión debe permitir a los usuarios iniciar sesión por medio de su código de estudiante y contraseña.
- El módulo de inicio de sesión debe mostrar el estado del formulario de usuario y contraseña constantemente para avisar al usuario de los posibles errores al diligenciar dicho formulario.

- El módulo de inicio de sesión debe mostrar por medio de mensajes emergentes los posibles errores que puedan suceder al momento de iniciar sesión como lo pueden ser la falta de conexión a Internet, error en las credenciales o error al realizar peticiones.

- Si es la primera vez que se inicia sesión (en cualquier dispositivo), debe mostrar un modal para emparejar el dispositivo con la sesión del estudiante. Si ya se ha iniciado sesión en las anteriores 24 horas en otro dispositivo, el estudiante no podrá iniciar sesión en un nuevo dispositivo hasta cumplir el tiempo restante. De lo contrario se deberá mostrar un modal de autenticación en dos pasos para iniciar sesión.

- Si el módulo de inicio de sesión despliega el modal de autenticación en dos pasos, éste debe solicitar un código de 6 dígitos que se le enviará al correo institucional y personal con el que se registraron al ingresar a la universidad. Dicho código debe tener una caducidad de 10 minutos.

- El módulo de inicio de sesión debe permitir a los usuarios reportar errores por medio de un acceso directo a la página web de la mesa de ayuda ofrecida por la universidad.

- El módulo de inicio de sesión si se está ejecutando en su versión web deberá mostrar un modal con acceso directo a las tiendas de Android, Apple y Huawei.

3.2.2. Requerimientos No Funcionales.

- El modal que indica las horas restantes si ya se inició en un dispositivo en las últimas 24

horas, debe mostrar exactamente cuantas horas restantes quedan para poder iniciar sesión. Si queda menos de una hora, debe mostrar cuantos minutos restantes falta.

- El módulo de login debe mostrar cuál versión de la aplicación está usando.
- El módulo de login cambiará el fondo de pantalla aleatoriamente al iniciar la aplicación.

3.3. Carnet Virtual

3.3.1. Requerimientos Funcionales.

- El módulo de carnet virtual debe mostrar nombres, apellidos, foto y código QR de control de acceso del estudiante.

3.3.2. Requerimientos No Funcionales.

- El módulo de carnet virtual debe mostrar un mensaje de bienvenida, el día y la fecha actual en el header del mismo.

3.4. Horario

3.4.1. Requerimientos Funcionales.

- El módulo de horario debe agrupar y listar por días las materias matriculadas por el estudiante.

- El módulo de horario debe mostrar en cada bloque de horas de clase el salón, la sigla del edificio, la hora de inicio y la hora de finalización. Además, debe mostrar el nombre, código y grupo de la materia.
- El módulo de horario debe tener un botón en cada clase que tenga sala virtual que redirija hacia la aplicación de *Zoom* y entre a la sala correspondiente a la clase.
- El módulo de horario debe tener un botón de ayuda en el cabecero del mismo. Este botón mostrará un modal con las siglas de los edificios presentes en el horario. La información listada en este modal es variable y depende de los edificios asignados a las clases del estudiante.

3.4.2. Requerimientos No Funcionales.

- El módulo de horario debe tener en su cabecero un botón donde muestre un modal con el significado de cada una de las abreviaturas de los edificios de la universidad.
- El módulo de horario al momento de presionar el botón de *Zoom* para entrar a las salas virtuales, debe abrir un modal el cuál tenga información relevante a el ID sala, el código de la sala y la contraseña. Además, el campo de código de la sala y contraseña deben contar con un botón para copiar estas credenciales al portapapeles.

3.5. Notas

3.5.1. Requerimientos Funcionales.

- El módulo de notas debe listar por medio de acordeones las materias matriculadas por el estudiante.
- El módulo de notas debe visualizar en cada materia listada información relevante como el código de la materia, el código, grupo, créditos, nombre del profesor, notas, descripción de las notas, porcentaje de notas, nota definitiva y acceso directo al simulador de notas de la materia seleccionada.
- Si la descripción de las notas es muy larga, se colapsará el texto hasta el ancho de píxeles sugerido por el equipo de UX/UI. Al presionar la descripción de la nota, el texto se expanderá mostrando todo su contenido.
- El módulo de notas en cada materia debe tener un botón para redirigir al simulador de notas de la materia seleccionada.
- Si se generó la definitiva de la materia seleccionada, se debe poder visualizar el valor de la definitiva sin necesidad de desplegar los acordeones que listan las materias.

3.5.2. Requerimientos No Funcionales.

- El modulo de notas debe abrir un sólo acordeón a la vez. Dicho acordeón al no estar desplegado, debe mostrar únicamente el nombre de la materia en su título/etiqueta. Mientras que al estar desplegado además de mostrar el nombre de la materia, debe mostrar su código (del mismo modo en su título/etiqueta).

- Si no se encuentran notas a la materia desplegada, se debe mostrar un mensaje indicando que aún no hay notas registradas.
- Si no se encuentran profesores en la materia desplegada, se debe mostrar un mensaje indicando que aún no hay profesor asignado.

3.6. Simulador de Notas

El módulo de simulador de notas se divide en simulador padre y simulador hijo. El simulador padre es la pantalla donde se encuentran los accesos a cada simulador individual de las materias matriculadas y donde se hacen los cálculos del promedio ponderado con base a las notas que se asignen en cada simulador hijo. Por otro lado, el simulador hijo es el simulador especifico de cada materia.

3.6.1. Vista principal.

3.6.1.1. Requerimientos funcionales. :

- La vista principal del módulo de simulador de notas debe cargar las materias que el estudiante tiene matriculas (únicamente materias con calificación cuantitativa).
- La vista principal del módulo de simulador de notas padre debe calcular automáticamente el promedio ponderado respecto a las notas del simulador hijo.
- La vista principal del módulo de simulador de notas padre debe tener un pie de página que debe estar siempre fijo. Este pie de página contiene el valor del promedio ponderado teniendo

en cuenta los valores de los simuladores hijos. Además, debe actualizarse a tiempo real. Por otra parte, el contenido superior de dicho pie de página, debe permitir hacer desplazamiento (*scroll*) vertical.

3.6.1.2. Requerimientos no funcionales.

- Los valores de las notas deben tener máximo un dígito como mínimo.

3.6.2. Vista secundaria.

3.6.2.1. Requerimientos funcionales.

- El módulo de simulador de notas debe al inicializarse por primera vez, consultar si hay notas en el sistema para la materia seleccionada. De ser así, debe cargarlas automáticamente. De lo contrario, mostrar el simulador en su estado vacío.
- El módulo de simulador de notas debe calcular la definitiva automáticamente mientras tenga notas y porcentajes válidos (En otras palabras, que los campos del formulario estén diligenciados correctamente). En caso de algún error o inconsistencia, se le avisará al usuario el error especifico.
- El módulo de simulador de notas debe poder agregar y eliminar conjuntos de nota y porcentaje.

- El módulo de simulador de notas debe mostrar en todo momento el valor de la sumatoria de los porcentajes de las notas a tiempo real.

- El módulo de simulador de notas debe mostrar error en el formulario si al ingresar un nuevo porcentaje, éste supera el 100 %. Mientras esto pase, no se podrá agregar más notas hasta que el estudiante corrija el campo. Si varios campos del input de porcentaje hacen que se supere el 100 %, hasta que la sumatoria de todos los porcentajes sea menor o igual a 100, el formulario debe mostrar error.

- El módulo de simulador de notas no debe permitir ingresar caracteres especiales, así como en el input de nota solo se aceptan valores del 0 al 5, mientras que en el input de porcentaje se aceptan valores del 1 al 100. Tanto el input de nota como de porcentaje solo se acepta máximo un decimal.

- El módulo de simulador de notas debe tener un input donde se alojará la definitiva que se calcula constantemente de manera automática. Si se modifica este input, se borrarán todas las notas simuladas dejando el simulador en estado vacío, la cual de inmediato se llenará el input de notas con el valor modificado anteriormente y el input de porcentaje con el valor de 100.

- El módulo de simulador de notas debe tener un botón de restablecer. Dicho botón al ser presionado, inicializará el simulador. De tal manera que si hay notas cargadas en el sistema, las cargará. De lo contrario, el simulador quedará en su estado vacío.

- El input de la definitiva y el botón de restablecer siempre deben estar a fijos en la pantalla por medio de un pie de página. El contenido que queda en la parte superior de dicho pie de página, debe poderse hacer desplazamiento *scroll* de manera vertical.

- El módulo de simulador de notas debe informar al estudiante si hubo un cambio en sus notas, si se agregó notas nuevas o si se eliminó alguna nota del sistema de estudiantes por medio de un modal. Esto con respecto a la última vez que inicializó el simulador.

- El módulo de simulador de notas debe detectar cuando un campo está vacío. Avisándole así al estudiante que hay un error en el formulario.

- El módulo de simulador de notas debe guardar en el *Local Storage* todas las notas y definitivas generadas por los simuladores hijos a tiempo real.

3.6.2.2. Requerimientos no funcionales.

- Si se agrega en el input de nota un valor entero entre *6* y *9*, el input debe transformar dicho valor en un número decimal. Por ejemplo, si se ingresa el número *6*, este se convierte a *0.6*, y así sucesivamente.

- Si se intenta colocar en el input de nota un valor con dos decimales, éste campo se va a convertir en su estado vacío inicial.

- Si en el input de nota se coloca únicamente un valor entero, al salir del input debe formatear ese entero a un valor decimal, Por ejemplo, si se ingresa el valor de *4*, al salir del input este

campo debe visualizarse como *4.0*.

- El cabecero de la vista secundaria del simulador debe mostrar el código de la materia.

3.7. Histórico semestral

El módulo del histórico semestral se compone de una vista principal y una secundaria. El módulo padre es el encargado de agrupar y listar los datos relevantes de los semestres cursados por el estudiante. Mientras que el módulo hijo se encarga de listar las materias vistas, el promedio acumulado, los créditos vistos y los créditos aprobados del semestre seleccionado.

3.7.1. Vista principal.

3.7.1.1. Requerimientos funcionales.

- La vista principal del modulo del histórico semestral debe mostrar listados los semestres cursados por el estudiante de forma descendente. Cada elemento de la lista debe contener el nivel del semestre, el año, periodo académico y el promedio.
- La vista principal del modulo del histórico semestral debe tener un pie de página siempre visible en la pantalla el cual debe mostrar el promedio acumulado y los créditos acumulados.
- El pie de página de la vista principal debe estar siempre fijo. El contenido superior de dicho pie de página, debe permitir hacer desplazamiento (*scroll*) vertical.

3.7.1.2. Requerimientos no funcionales.

- Los valores de los promedios de cada semestre deben mostrarse con 1 decimal, mientras que el promedio acumulado debe mostrarse con 2 decimales.

3.7.2. Vista secundaria.

3.7.2.1. Requerimientos funcionales.

- La vista secundaria del módulo de notas debe listar por medio de acordeones las materias matriculadas por el estudiante.
- La vista secundaria del módulo de notas debe visualizar en cada materia listada información relevante como el código de la materia, el código, grupo, créditos, nombre del profesor, notas, descripción de las notas, porcentaje de notas, nota definitiva y acceso directo al simulador de notas de la materia seleccionada.
- Si la descripción de las notas de la vista secundaria es muy larga, se colapsará el texto hasta el ancho de píxeles sugerido por el equipo de UX/UI. Al presionar la descripción de la nota, el texto se extenderá mostrando todo su contenido.

3.7.2.2. Requerimientos no funcionales.

- El cabecero del módulo debe mostrar el periodo académico del cuál se está presentando la información.

4. Diseño

Teniendo en cuenta que la aplicación debe ser altamente robusta y ser capaz de integrarse con otros servicios de la universidad, la aplicación sigue una arquitectura orientada a servicios (SOA, por sus siglas en inglés). Este un enfoque para el desarrollo de sistemas de software que se basa en la idea de que los componentes de un sistema deben ser diseñados como servicios independientes y autónomos que puedan comunicarse entre sí a través de interfaces bien definidas.

En una arquitectura SOA, los servicios son unidades lógicas de funcionalidad que pueden ser invocados por otros servicios o por aplicaciones cliente. Cada servicio tiene una interfaz claramente definida que especifica los mensajes que puede recibir y los mensajes que puede enviar. Estos mensajes se envían mediante el protocolo REST.

Los servicios anteriormente mencionados, que se comprenden como el *backend*, se comunican con el dispositivo del cliente mediante una API expuesta a Internet, en este dispositivo se encuentra el *frontend*, que maneja la interfaz de usuario, el renderizado y las interacciones del usuario.

4.1. Backend

4.1.1. Componentes de la API. La API principal de la aplicación, a la cual se conecta el *frontend*, se compone de un *backend* principal que también actúa como intermediario

(*middleware*) entre otros *backends* y servicios que usa la aplicación. Este *backend* está conectado a la base de datos principal de la universidad.

Dado que gran parte de los datos de los estudiantes se encontraban en otra base de datos, se hizo uso extensivo de un *backend* secundario conectado a este. Este *backend* secundario se encarga de recuperar los datos necesarios de la base de datos correspondiente y enviarlos al *backend* principal mediante REST.

El *backend* principal se encarga de servir todos los módulos del proyecto. En los casos en que la información está almacenada en una base de datos diferente o requiere del empleo de un servicio transversal, este se comunica mediante un cliente HTTP.

Lo mencionado anteriormente se puede observar en la figura 5.

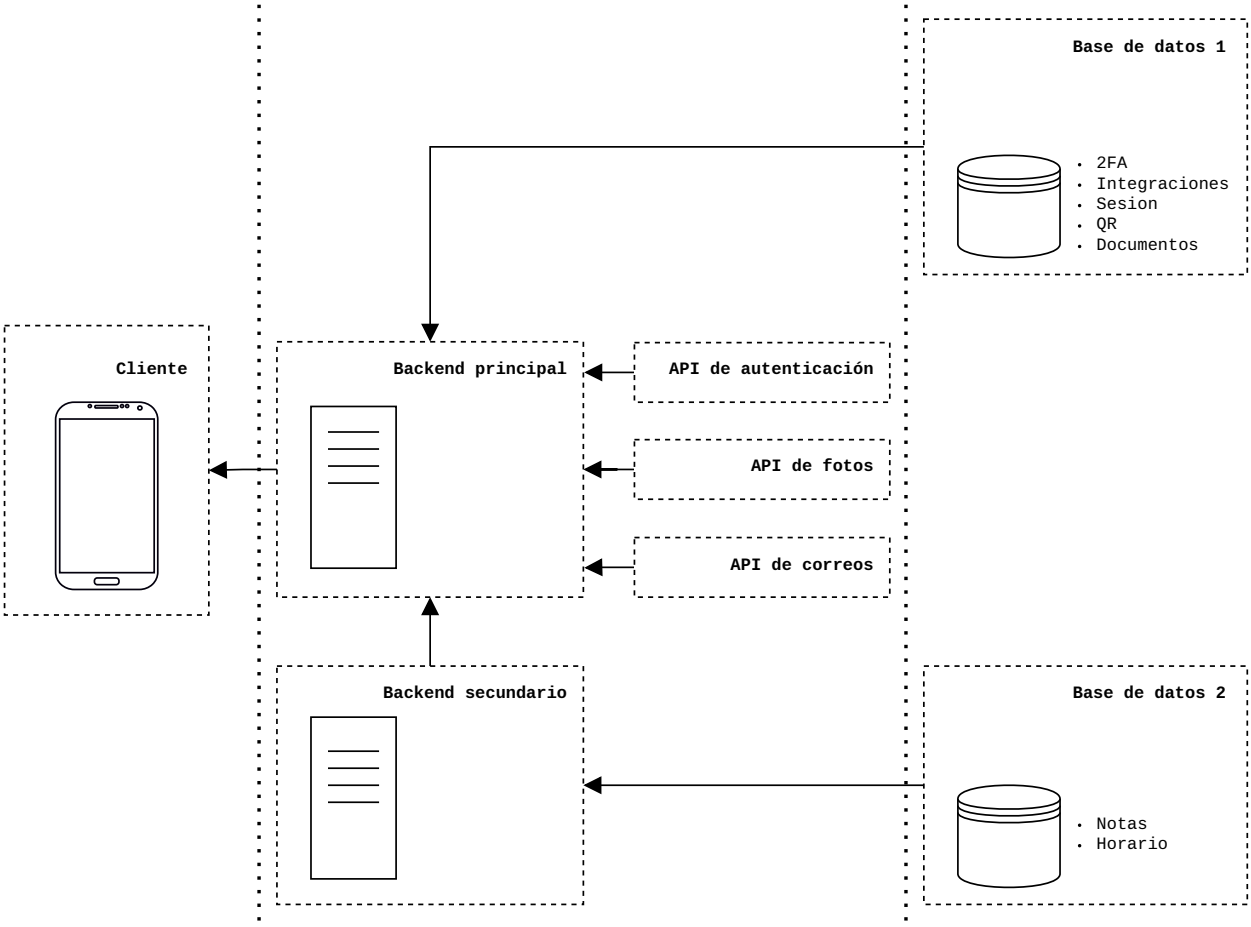


Figura 5. Arquitectura general de la API.

4.1.2. Componentes del backend. En cuanto a la arquitectura, esta está basada en varios componentes interconectados. En primer lugar, se crearon diferentes controladores que se encargan de recibir y enviar las solicitudes REST a través de la red. Estos controladores son la interfaz principal entre la aplicación móvil y el *backend*.

Además de los controladores, se crearon varios servicios internos encargados de realizar la lógica del negocio y manejar los datos. Estos servicios son responsables de procesar las solicitudes

recibidas por los controladores y generar las respuestas correspondientes. También se utilizan para realizar operaciones de validación, autenticación, autorización y cualquier otra lógica específica de la aplicación.

Por otro lado, los repositorios son los componentes encargados de interactuar con la base de datos mediante *Hibernate*. Estos repositorios se conectan a los modelos, que definen las tablas de SQL, para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) en la base de datos. Los modelos son la representación en código de las tablas de la base de datos, y se utilizan para mapear los datos almacenados en la base de datos a objetos en memoria.

Además de estos componentes principales, también se utilizaron otros componentes auxiliares en el diseño del *backend*. Los DTO (*Data Transfer Objects*) son estructuras de datos que se utilizan para definir la estructura de los JSON recibidos y enviados por los controladores. Los *mappers* son componentes que se encargan de convertir los modelos a DTOs, y viceversa.

Por último, los clientes son componentes que se encargan de realizar llamadas a otras APIs en caso de que la aplicación necesite interactuar con otros *backends*. Estos clientes son útiles para abstraer la complejidad de las llamadas a otras APIs.

Lo mencionado anteriormente se puede observar en la figura 6.

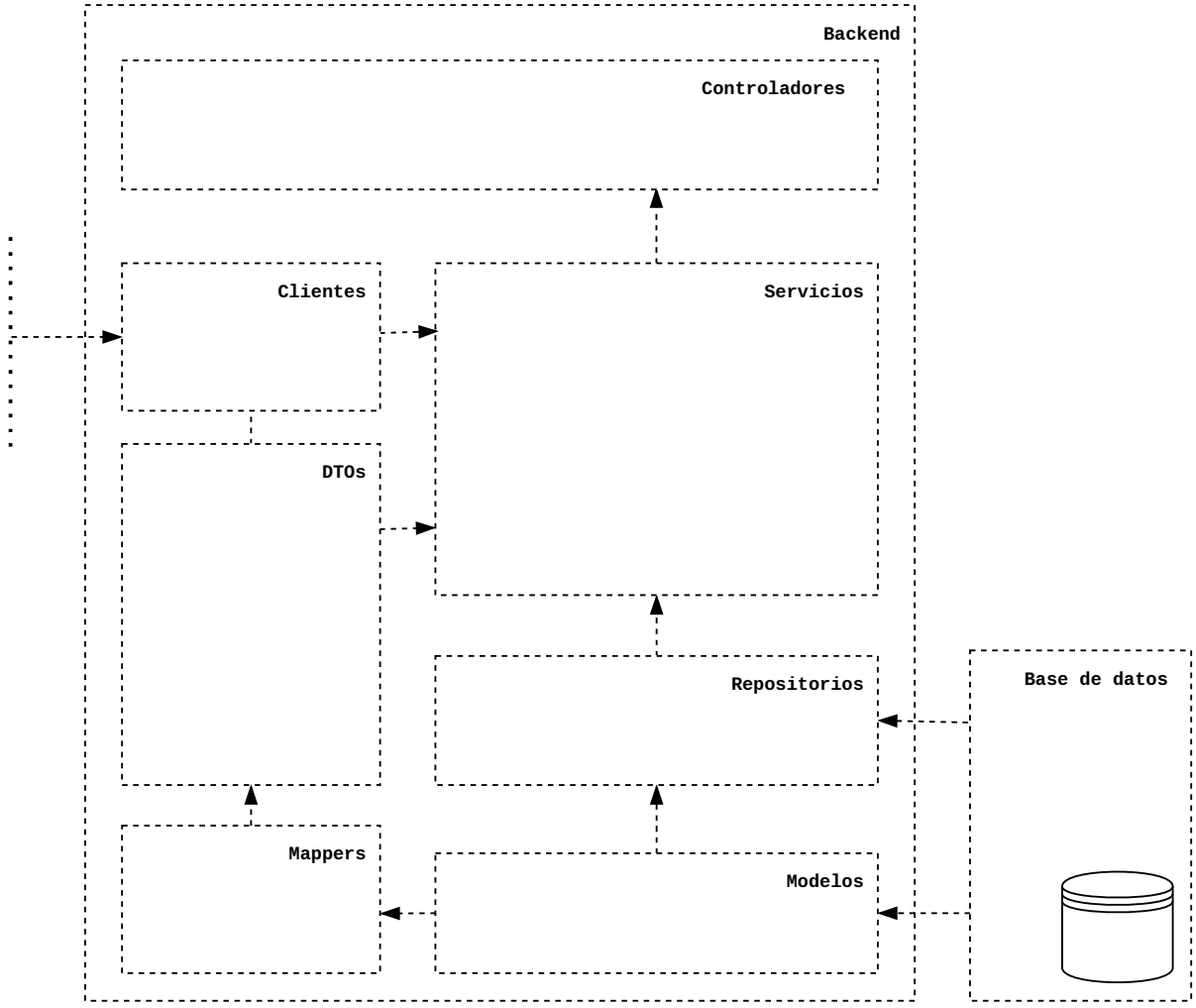


Figura 6. Arquitectura general de los *Backend*.

4.2. Frontend

4.2.1. Arquitectura de alto nivel. La arquitectura del *frontend* se divide en cuatro bloques principales, como se ilustra en la Figura 7. La interfaz de usuario se crea con HTML, CSS y *JavaScript*. Se utiliza un *framework* web para desarrollar los componentes de las vistas y

gestionar la navegación. A su vez, se utilizan los componentes específicos para dispositivos móviles que incluye el *framework* multiplataforma, que se pueden personalizar fácilmente. Además, el *framework* multipaltaforma proporciona una capa de conexión con las APIs nativas del dispositivo para personalizar la aplicación y aprovechar sus funcionalidades específicas.

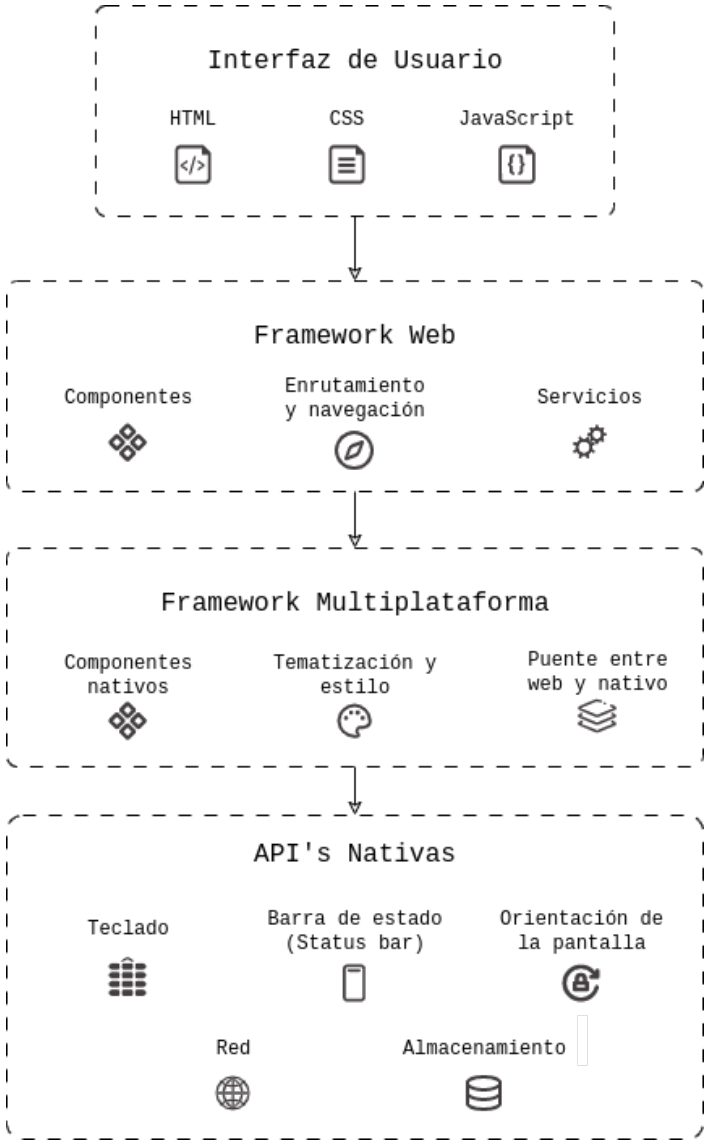


Figura 7. Arquitectura en alto nivel del *Frontend*.

4.2.2. Componentes del frontend. El *frontend* del proyecto se diseñó utilizando una arquitectura modular interconectada, como se ilustra en la Figura 8. El primer módulo que se carga es el módulo *App*, el cual importa todos los módulos necesarios para el correcto funcionamiento de la aplicación. Entre estos módulos se encuentra *Shared*, el cual contiene componentes compartidos utilizados en diferentes partes de la aplicación, como la cabecera (*header*), los modales, entre otros. También se encuentra el módulo *Servicios*, donde se define la lógica para consumir los *endpoints* del *backend*, almacenar datos de la aplicación y realizar otras tareas. Además, se utiliza el *Router* para redirigir al módulo de *Autenticación* o al módulo de *Home*, según sea necesario.

El módulo de *Home* tiene su propio *Router* y se encarga de redirigir a otros módulos dentro de él. El módulo predeterminado que se carga en *Home* es el de *Carnet Virtual*.

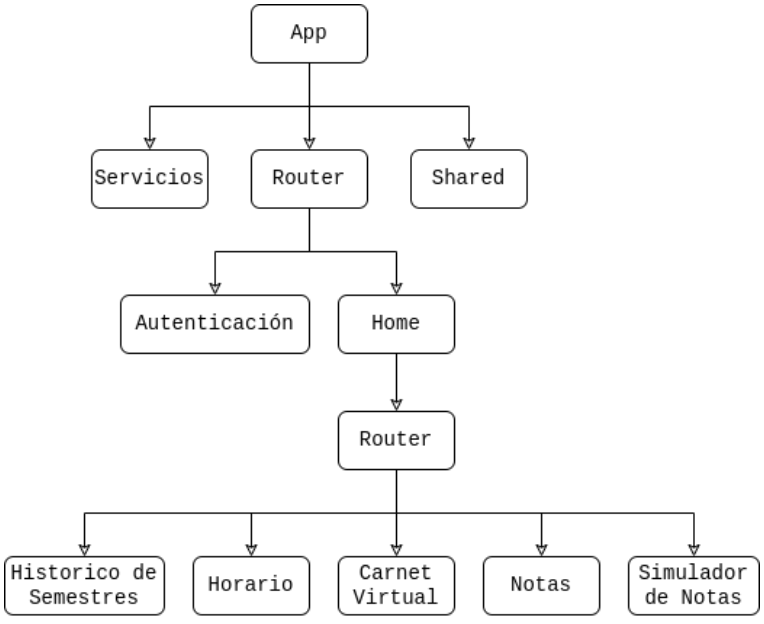


Figura 8. Interconexión de módulos del *Frontend*.

5. Metodología

El diseño y desarrollo de la aplicación fue realizado bajo la metodología *SCRUM* por un equipo dedicado al proyecto, Algunos componentes involucrados en la realización del proyecto, tales como algunos apartados visuales, el liderazgo del equipo y algunos servicios, fueron asignados por el proyecto RSI de la Universidad Industrial de Santander. El enfoque general de dicha metodología fue la entrega continua de valor.

El equipo de desarrollo trabajó en ciclos cortos llamados *sprints*, con una duración de 4 semanas. Cada *sprint* comenzó con una reunión de planificación, donde se seleccionaron las tareas más importantes para el *sprint* y se organizaron como historias de usuario.

Durante el *sprint*, el equipo se reunió periódicamente en una reunión corta de seguimiento en donde cada miembro del equipo informó sobre lo que se hizo, lo que se planea hacer y si hay algún obstáculo que lo impide. Esto ayudó al equipo a mantenerse en la misma página y a resolver rápidamente cualquier problema que surja. Al final de cada *sprint*, el equipo realizó una revisión del trabajo completado para recibir retroalimentación. Además, se llevó a cabo una retrospectiva del *sprint* para analizar lo que funcionó bien y lo que se puede mejorar en el próximo *sprint*.

Para gestionar el trabajo en nuestro proyecto, utilizamos la herramienta de manejo *Taiga*, que nos permitió visualizar el estado actual de cada tarea y asegurarnos de que ninguna tarea se quedara varada.

6. Implementación

A continuación se detalla todo el proceso de desarrollo que se llevó a cabo para crear la aplicación propuesta. La cual, además de ser una aplicación multiplataforma, su funcionalidad como un carnet virtual implicó desarrollar una integración los torniquetes de entrada/salida y desarrollar el marco para permitir acceso a otros servicios de la universidad en el futuro, como biblioteca o comedores.

6.1. Backend

El *backend* desarrollado para la aplicación fue escrito en *Java*, con el *framework* de *Spring Boot* bajo la arquitectura mencionada en el marco teórico, este framework fue escogido por su carácter empresarial, comunidad activa, robustez y alta integración con otras librerías.

En este sentido, y teniendo en cuenta las necesidades de la aplicación, el backend se dividió en 8 servicios principales, su nombre y funciones principales se mencionan a continuación.

6.1.1. Servicio de Notas. Este es el servicio encargado de ofrecer la información relacionada a las calificaciones de asignatura y semestre de un estudiante, para ello se usa un cliente de Feign (librería de Java) para solicitar datos a un *backend* secundario, este servicio cuenta con seguridad basada en roles (RBAC), lo que significa que solo es accesible para aquellos usuarios que cuenten con el rol de estudiante.

6.1.2. Servicio de Horario. Este es el servicio que ofrece la información relacionada a las clases programadas de un estudiante, de manera similar al servicio de notas, se conecta a un *backend* secundario mediante Feign y cuenta con seguridad basada en roles.

6.1.3. Servicio de QR. Este es el servicio encargado de la verificación y auditorías de códigos QR que funcionan como un método de acceso a los servicios de la universidad.

6.1.4. Servicio de Autenticación. Este es un servicio que se conecta al *backend* de autenticación del proyecto RSI, y es el que permite identificar los usuarios presentes en la base de datos de la universidad, del mismo modo, este es el que envía las acciones que un usuario puede realizar en el menú de la aplicación.

6.1.5. Servicio de Sesión. Este es un servicio encargado de manejar las sesiones de los usuarios de la aplicación, este evita que exista más de una sesión por usuario y las invalida de ser necesario, para esto el servicio asigna y almacena identificadores de los dispositivos y verifica si estos cambian.

6.1.6. Servicio de Documentos de Identificación. Este es un servicio encargado del manejo de los usuarios y los documentos de identificación asociados a estos, del mismo modo permite obtener la información personal de cada usuario para otros servicios.

6.1.7. Servicio de Autenticación de dos Factores (2FA). Este es el servicio que maneja el emparejamiento con dispositivos mediante la autenticación de dos factores, este trabaja

con los servicios de sesión y la API de mensajes presente en el proyecto RSI para verificar a aquellos usuarios que cambien de dispositivo mediante códigos enviados a sus correos.

6.1.8. Servicio de Fotos. Este es el servicio que maneja la foto mostrada en la pantalla principal de la aplicación, debido a las dificultades asociadas a la recuperación de las fotos de una persona (que pueden estar distribuidas entre varias bases de datos), se creó este servicio como una manera de abstraer dicha complejidad.

6.2. Frontend

6.2.1. Generalidades. Antes de explicar cada módulo de la aplicación, es importante mencionar algunas funcionalidades que se aplican a varios módulos pero que no son específicas de uno en particular.

6.2.1.1. Funcionalidad sin conexión a Internet. Todos los módulos de la aplicación, excepto el módulo de *Autenticación*, pueden funcionar sin conexión a Internet. Se ha implementado el patrón de diseño *Repository* para tener dos fuentes de datos: el servidor (base de datos remota) y el almacenamiento del dispositivo (base de datos local). Esto se ilustra en la Figura 9.

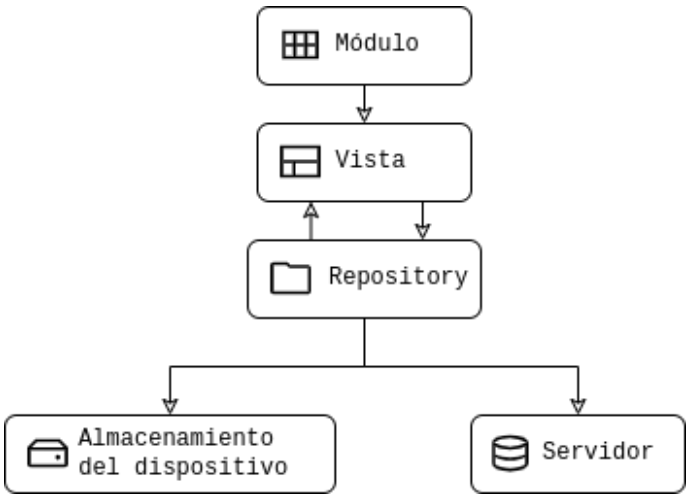


Figura 9. Diagrama del patrón *Repository*.

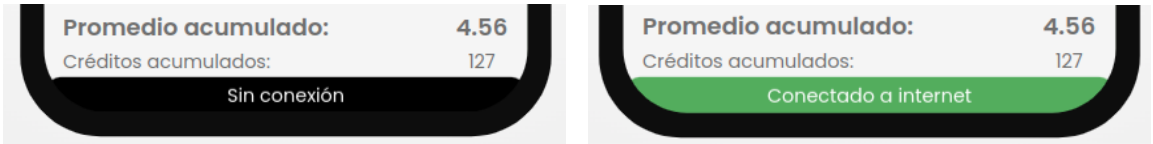
El flujo de esta funcionalidad en la aplicación es el siguiente:

- La primera vez que el usuario accede a la aplicación, no cuenta con datos almacenados, por lo que es necesario consultarlos en el *backend* (requiere conexión a Internet).
- Una vez que se obtiene algún dato, se almacena en el dispositivo. Lo anterior permite que esos datos que el usuario ya haya cargado o obtenido, estén disponibles cuando no tenga conexión a Internet.
- Si el usuario ya ha accedido a un módulo, como por ejemplo el de *Horario*, y trata de hacerlo sin conexión a Internet, la aplicación cargue los datos almacenados para permitirle el acceso. Si el usuario intenta acceder a un módulo al que nunca ha ingresado sin tener conexión a Internet, aparece la pantalla que se ilustra en la Figura 11.
- Cuando el usuario recupera la conexión a Internet, se consultan los datos del *backend* para

determinar si se deben actualizar los datos almacenados en el dispositivo. Si los datos nuevos son idénticos a los almacenados localmente, no se actualiza nada. Si hay cambios, se actualizan tanto los datos almacenados como la vista en la que se encuentra el usuario.

- Cuando el usuario tiene acceso a Internet y datos almacenados, la aplicación muestra primero los datos locales mientras consulta los datos del *backend* en segundo plano. De esta manera, se garantiza que la interfaz se cargue más rápido y se tenga la información actualizada.

Por otra parte, se implementó un aviso para informar al usuario cuando no dispone de acceso a Internet. Este aviso aparece en la parte inferior de la pantalla, tal como se muestra en la Figura 10(a). Cuando el usuario recupera la conexión a Internet, se muestra un aviso durante aproximadamente dos segundos, como se ilustra en la Figura 10(b), para después desaparecer.



(a) Aviso de no conexión a Internet (b) Aviso de recuperación de la conexión a Internet

Figura 10. Avisos de no conexión al usuario.

Figura 11. Pantalla cuando no hay Internet y no se tienen datos almacenados.

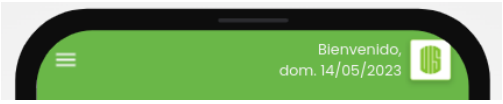
6.2.1.2. Estrategia de carga. La aplicación consta de dos módulos principales: *Autenticación* y *Home*. Estos módulos se cargan utilizando la técnica de carga diferida (*lazy loading*). Para optimizar el rendimiento, se implementó la carga diferida en los módulos hijos más pesados del módulo *Home*, que son *Histórico de Semestres* y *Simulador de notas*. Esto permite reducir el tiempo de carga del módulo *Home* y acelerar la carga inicial de los módulos *Carnet Virtual*, *Notas* y *Horario*. En la Figura 8 se visualiza la estructura de los módulos.

6.2.1.3. Internacionalización. La aplicación es compatible con dos idiomas: inglés y español. El idioma que se utiliza en la aplicación depende del idioma que el usuario haya configu-

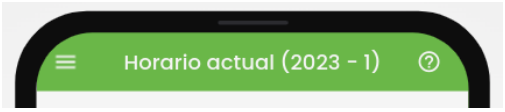
rado en su dispositivo móvil o navegador. Todos los mensajes en ambos idiomas se almacenan en bases de datos, lo que permite su modificación en cualquier momento y garantiza que los cambios se apliquen automáticamente a todos los usuarios.

6.2.1.4. Guardian de rutas. Cada vez que se accede a una ruta o módulo en la aplicación, se activa el *Guardian de Rutas*. Esta función permite controlar el acceso a determinadas áreas de la aplicación. De este modo, se restringe el acceso a los módulos de la aplicación para los usuarios no autenticados, y se impide el acceso al módulo de *Autenticación* para los usuarios ya autenticados. El *Guardian de Rutas* verifica la autenticidad del usuario mediante su token de autenticación.

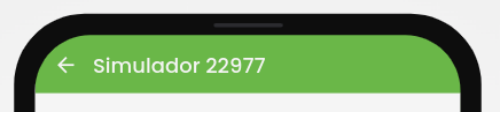
6.2.1.5. Cabecera de la aplicación (Header). Excepto por el módulo de autenticación, todos los demás módulos tienen una cabecera (*header*) que gestiona la navegación en la aplicación y muestra información como el título o las acciones del módulo en el que se encuentra el usuario. El diseño se basó en los lineamientos propuestos por Material 3.



(a) Cabecera del módulo de Carnet Virtual



(b) Cabecera del módulo de Horario



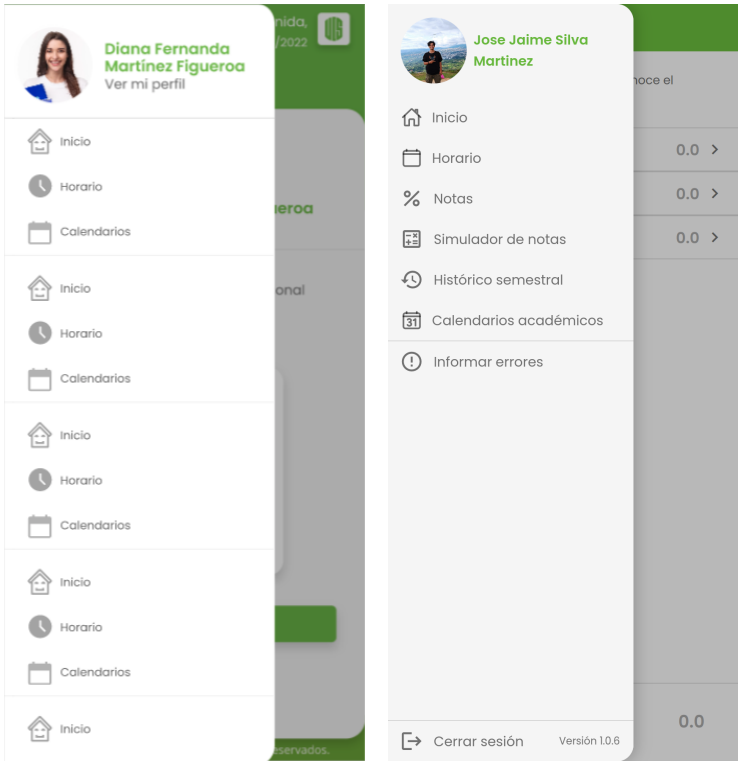
(c) Cabecera en vista secundaria del módulo de Simulador de Materias

Figura 12. Cabecera de la aplicación.

En la Figura 12 se muestra la cabecera de la aplicación, que varía según el módulo en el que se encuentre el usuario. Si se encuentra en una vista secundaria, en lugar del icono de menú hamburguesa (tres líneas horizontales), aparece una flecha para volver a la vista principal, como se ilustra en la Figura 12(c).

Cuando el usuario presiona el icono de menú hamburguesa o realiza un gesto de deslizar hacia la derecha en el borde izquierdo de su dispositivo móvil, se despliega un menú lateral, como se muestra en la Figura 13(b). A diferencia del diseño planteado en la Figura 13(a), se hicieron algunos ajustes para utilizar los iconos de material, e incluir dos apartados adicionales: *Informar errores*, que redirige a la mesa de ayuda de la Universidad Industrial de Santander, y *Cerrar sesión*, que permite al usuario cerrar la sesión y ver la versión de la aplicación.

Cabe mencionar que la funcionalidad *Ver mi perfil* no se implementó, ya que el módulo de Carnet Virtual cumple esa función.

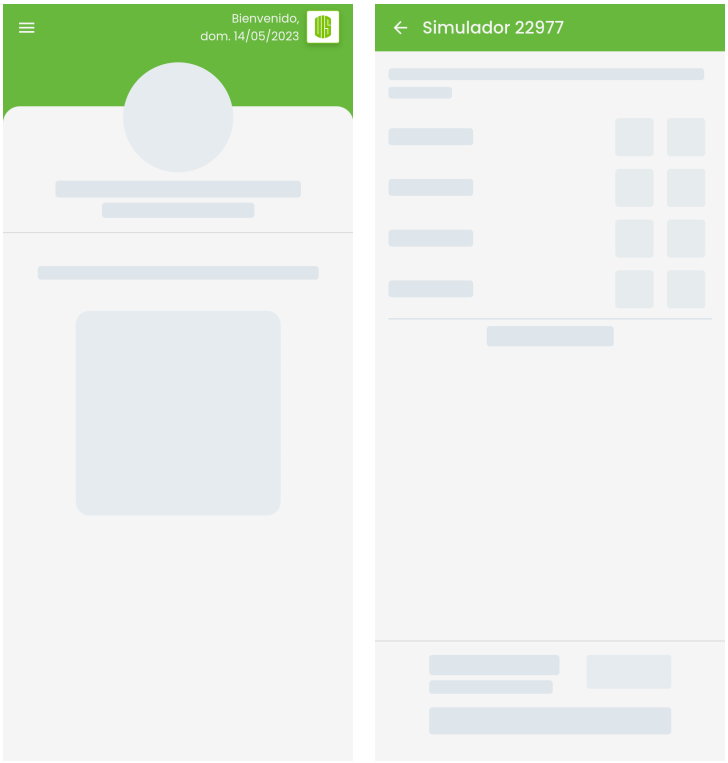


(a) Diseño (b) Implementación

Figura 13. Comparación entre diseño e implementación del menú lateral.

6.2.1.6. Estados de carga. Cada módulo de la aplicación cuenta con un estado de carga esqueleto, es decir, se muestra una estructura preliminar del contenido antes de cargar los datos reales. Este proceso se ilustra de manera más clara en la Figura 14. Durante la carga de los datos para mostrar la vista del módulo, se presenta el estado de carga esqueleto, y una vez que se determina si hay o no hay datos para mostrar, se cambia a la vista correspondiente. Esta estrategia se utiliza para hacer más ameno y visual el tiempo de espera para el usuario. Por otra parte, cada vez que se hace una petición al *backend*, se muestra una barra de carga en la parte superior de la

aplicación, tal como se ilustra en la Figura 15. Una vez que se resuelve la petición, ya sea con éxito o con error, la barra desaparece.



(a) Carga esqueleto en el módulo de *Carnet Virtual*.

(b) Carga esqueleto en el módulo de *Simulador de Notas*.

Figura 14. Ejemplos de carga esqueleto.

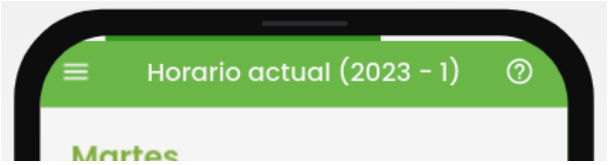


Figura 15. Barra de carga superior.

Adicionalmente, se implementó una funcionalidad común en las aplicaciones móviles, conocida como *pull to refresh*, que permite actualizar la vista actual realizando un gesto de deslizar hacia abajo en la parte superior de la misma. En la Figura 16 se ilustra como se ve a nivel de interfaz.

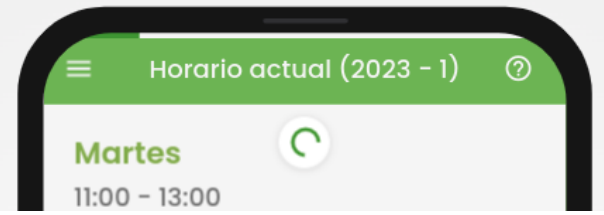
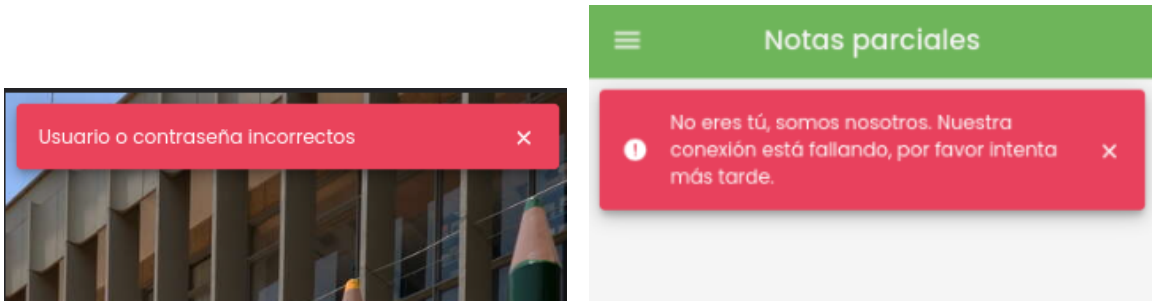


Figura 16. Funcionalidad de *pull to refresh*.

6.2.1.7. Barra de mensajes. La aplicación utiliza *snackbars*, que son pequeñas áreas rectangulares que aparecen en la pantalla para mostrar breves notificaciones o mensajes al usuario. En este caso, principalmente se utilizan para informar errores, como se ilustra en la Figura 17.



(a) *Snackbar* para usuario o contraseña incorrectos (b) *Snackbar* para error en el servidor

Figura 17. Ejemplos de *snackbars* en la aplicación.

6.2.1.8. Modales. En la aplicación, para mostrar contenido que requiere atención por parte del usuario, información u opciones adicionales, entre otras cosas, se utilizan modales. En la Figura 18 se muestra la estructura de estos. Todos los modales que se utilizan siguen los lineamientos descritos por Material 3 para los cuadros de dialogo.

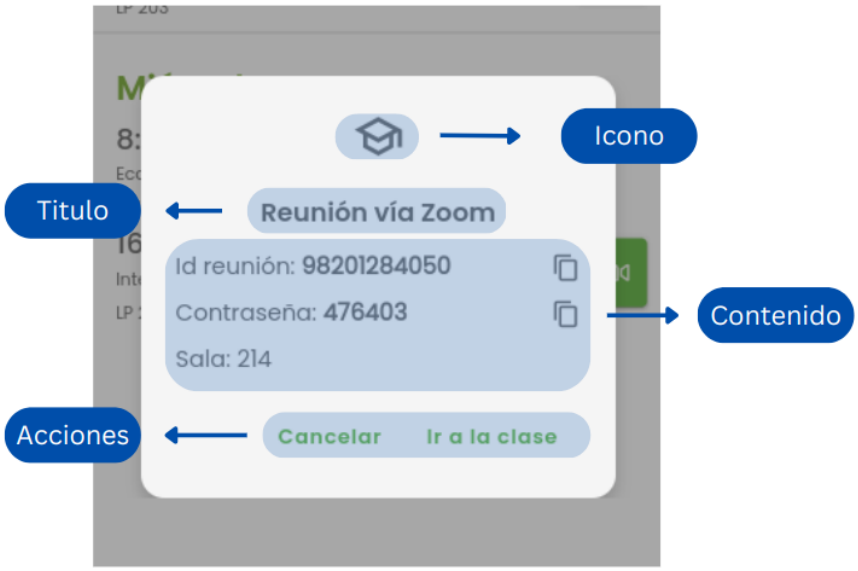
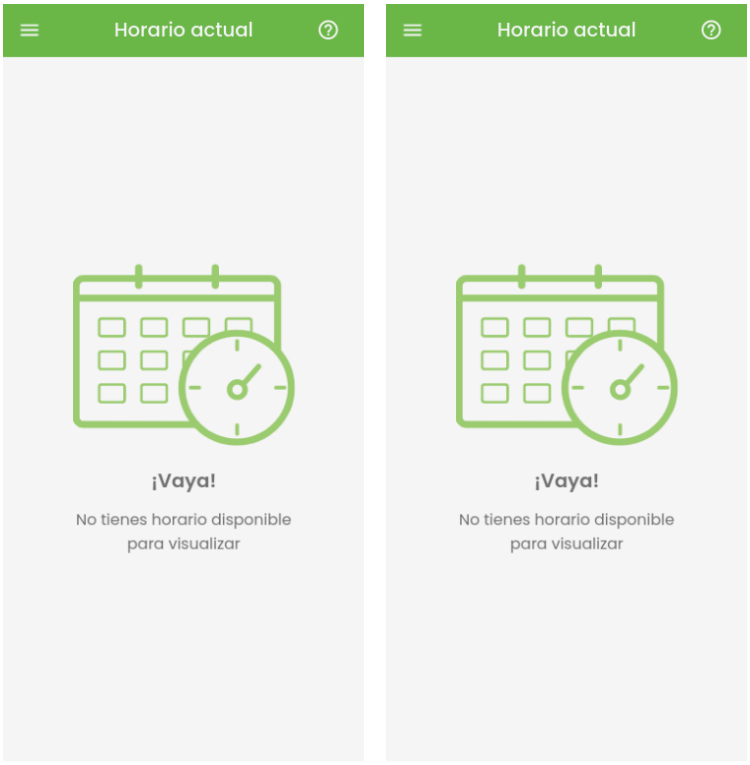


Figura 18. Estructura de un modal en la aplicación.

6.2.1.9. Estados vacíos. Para abordar los casos en los que no se dispone de información para mostrar en las vistas de los módulos, se han implementado vistas informativas que comunican al usuario de manera visual y agradable la razón por la cual no se pudo obtener la información. Esta funcionalidad se ha implementado en todos los módulos, a excepción del módulo de Autenticación. En la Figura 19 se presentan dos ejemplos de estados vacíos, uno correspondiente al módulo de Horario y otro al Simulador de Notas.



(a) Estado vacío para el Horario (b) Estado vacío para el Simulador

Figura 19. Estados vacíos para los módulos de *Horario* y *Simulador de Notas*.

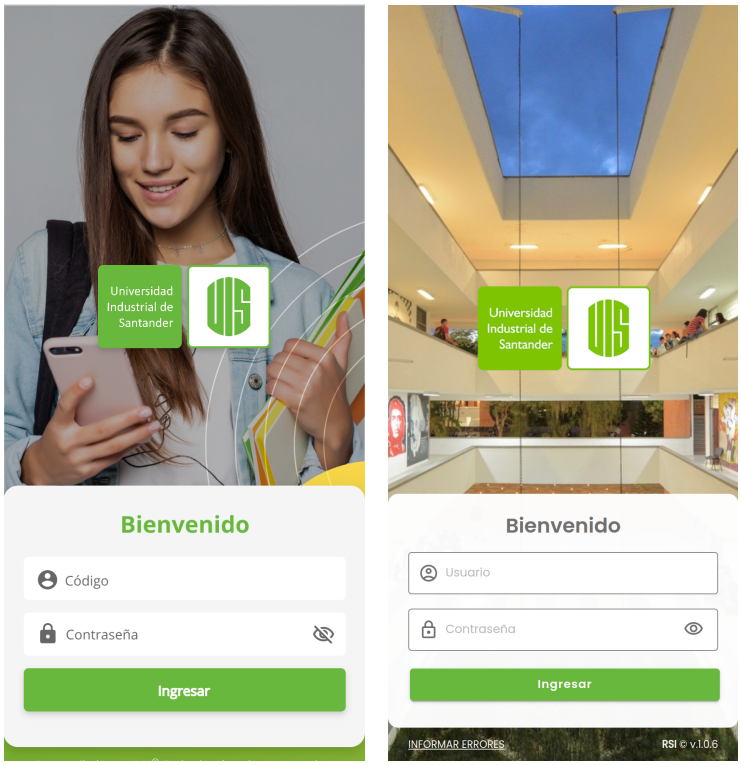
Algunas situaciones generales que conducen a la visualización de estas vistas incluyen la falta de datos proporcionados por el *backend*, como valores *null*, listas vacías ([]), entre otros. Asimismo, en casos poco frecuentes pero posibles, si el *backend* está inactivo, su respuesta será de estado 500 y se devolverá un valor *null*.

6.2.2. Autenticación. Al iniciar la aplicación, lo primero que encuentra el estudiante es el módulo de inicio de sesión, donde por medio de su código de estudiante y contraseña como credenciales se puede autenticar. Una vez autenticado, la sesión se vincula al dispositivo donde se esté ejecutando la aplicación (sea la versión móvil o desde el dispositivo donde se esté ejecutando la versión web). Esto se plantea de esta forma por cuestiones de seguridad, principalmente para mitigar la suplantación mediante la distribución indebida de credenciales y el ingreso al campus de personas ajenas a la comunidad estudiantil. Si se desea iniciar sesión en un dispositivo diferente al actual, y se detecta que ya pasaron 24 horas desde el último inicio de sesión, se despliega un sistema de autenticación de dos pasos. De lo contrario, el usuario tiene que seguir usando el dispositivo donde la sesión esté activa o esperar las 24 horas correspondientes. Este proceso de autenticación en dos pasos funciona enviando un código de 6 dígitos a los correos suministrados por el estudiante a la universidad (tanto correo personal, como correo institucional) y mediante un intervalo de tiempo determinado, el estudiante debe ingresar el código que recibió en sus respectivos correos y así poder iniciar sesión desde un dispositivo nuevo.

En la Figura 20 se presentan los diseños que se utilizaron para desarrollar este módulo. Durante su desarrollo, así como en el de otros módulos, se realizaron algunas modificaciones en el diseño original con el objetivo de mejorar la aplicación al máximo. Esto se puede observar en las diferencias entre la Figura 20(a) y la Figura 20(b). Algunas de estas diferencias son la imagen de fondo y el contorno de los campos de usuario y contraseña.

Para darle más personalidad a la aplicación, se agregaron imágenes aleatorias de diferentes

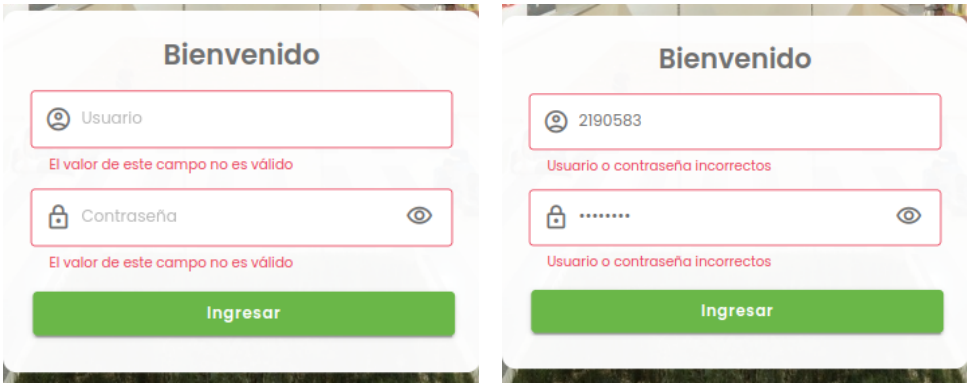
lugares de la universidad, cuidadosamente seleccionadas. Cada vez que un usuario accede a la pantalla de inicio de sesión, se selecciona una imagen aleatoria. Cabe mencionar que cada imagen se comprimió para lograr el mejor equilibrio entre calidad y peso y evitar problemas con su carga.



(a) Diseño (b) Implementación

Figura 20. Comparación entre diseño e implementación del módulo de *Autenticación*.

Se incorporó gestión de errores en los campos de inicio de sesión. Para realizar la petición de inicio de sesión se verifica que el usuario haya ingresado ambos campos requeridos, es decir, usuario y contraseña; en caso contrario, aparecen los errores mostrados en la Figura 21(a). Asimismo, si alguno de estos dos campos es incorrecto, se muestran los errores ilustrados en la Figura 21(b).



(a) Errores para campos vacíos o inválidos. (b) Errores para usuario o contraseña incorrectos.

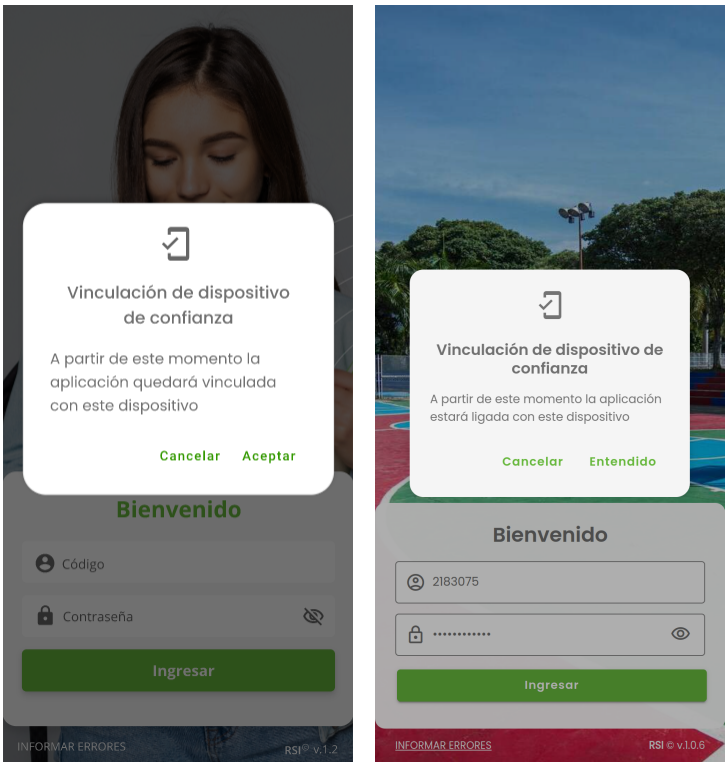
Figura 21. Manejo de errores en los campos de inicio de sesión.

Cabe destacar la posibilidad de visualizar la contraseña mediante un botón de activacíon/desactivacíon, es decir, el botón con el icono de un ojo ubicado dentro del campo de la contraseña.

Por otra parte, para el proceso de enlazar el dispositivo, que solo se hace una vez por usuario, se muestra un modal tal y como ilustra en la Figura 22(a). Si el usuario no acepta, no puede iniciar sesión. Si el usuario acepta, se realiza el proceso de inicio de sesión, en donde siempre se generan dos *tokens*:

- Token de autenticación: Es de tipo JWT, y codifica cierta información como el ID del usuario y un tiempo de expiración. A través de este se controla la duración de la sesión del usuario. Si el usuario cierra su sesión éste es eliminado.
- Token del dispositivo: Es una cadena de texto extensa que se genera desde el servidor y se

utiliza como identificador único del dispositivo. Por ende, se mantiene incluso si el usuario cierra sesión, y se almacena en un lugar que no sea fácilmente borrable por el usuario.

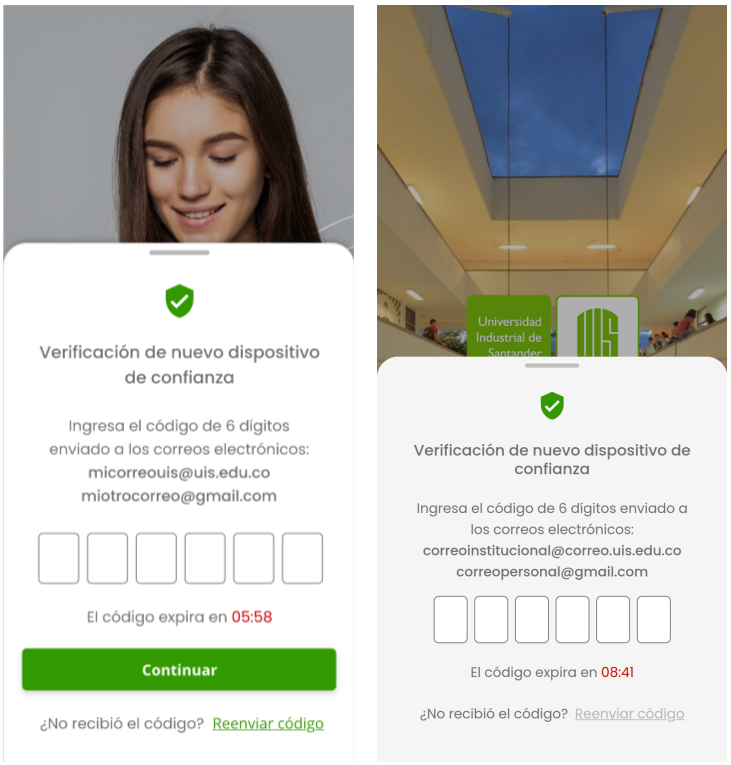


(a) Diseño (b) Implementación

Figura 22. Comparación entre diseño e implementación del modal de enlazar el dispositivo.

Si un usuario quiere cambiar de dispositivo, o perdió el token que identifica su dispositivo, debe realizar un proceso de autenticación en dos pasos para ingresar de nuevo a la aplicación. En la Figura 23(a) se presenta el diseño para el desarrollo de la autenticación en dos pasos. Como se observa en la Figura 23(b) hay varias diferencias. Por un lado, se eliminó el botón *Continuar* ya que se decidió mejorar la experiencia del usuario al remover una interacción adicional al tener que presionar el botón. De esta manera, una vez el usuario entra los 6 dígitos, se realiza automática-

mente la validación. Por otro lado, se cambió el color de fondo de la hoja inferior emergente o *bottom sheet*, para mayor contraste.



(a) Diseño (b) Implementación

Figura 23. Comparación entre diseño e implementación de la autenticación en dos pasos del módulo de *Autenticación*.

Se limitó la cantidad de veces que un usuario puede reenviar el código del 2FA (autenticación en dos pasos) para evitar posibles malas intenciones con sobrecargas en el *backend*.

(a) La cuenta regresiva finalizó y el código ya expiró.

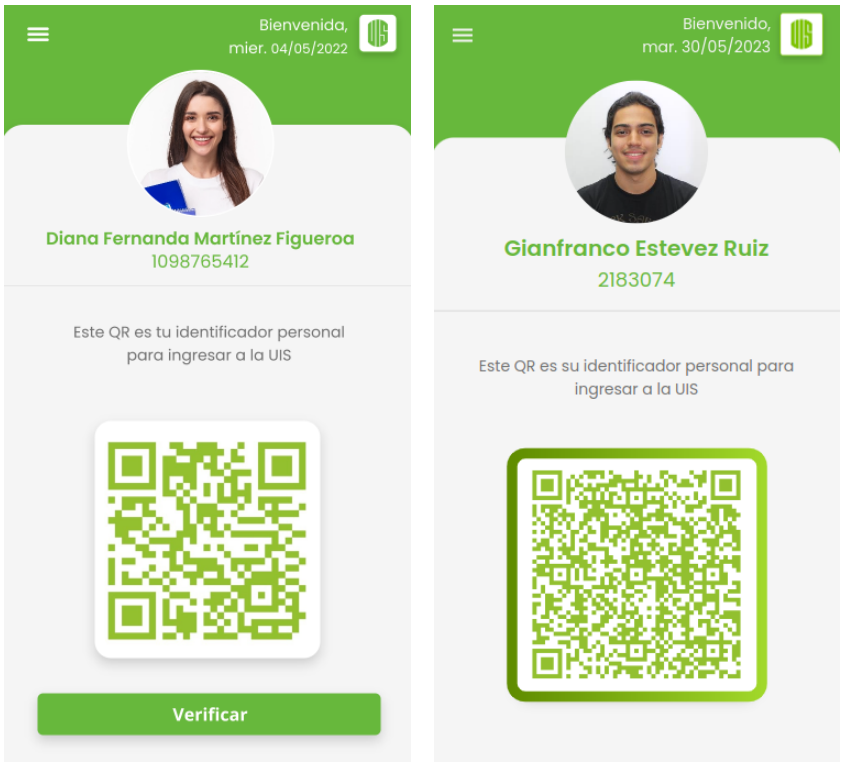
(b) El código ingresado no es válido.

Figura 24. Gestión de expiración y errores en la autenticación en dos pasos.

Cuando se acaba el tiempo y el código expira (Figura 24(a)), los campos de entrada se deshabilitan hasta que el usuario reenvíe otro código. En cuanto al funcionamiento de los campos de entrada para el código, se implementaron de tal manera que siguiese una funcionalidad similar al que suelen tener los campos de este tipo. Los campos solo aceptan valores numéricos, y cuando el usuario ingresa un dígito se cambia automáticamente a la siguiente casilla, y así hasta llegar a la última. Una vez se ingresa el valor de la última casilla, se valida en el servidor que el código sea válido, en caso contrario aparece el mensaje que se ilustra en la Figura 24(b) durante un tiempo de aproximadamente 2 segundos y se vacían los valores de las casillas.

6.2.3. Carnet Virtual. Una vez que se ha iniciado sesión en la aplicación, se redirige automáticamente al módulo del *Carnet Virtual*. El propósito de este módulo es recrear un carnet físico en formato digital. Incluye el nombre, la foto y un código QR para controlar el acceso del estudiante. Este código QR contiene información confidencial del estudiante y se utiliza como

mecanismo de acceso a los servicios integrados que la universidad requiere en esta aplicación.



(a) Diseño (b) Implementación

Figura 25. Comparación entre diseño e implementación para el módulo de *Carnet Virtual*.

Para este módulo, se ha implementado una barra de carga alrededor del código QR. Esto permite verificar de manera visual que se encuentra dentro de la aplicación y no en una captura de pantalla, evitando posibles fraudes. Gracias a esta mejora, se eliminó la necesidad del botón *verificar* ilustrado en la Figura 25(a).

6.2.4. Horario. La aplicación permite la visualización del horario actual del estudiante. En este módulo el estudiante puede acceder a una pantalla donde encuentra datos relevantes

como lo son las horas de las clases, salón, grupo, código, credenciales y accesos directos a las salas de clases virtuales que hayan disponibles para cada materia. La información presentada anteriormente se ordenada por los días de clase y agrupados por las materias correspondientes como se ilustra en la Figura 26.

(a) Diseño (b) Implementación

Figura 26. Comparación entre diseño e implementación para pantalla principal del módulo de *Horario*.

Si es la primera vez que el usuario entra a este módulo, aparece el mensaje de la Figura 27, el cual puede ocultarse al presionar el botón de *Entendido*. Las clases que tengan credenciales para

acceder a las aulas virtuales tendrán un botón verde, que al presionarlo, muestra un modal con la información de acceso a la clase, como se ilustra en la Figura 28(b).

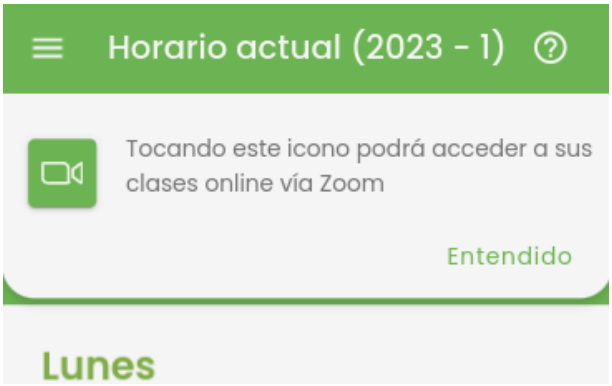
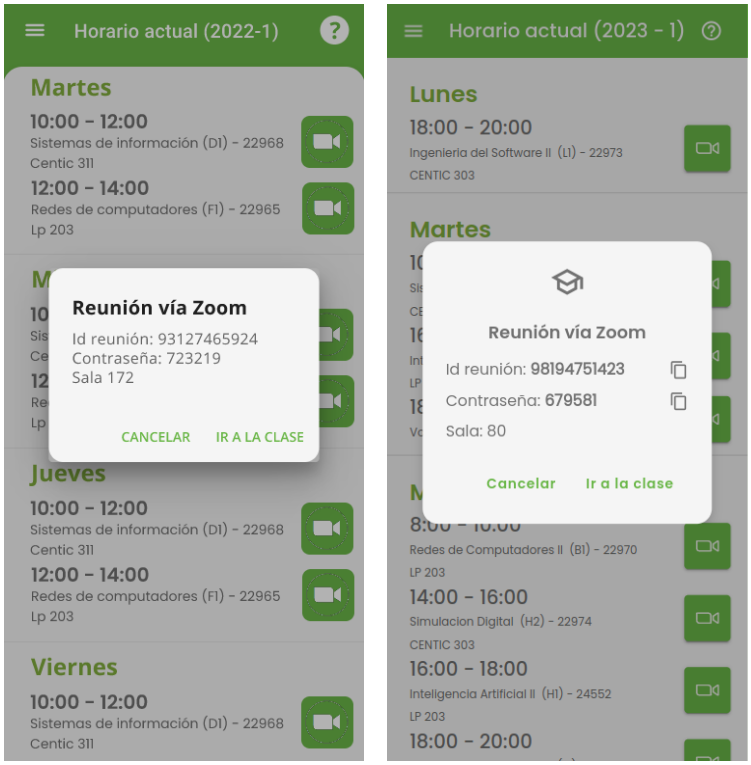


Figura 27. Mensaje de información del módulo de *Horario*.

Además, para organizar el horario de manera clara y concisa, se muestra en primer lugar el día actual. Por ejemplo, si hoy es martes, se coloca primero *Martes* en la lista, seguido de los días restantes de la semana en los que hay clases. Para indicar el cambio de semana, se utiliza una linea de color gris más intenso.

Por otra parte, cuando el usuario presiona el botón de *Ir a la clase*, se redirige a la sala en la plataforma *Zoom* donde se llevará a cabo la clase. Además, se agregó una funcionalidad para copiar las credenciales al portapapeles. Al presionar los iconos del lado derecho de las credenciales de *ID reunión* y *Contraseña* como se observa en la Figura 28(b), se copia al portapapeles el valor de la credencial seleccionada y aparece un mensaje en la parte inferior de la pantalla.

Por otro lado, si el usuario no conoce el significado de alguna abreviatura de un edificio en

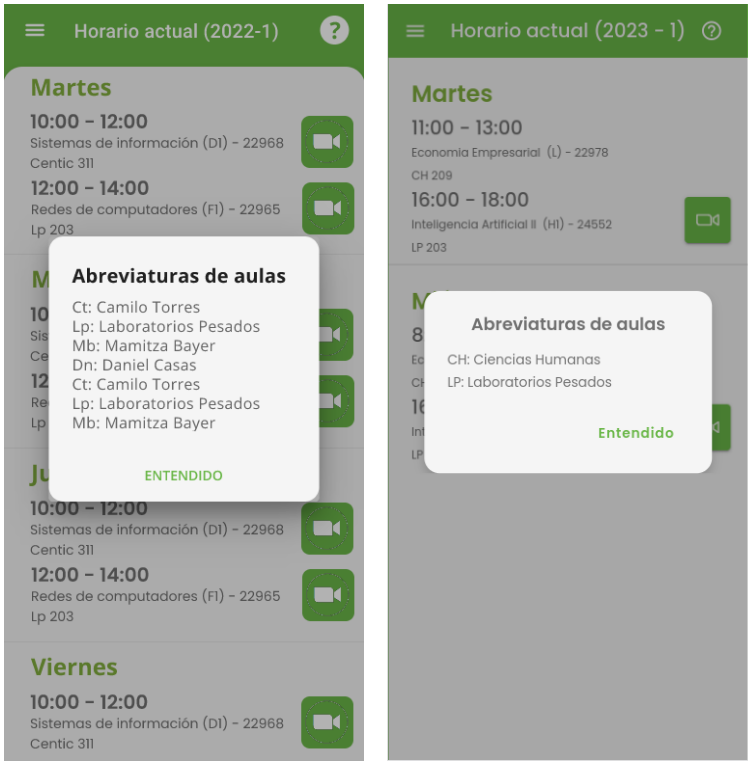


(a) Diseño (b) Implementación

Figura 28. Comparación entre diseño e implementación para el modal con información de la clase.

su horario, al presionar el botón con el signo de interrogación que se encuentra a la derecha de la cabecera, se muestra el modal que se ilustra en la Figura 33.

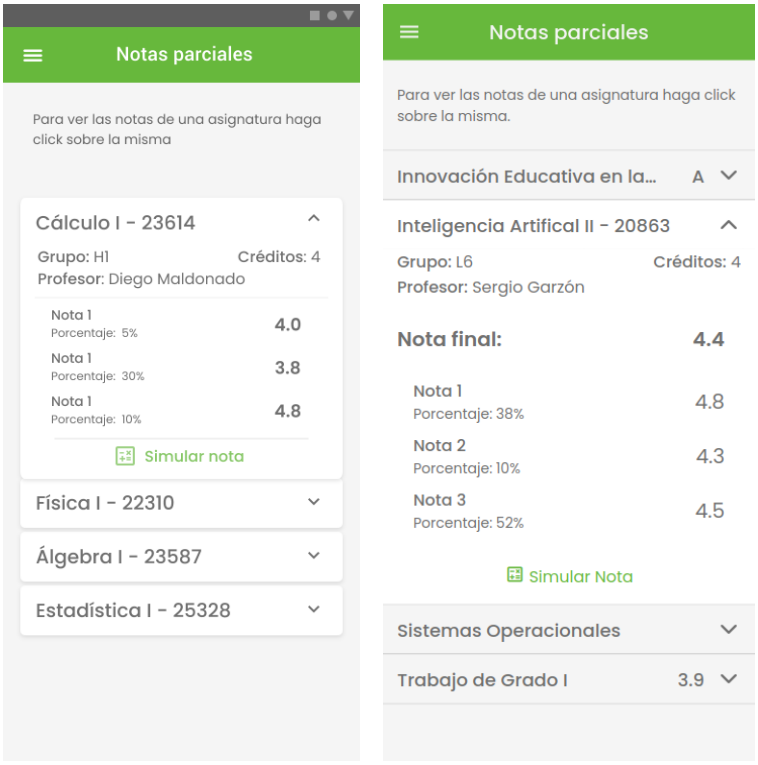
Inicialmente se diseñó que esta sección mostrara las siglas de todos los edificios, pero posteriormente se decidió mostrar únicamente las siglas relevantes para el usuario, es decir, aquellas que se encuentran en su horario. En caso de que el nombre del edificio sea idéntico a su sigla, como es el caso del CENTIC, este no se muestra en dicha sección.



(a) Diseño (b) Implementación

Figura 29. Comparación entre diseño e implementación para el modal con información de las abreviaturas de edificios.

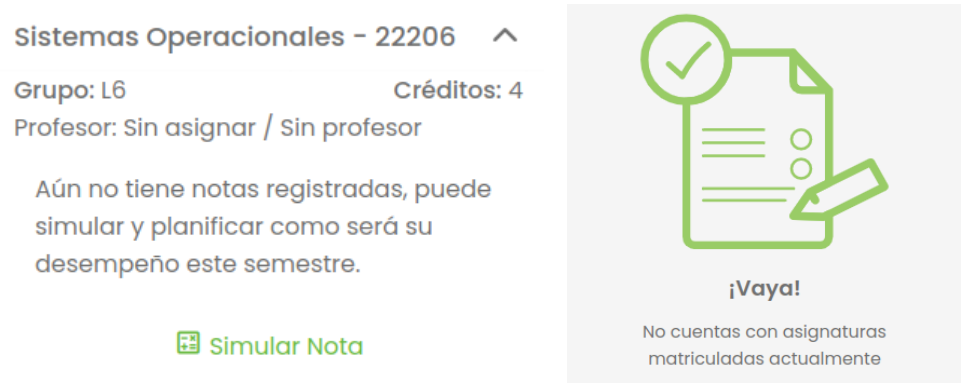
6.2.5. Notas del semestre. La aplicación permite la visualización de las notas del semestre. Estas notas se presentan por medio de acordeones y contienen información relevante como lo son el nombre de la materia, código, profesor, grupo, créditos, notas, nombre de las notas, porcentaje de las notas y valor de las mismas. Además, se encuentra un acceso directo al simulador de notas. Este acceso directo debe dirigir directamente al simulador de la materia seleccionada.



(a) Diseño (b) Implementación

Figura 30. Comparación entre diseño e implementación para el módulo de *Notas*.

Este módulo puede tener tres estados vacíos, como lo son cuando no se ha registrado alguna nota, cuando un profesor no está asignado y cuando no el estudiante no tiene materias matriculadas. Visualizándose se la siguiente manera:



(a) Estados vacíos de profesor y notas (b) Vista sin materias matriculadas

Figura 31. Comparación entre diseño e implementación para el módulo de *Notas*.

Al momento de generarse una definitiva, el usuario ahora visualiza un nuevo texto en el acordeón de la asignatura tanto expandido como colapsado con el valor de la definitiva sea cuantitativa o cualitativa.

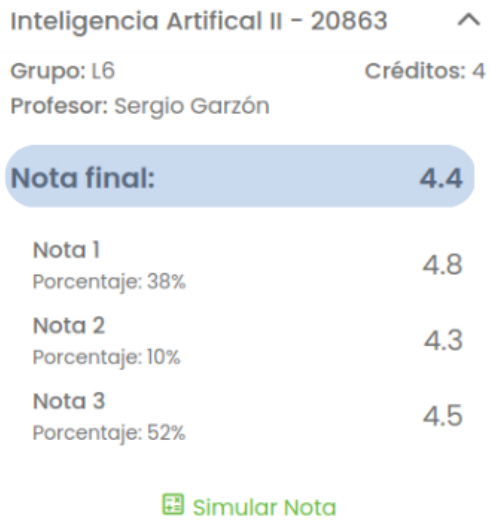
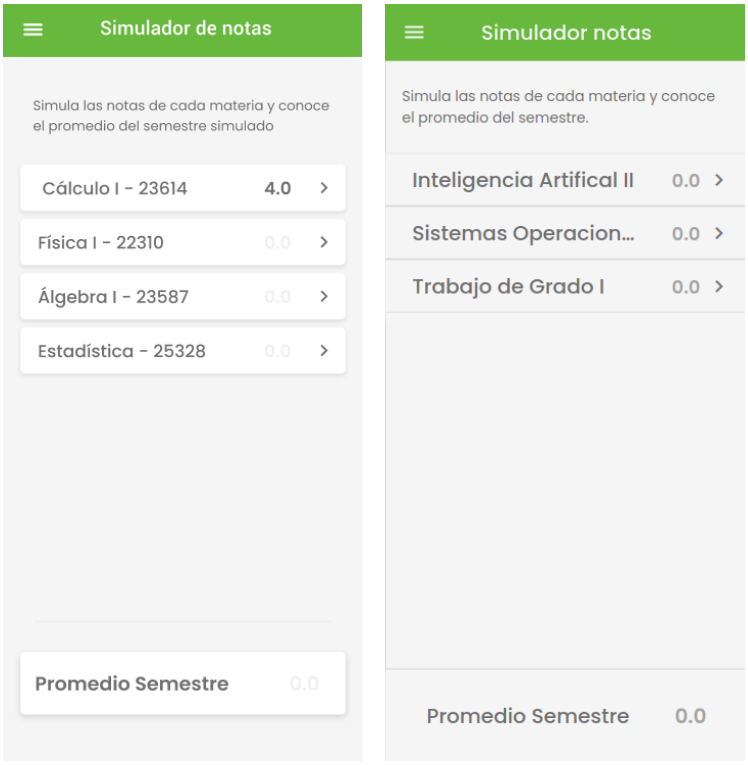


Figura 32. Visualización de una nota definitiva generada.

6.2.6. Simulador de notas. Este módulo permite que los estudiantes puedan planificar su semestre mediante un simulador de notas. Este simulador por defecto carga las materias que tenga matriculadas el estudiante y permite el cálculo de notas y promedio semestral de una forma agradable, rápida y automática. El valor agregado y la virtud de este simulador es la interacción con el usuario, ya que cuenta con validaciones en los input porcentaje y nota, visualización del porcentaje actual simulado, adición y borrado de notas de forma rápida, actualización a tiempo real de la nota definitiva de forma automática, aviso al estudiante de un cambio, adición, o borrado de notas en el sistema académico, posibilidad de agregar una nota definitiva sin necesidad de agregar notas, restablecer el simulador a su estado por defecto e importación de las notas en el sistema directamente al simulador. Este simulador cuenta con dos vistas, una vista principal y una vista secundaria.

6.2.6.1. Vista principal. La vista principal del simulador de notas carga las notas de las materias matriculadas por el estudiante. El estado inicial de cada de las materias es vacío, visualizándose las materias con nota de *0.0* al igual que el promedio del semestre. Esta vista solo cargará las materias con calificación cuantitativa.



(a) Diseño (b) Implementación

Figura 33. Comparación entre diseño e implementación para el módulo del *Simulador de Notas*.

Además, si se actualiza, borra o agrega una nota en la vista secundaria, automáticamente lo detecta la vista principal y recalcula el promedio ponderado.

6.2.6.2. Vista secundaria. La vista secundaria del simulador contiene los datos de las notas, porcentajes y definitivas de cada materia. Esta vista se compone de los siguientes elementos:

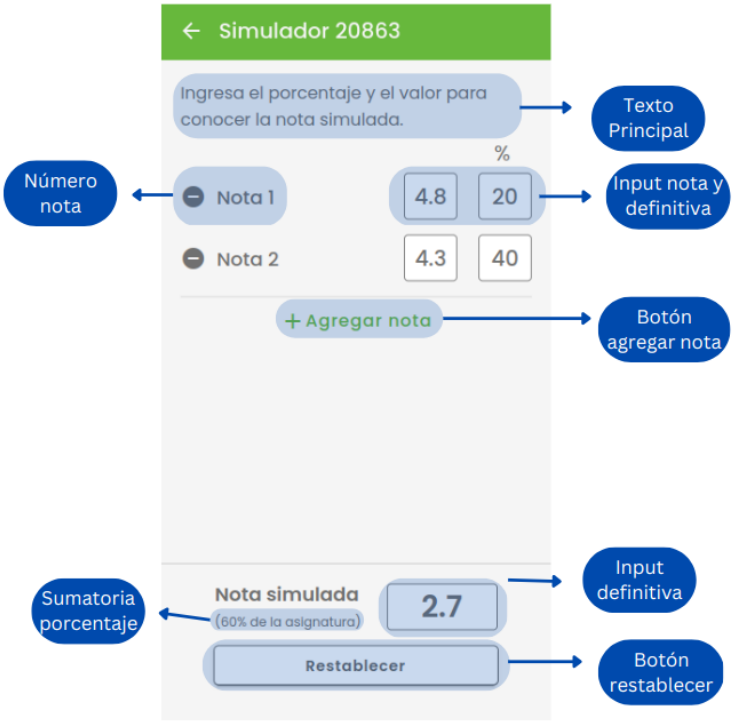


Figura 34. Elementos de la vista secundaria del módulo de *Simulador de Notas*.

Al seleccionar una materia de la vista principal, se entra a la vista secundaria que corresponde a la vista dicha materia. Inicialmente, si el estudiante aún no cuenta con notas en la materia seleccionada, el simulador se encuentra en su estado vacío.

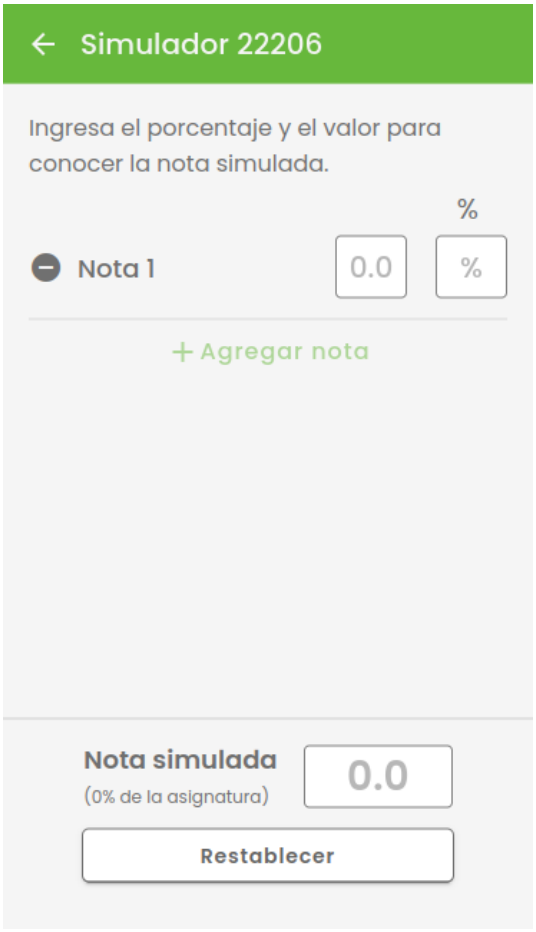


Figura 35. Visualización del estado vacío de la vista secundaria del módulo del *Simulador de Notas*.

Si el estudiante cuenta con notas en el sistema al momento de abrir el simulador de la materia por primera vez, éstas se cargan al simulador automáticamente.

El estudiante tiene acceso a los input de notas y de porcentajes en todo momento al estar en esta vista. Estos input cuentan con validaciones como que las notas deben estar en un rango de 0 a 5 y los porcentajes de 1 a 100. Además, se visualiza en tiempo real la suma de todos los input de porcentaje y el valor de la definitiva en el momento. Si la suma de todos los input

de porcentaje alcanzó el 100 %, el botón de *Agregar nota* se deshabilita. Cabe destacar que el formulario reactivo solo guarda la nota localmente para la persistencia de datos si el formulario es válido. De lo contrario, se guarda la información del simulador hasta dónde éste fue válido.

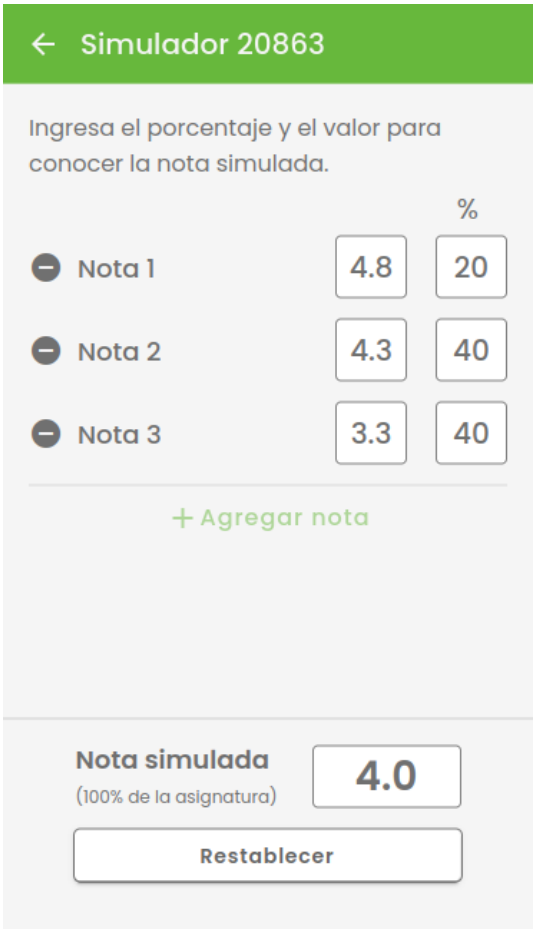


Figura 36. Visualización de la vista secundaria del módulo de *Simulador de Notas*.

Si se detecta que se ingresó en un input un valor que superó el 100 % de la sumatoria total, el formulario muestra error debajo del botón *Agrega nota*. El formulario vuelve a estar correcto y elimina los errores cuando la suma de todos los input sea menor o igual a 100.

Cada vez que se elimine una nota se recalcula la definitiva. Si al borrar dicha nota la sumatoria de los porcentajes sigue superando el 100 %, sigue apareciendo error en el formulario hasta que la sumatoria de los input tengan valores válidos.

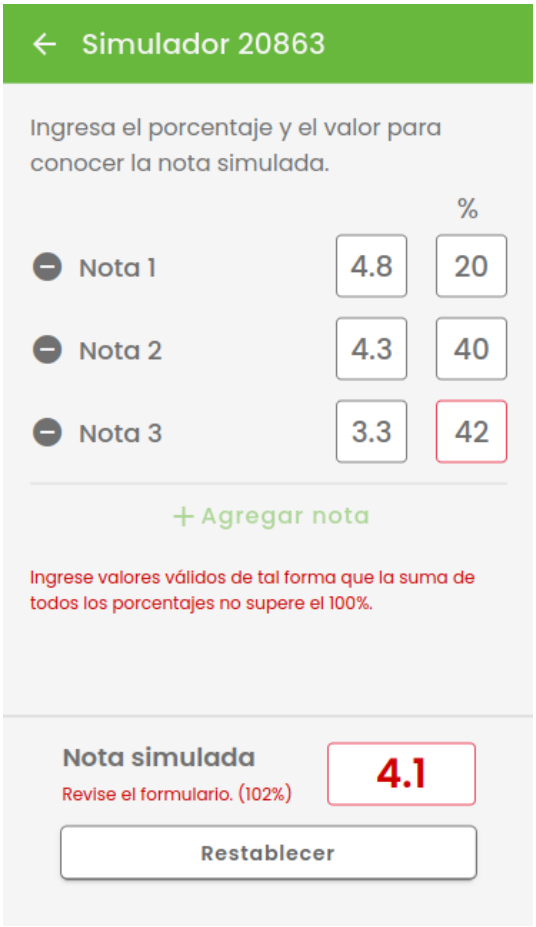


Figura 37. Visualización de la vista secundaria del *Simulador de Notas* con porcentaje inválido.

Cuando se está agregando una nota, se muestra un mensaje alusivo a que debe completar los campos para recalcular la nota definitiva. Esto para darle a entender al usuario que los campos vacíos (o uno de los dos inputs vacíos) no son válidos.

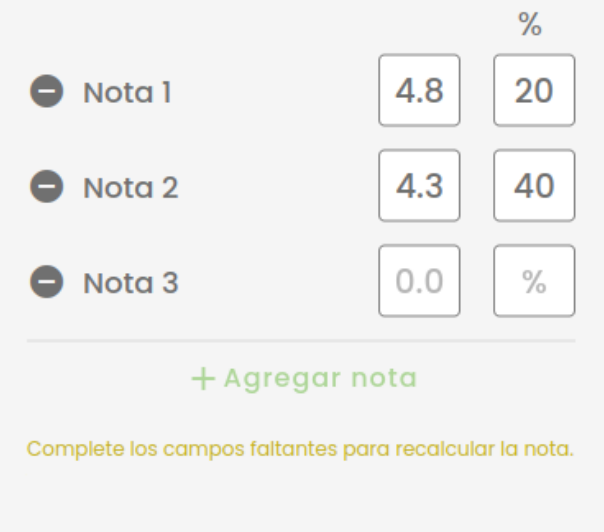


Figura 38. Visualización de la vista secundaria del *Simulador de Notas* al agregar una nota.

Si el usuario desea que su nota definitiva sea una nota especifica, puede hacerlo mediante el input de definitiva. Este input va cambiando su valor dependiendo de las notas y los porcentajes agregados con anterioridad. Pero también se puede acceder a él modificar el valor actual. Al modificar entre valor con notas entre 0 a 5, se eliminan todas las notas (si existen) y solo queda una con el valor ingresado y con un porcentaje del 100 %.

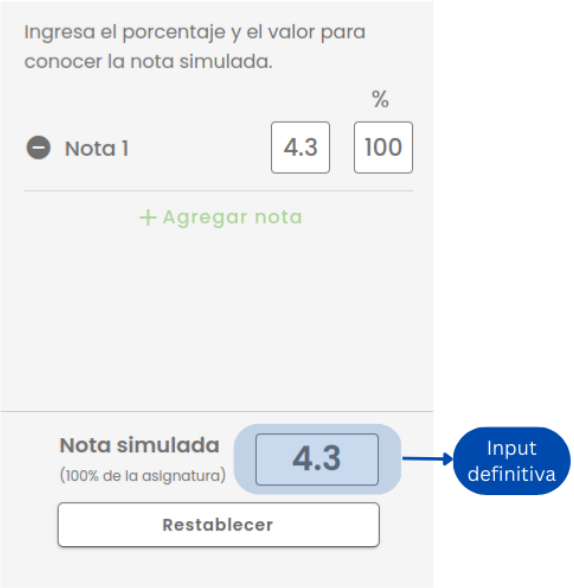


Figura 39. Visualización de la vista secundaria del *Simulador de Notas* al editar el input de la definitiva.

La vista secundaria del simulador también cuenta con un botón de restablecer como se observa en la Figura 34. Este botón tiene como funcionalidad eliminar las notas simuladas y agregar automáticamente las notas que el estudiante tenga en el sistema. Si el estudiante no tiene notas registradas, estará el simulador en su estado vacío como en la Figura 35.

Finalmente, si el simulador ha sido inicializado por primera vez y después de esto se agrega, elimina o edita una nota en el sistema de estudiantes, se da aviso al estudiante mediante un mensaje. Este mensaje indica que para cargar las notas del sistema el estudiante debe oprimir el botón restablecer.

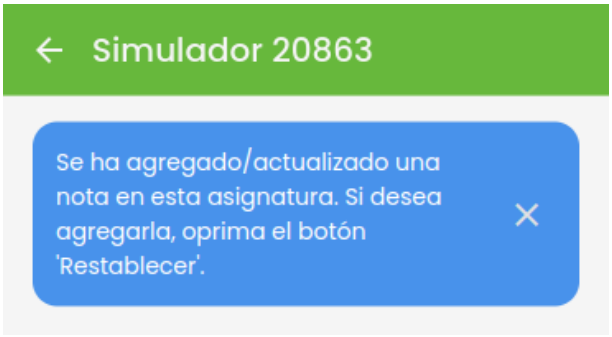


Figura 40. Mensaje en la vista secundaria del *Simulador de Notas* alusivo al cambio de una nota en el sistema de estudiantes.

Al generar una definitiva en la vista secundaria, la vista principal tomará este valor y calcula el promedio ponderado automáticamente con respecto a las demás materias matriculadas.

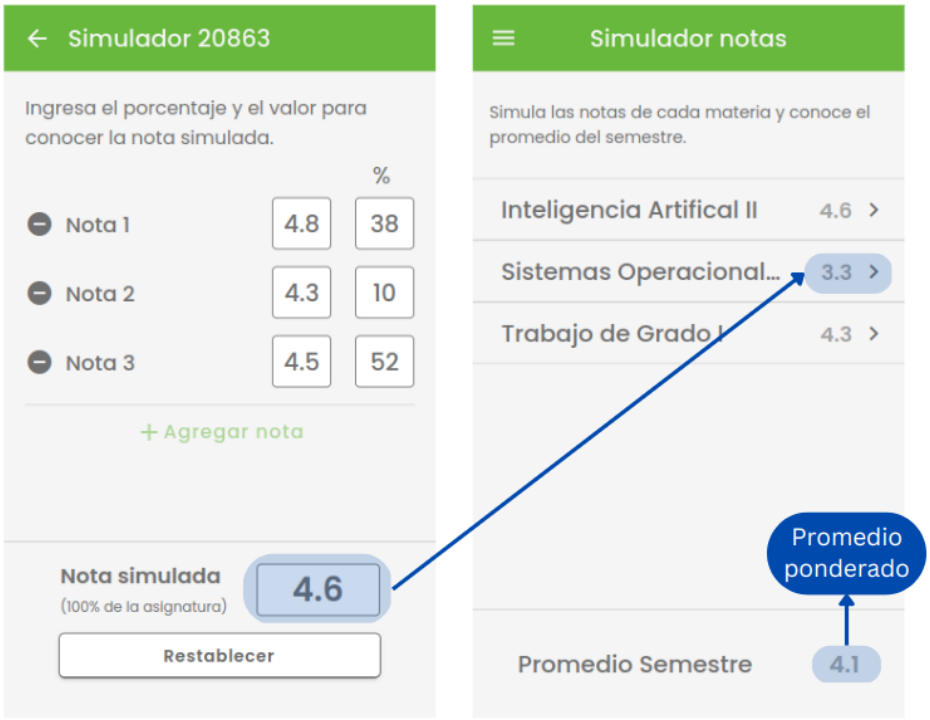
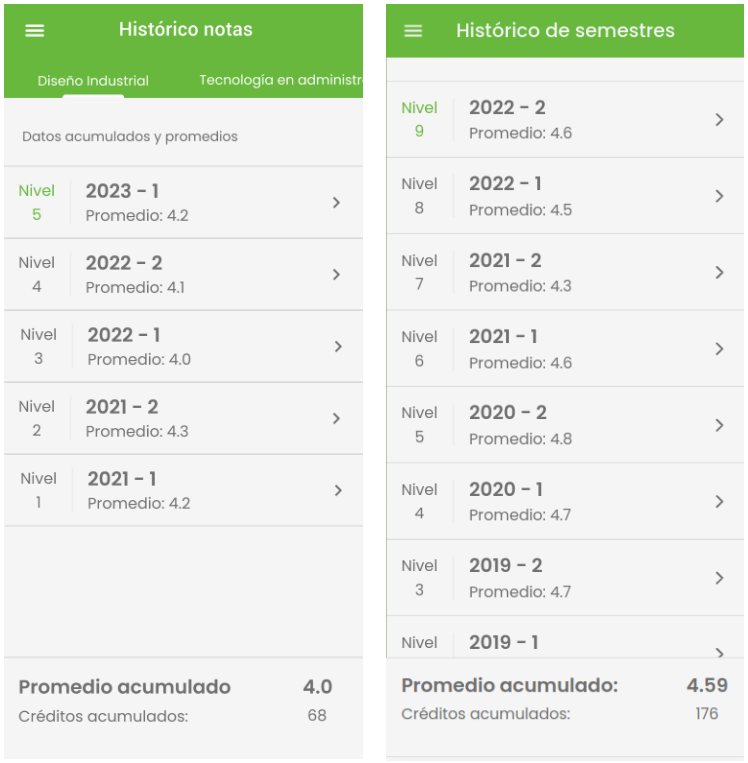


Figura 41. Interacción nota definitiva entre la vista principal y secundaria del *Simulador de Notas*.

6.2.7. Histórico de promedios y asignaturas vistas. En este módulo, los usuarios tienen la capacidad de visualizar su historial de promedios por semestre, presentado en una lista desde el último semestre hasta el primero. Se divide en dos vistas, una con información general de todos los semestres, y otra con información especifica de un semestre.



(a) Diseño (b) Implementación

Figura 42. Comparación entre diseño e implementación para la vista principal del módulo *Histórico Semestral*.

En la Figura 42 se muestra la vista principal del módulo, donde se presenta, para cada cada semestre, información sobre el periodo, nivel y promedio ponderado obtenido en dicho semestre.

En la parte inferior de la pantalla se muestra el promedio acumulado total y los créditos acumulados. La Figura 42(a) muestra una selección en la parte superior, debajo de la cabecera (*header*), que permite a las personas con dos o más carreras visualizar su historial de semestres por carrera. Si el estudiante solo ha cursado una carrera, esta selección no se muestra, como se observa en la Figura 42(b).

(a) Diseño (b) Implementación

Figura 43. Comparación entre diseño e implementación para la vista secundaria del módulo *Histórico Semestral*.

Cuando el usuario selecciona un semestre, se le redirige a una vista secundaria que muestra los detalles de las materias cursadas en ese semestre, incluyendo las notas finales, el promedio del

semestre y los créditos cursados y aprobados. Esto se ilustra en la Figura 43. Las asignaturas se presentan en forma de acordeón. En su estado colapsado, solo se muestra el nombre de la asignatura y su nota final. Al expandir una asignatura, se muestra información adicional, como se muestra en la Figura 44. En el apartado del nombre, también se incluye el código de la asignatura, y en el contenido se muestra el grupo, el o los profesores, la cantidad de créditos de la asignatura, la nota final y las notas parciales subidas por el profesor, que incluyen su porcentaje, nombre y nota.

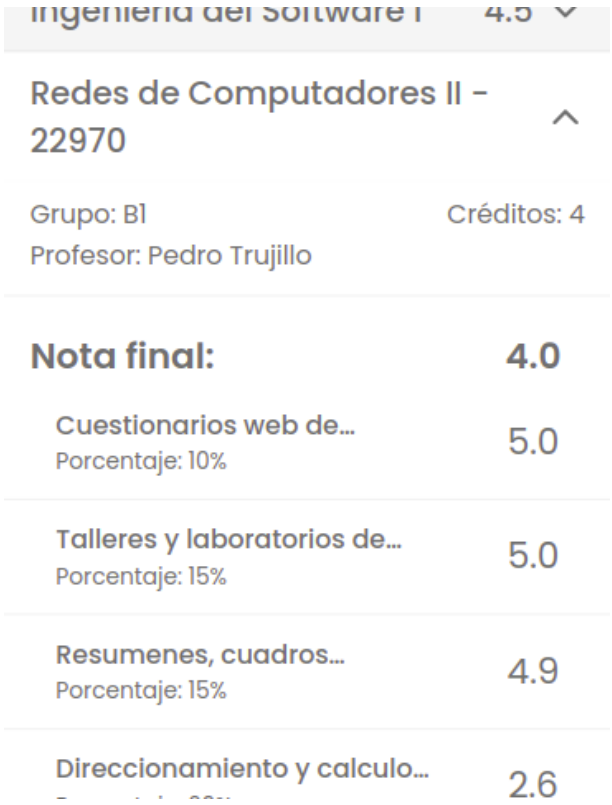


Figura 44. Detalle de asignatura.

6.2.8. Despliegue en plataformas nativas.

6.2.8.1. Configuración de las plataformas. Cada plataforma requiere su propio directorio en la raíz del proyecto. Utilizando la línea de comandos (CLI) proporcionada por *Capacitor*, se configuraron cada una de ellas para que tuvieran todos los elementos necesarios para ejecutarse en entornos nativos.

Dentro de esta configuración se incluyó la gestión de los *plugins* de *Capacitor*, los cuales permiten acceder a las APIs nativas de los dispositivos. En el desarrollo de la aplicación, se utilizaron los siguientes *plugins*:

- Plugin de Red: Con este plugin se pudo acceder a eventos relacionados con cambios en el estado de la red. Se empleó para mostrar mensajes de falta de conexión.
- Plugin de Pantalla de inicio (*Splash Screen*): Permite configurar la pantalla de inicio mientras se carga la aplicación. Se empleó para personalizar la pantalla inicial.
- Plugin de Orientación de la pantalla: Permite bloquear la orientación de la pantalla cuando la aplicación está en primer plano. Se empleó para evitar que el celular cambie de orientación cuando el usuario lo acerca a un lector de QRs.
- Plugin de Barra de estado (*Status bar*): Permite configurar el color del *status bar*, así como otros aspectos de esta. Se empleó para colocar transparente el *status bar* en el módulo de autenticación, y verde en todos los módulos dentro de Home.
- Plugin de Almacenamiento: Permite acceder al almacenamiento del dispositivo. Se empleó

para implementar el patrón *repository* (funcionalidad *offline*), y guardar algunos datos como el identificador del dispositivo.

- Plugin del Teclado: Este plugin brindó acceso a los eventos del teclado del dispositivo, como cuando se muestra u oculta. Se utilizó para ocultar el *footer* de la aplicación en el módulo de autenticación cuando se muestra el teclado.

6.2.8.2. Construcción de las aplicaciones. Una vez se tenían configurados los proyectos para cada plataforma, fue posible construir y empaquetar las aplicaciones nativas para su distribución utilizando una serie de comandos específicos. Este proceso se realiza de forma individual para cada plataforma, lo que significa que no es necesario construir la aplicación para *iOS* si se está construyendo para *Android*, y viceversa.

Durante el desarrollo, se llevaron a cabo pruebas tanto en entornos web como nativos para garantizar el correcto funcionamiento de la aplicación en diferentes plataformas. Es importante destacar que, para realizar pruebas en *iOS*, fue necesario contar con un equipo MAC para construir la aplicación y utilizar los simuladores disponibles en *Xcode*.

6.2.8.3. Subida a tiendas de aplicaciones. La subida de la aplicación a la *Google Play Store* y a la *App Store* se realizó siguiendo los pasos detallados a continuación:

Google Play Store

- Creación de una cuenta de desarrollador en la *Google Play Console*.

- Construcción y firma de la aplicación en modo producción.
- Creación de un nuevo listado de aplicaciones en la *Google Play Console*. Se proporcionó el título, descripción, capturas de pantalla, imágenes promocionales, categoría e información de contacto para la aplicación.
- Subida del bundle (Paquete de *Android*) de la aplicación.
- En este punto, se puede elegir si lanzar la aplicación como una versión beta o alfa para pruebas, o lanzarla directamente al público en general. Se eligió lanzarla en algunas ocasiones como una beta cerrada para verificar su correcto funcionamiento.

La aplicación está disponible en el siguiente enlace.

App Store

- Unirse al *Programa de Desarrolladores de Apple*, que proporciona acceso a los recursos y herramientas necesarios para el desarrollo y distribución de aplicaciones en la *App Store*.
- Crear un *App ID* en la cuenta de desarrollador de *Apple*.
- Generar certificados y perfiles de aprovisionamiento, lo que implica la creación de un certificado de desarrollador, un certificado de distribución en la *App Store* y perfiles de aprovisionamiento.
- Configurar *App Store Connect*, donde se definen los metadatos de la aplicación, como el nombre, descripción, palabras clave, etc.

- Preparar la aplicación para su envío, lo cual incluye la creación de una versión archivada desde *Xcode* que se utilizará en el proceso de envío.
- Proceso de revisión de la aplicación por parte de *Apple*. Después de subir la aplicación, *Apple* realiza una revisión exhaustiva para asegurarse de que cumple con las guías de la *App Store* y no viola ninguna política.
- Publicación de la aplicación.

La aplicación en la *App Store* se encuentra disponible en el siguiente enlace.

6.2.9. Integraciones. Este módulo se encarga de la integración de la aplicación con los torniquetes y vigilantes de la universidad, así como de proporcionar una interfaz para integraciones con otros servicios de la institución, como la biblioteca y los comedores. El módulo se compone de tres componentes principales: un método para a accesos a servicios mediante QR (también llamado token de ahora en adelante), un servicio de control de acceso, y un componente de auditorías.

El método para a accesos a servicios mediante QR permite la verificación de la validez del usuario y del QR en sí, lo que facilita el desarrollo y el testeo de errores en futuras integraciones con otros servicios de la universidad. Por otro lado, el servicio de control de acceso agrega verificaciones adicionales a la verificación genérica para permitir su implementación en torniquetes y una aplicación de vigilantes, con el fin de garantizar el correcto comportamiento del usuario en el proceso de entrada y salida de la universidad. Este servicio verifica que el mismo QR no sea usado

más de una vez, lo que evita el uso de capturas de pantalla y permite implementaciones futuras de QRs de un solo uso. También verifica que un usuario no pueda entrar a la universidad más de dos veces, lo que evita el uso indebido de QRs para ingresar a personas ajenas a la comunidad universitaria.

Por último, el componente de auditorías proporciona una capa adicional de seguridad y control, enfocado en la detección de comportamientos inusuales en el uso de los torniquetes. Además, este servicio ofrece una fuente de datos útiles para la universidad, que puede utilizar a su discreción.

7. Validaciones

Las validaciones de la aplicación se realizaron mediante los test unitarios de *frontend* y *backend*, las pruebas de estrés de los *endpoints* empleados y el equipo QA (Por las siglas de Aseguramiento de Calidad en inglés) del proyecto RSI de la universidad. El objetivo de estas validaciones fueron el de asegurar la calidad del código escrito, facilitar su mantenimiento y verificar el correcto funcionamiento de la aplicación para los diferentes casos que se puedan encontrar.

Por un lado, los test unitarios constituyeron una etapa crucial para evaluar de forma aislada y exhaustiva cada unidad de código, asegurando su correcto funcionamiento tanto en el *frontend*, donde se verificó la interacción con elementos visuales y la lógica asociada, como en el *backend*, donde se evaluó la lógica de negocio y la respuesta adecuada de los servicios, garantizando el cumplimiento de los requisitos funcionales.

Por otra parte, las pruebas de estrés aplicadas a los *endpoints* permitieron evaluar el sistema ante situaciones de alta exigencia, identificando cuellos de botella y asegurando su estabilidad. Del mismo modo, la aplicación de estas permitió establecer valores de rendimiento de la aplicación que podrán ser usados como factor a la hora de escalar sus recursos computacionales o modificar su arquitectura.

Finalmente, el equipo de QA fue el encargado de realizar las validaciones desde el punto de vista de un usuario final, entre las que se encuentran pruebas de funcionalidad, integración y

rendimiento, para asegurar el cumplimiento de los requerimientos funcionales y no funcionales establecidos anteriormente.

La implementación y resultados de las validaciones realizadas por los autores del presente proyecto, correspondientes a las pruebas unitarias y pruebas de estrés, serán expuestos a continuación.

7.1. Test Unitarios

7.1.1. Backend. En el *backend*, se llevaron a cabo pruebas unitarias mediante JUnit. Se creó un conjunto de pruebas para cada servicio con el fin de asegurar su correcto funcionamiento en diversos escenarios. En ese sentido, el *backend* cuenta con una cobertura de código de servicios aproximada del 80 %, lo que significa que se prueba una parte significativa de la implementación del *backend* al ejecutar las pruebas. En total, se cuenta con 8 servicios desarrollados por los autores del presente proyecto, los cuales fueron sometidos a un conjunto de pruebas específico. Teniendo en cuenta lo anterior, se escribió un total de 51 pruebas unitarias, todas las cuales se ejecutaron con éxito. En el anexo 1 se listan las pruebas unitarias realizadas organizadas por servicio.

7.1.2. Frontend. En el *frontend*, se realizaron pruebas unitarias utilizando Karma para ejecutar las pruebas y Jasmine para desarrollarlas. Se creó un conjunto de pruebas unitarias para cada componente y servicio con el fin de asegurar su correcto funcionamiento en diversos escenarios. Además, en términos de cobertura de código, la aplicación alcanzó aproximadamente

un 60 %. Esto significa que se prueba una parte considerable del código total al ejecutar las pruebas. En total, la aplicación contó con 35 componentes y servicios, y se desarrollaron 205 pruebas unitarias, todas las cuales se ejecutaron con éxito. En el anexo 2 se listan algunos ejemplos de las pruebas unitarias realizadas organizadas por módulo.

7.2. Pruebas de Estrés

Estas pruebas, por su naturaleza, se realizaron únicamente en el *backend* mediante JMeter para tres *endpoints* donde se espera una gran cantidad de solicitudes: la validación de QRs, en envío de menús y el envío de horarios

7.2.1. Validación de QRs. Este *endpoint* pertenece al servicio de QR, el cual no se comunica con otras APIs y se encuentra implementado únicamente en el *backend* principal, el *endpoint* realiza varias verificaciones con la base de datos y envía una respuesta de unos pocos bytes indicando si el QR es válido o no, este se encuentra proyectado a recibir una gran cantidad de consultas de manera constante por ser el responsable de gran parte de las integraciones de la aplicación con los servicios de la universidad, considerando que en 2022 la universidad en su sede de Bucaramanga contaba con 15484 estudiantes (UIS, 2022b), se asumió un caso de 5000 solicitudes concurrentes divididas en un minuto, los resultados se muestran a continuación.

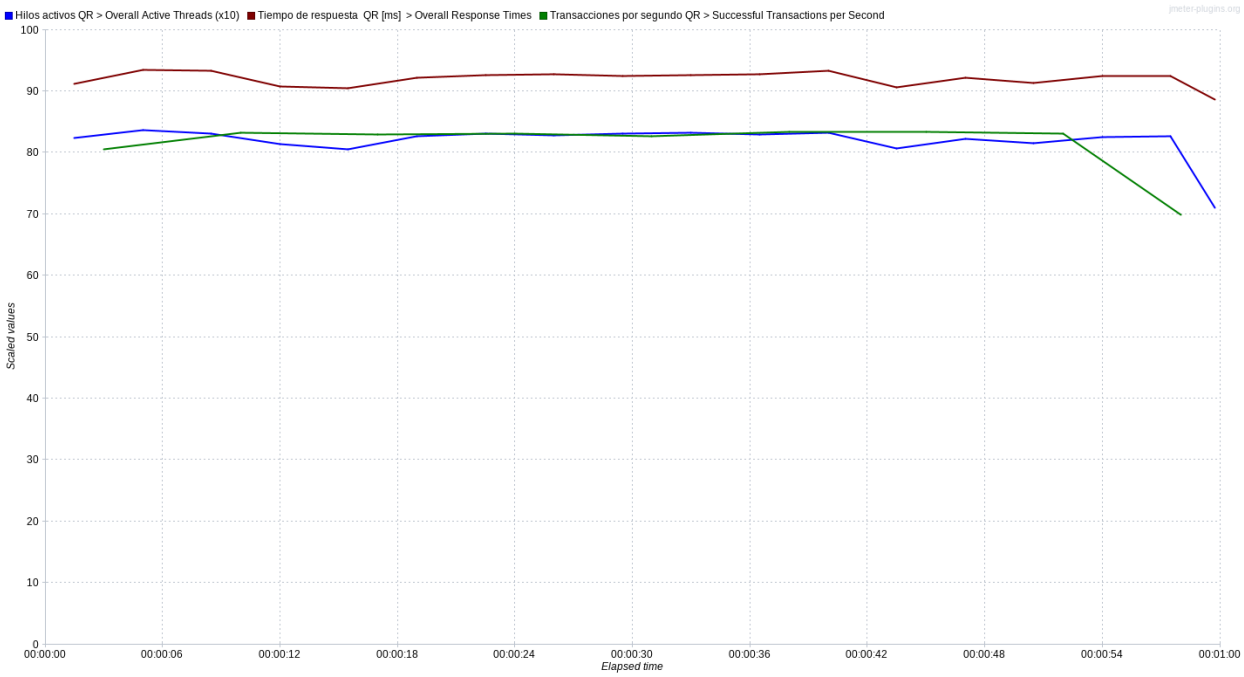


Figura 45. Resultados gráficos de pruebas de estrés para el servicio de verificación de códigos QR.

Nota. Se muestra la relación proporcional entre hilos activos (azul), tiempo de respuesta en milisegundos (rojo) y respuestas por segundo (verde)

Tabla 1
Resultados de pruebas de estrés para el servicio de verificación de códigos QR.

# Pruebas	T. resp. Promedio [ms]	Min [ms]	Max [ms]	Desv. Est. [ms]	Resp. / seg.
5000	92	77	322	9.1	83.23

Los resultados muestran un rendimiento notable y que la respuesta del *endpoint* tuvo una demora promedio de una décima de segundo.

7.2.2. Envío de Menús. Este *endpoint* pertenece al servicio de Autorización, el cual se comunica con la API de *Autorización*, tanto la API como el servicio usan la base de datos principal, su mayor diferencia con el *endpoint* anterior es que este retorna respuestas de mayor

tamaño, pues contienen información relacionada a las acciones que el usuario puede realizar de acuerdo a su rol, debido a que este *endpoint* se usa cuando el usuario accede al menú principal, se encuentra proyectado a recibir una gran cantidad de consultas de manera constante. Para este *endpoint*, considerando que el tamaño de la respuesta es varias veces mas grande, se asumió un caso de 2500 solicitudes concurrentes divididas en un minuto, los resultados se muestran a continuación.

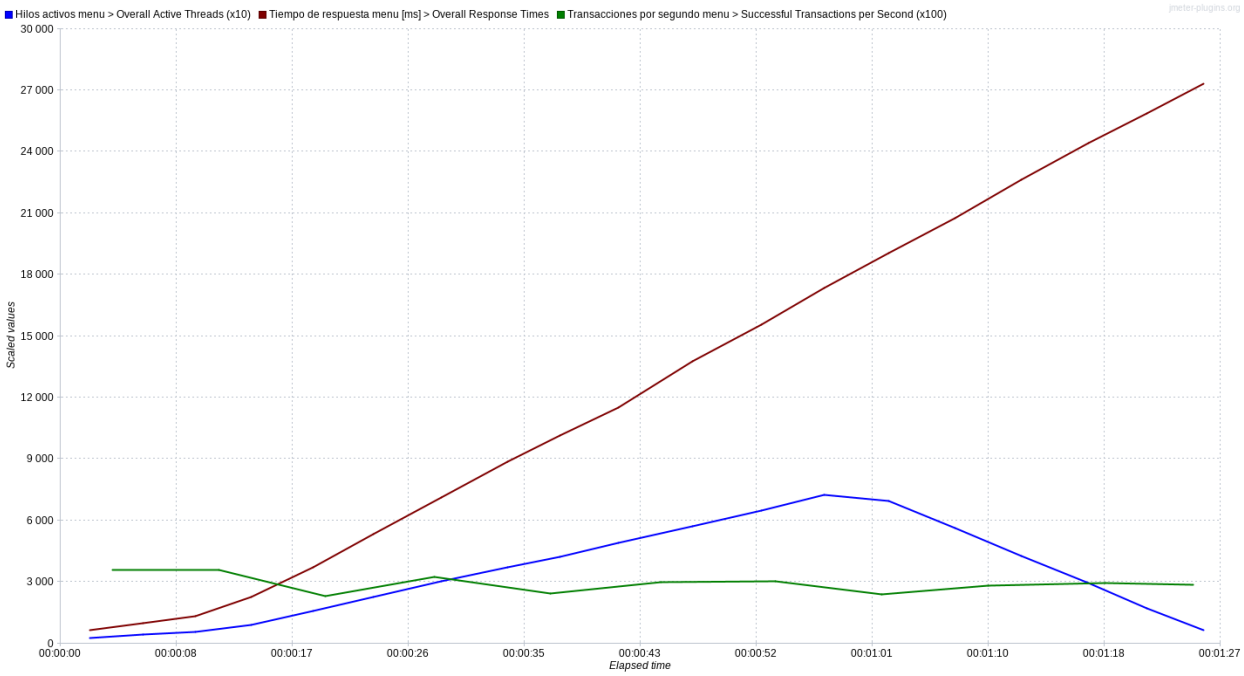


Figura 46. Resultados gráficos de pruebas de estrés para el servicio de envíos de Menús.

Nota. Se muestra la relación proporcional entre hilos activos (azul), tiempo de respuesta en milisegundos (rojo) y respuestas por segundo (verde)

Tabla 2
Resultados de pruebas de estrés para el servicio de envíos de Menús.

# Pruebas	T. resp. Promedio [ms]	Min [ms]	Max [ms]	Desv. Est. [ms]	Resp. / seg.
2500	12359	323	28735	8829.63	28.59

Los resultados muestran que se llegó a un límite de respuestas por segundo y se obtuvo un

rendimiento menor al del *endpoint* de QRs, lo que fue esperable debido al tamaño de las respuestas. Considerando que la cantidad de respuestas por segundo fue de 28.59, se establece un límite de 1700 solicitudes por minuto antes de saturar el *endpoint*. Sin embargo el tiempo de respuesta mínimo de alrededor de 3 décimas de segundo y el hecho de que la cantidad de solicitudes por minuto esperadas no llegan a dicho límite, se puede concluir que el rendimiento es aceptable.

7.2.3. Envío de horarios. Este *endpoint* es otro caso donde se proyecta una gran cantidad de usos dada la frecuencia con la que los estudiantes revisan su horario. Sin embargo, su implementación depende en gran medida de un cliente HTTP conectado a un *backend* secundario para obtener acceso a la base de datos que contiene la información académica del estudiante, lo que se identificó como un posible un cuello de botella. Es por esto que se realizó una prueba de estrés con carga reducida de 1000 solicitudes concurrentes divididas en un minuto, los resultados se muestran a continuación.

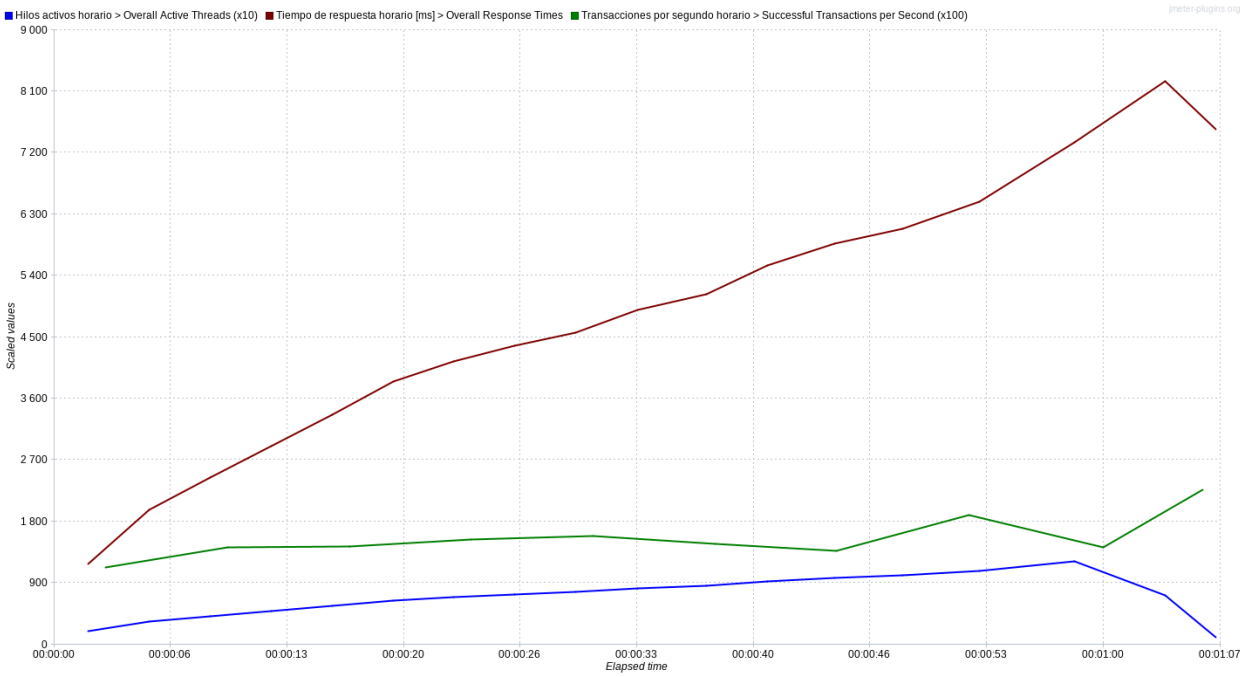


Figura 47. Resultados gráficos de pruebas de estrés para el servicio de envío de horarios.

Nota. Se muestra la relación proporcional entre hilos activos (azul), tiempo de respuesta en milisegundos (rojo) y respuestas por segundo (verde)

Tabla 3
Resultados de pruebas de estrés para el servicio de envío de horarios.

# Pruebas	**T. resp. Promedio [ms]**	**Min [ms]**	**Max [ms]**	**Desv. Est. [ms]**	**Resp. / seg.**
1000	5016	802	9328	1925.71	14.95

En este caso es claro que el *endpoint* llegó a su límite de respuestas por segundo, con un tiempo de respuesta mínimo de 802 milisegundos y uno máximo de poco más de 9 segundos. Considerando que el *endpoint* llegó a un promedio de 15 respuestas por segundo, se encuentra un límite de 900 solicitudes por minuto, posiblemente causado por un cuello de botella presente en el *backend* secundario de la aplicación y la base de datos secundaria. Esta hipótesis se debe debido a que el *endpoint* de menús, que cuenta con un tamaño de respuesta similar y también usa una API

pero con la base de datos principal, logró aproximadamente el doble de rendimiento.

Sin embargo, considerando que a lo largo del día es improbable que mas de 900 los estudiantes revisen su horario dentro del mismo minuto, y que el *frontend* ya hace uso de caché mientras se espera respuesta del *backend*, este cuello de botella no es un problema notable para la experiencia del usuario, y puede ser visto como un punto de mejora una vez se pueda usar únicamente el *backend* principal para el servicio de horario.

8. Conclusiones

El equipo de desarrollo del presente documento, en colaboración con el proyecto RSI, diseñó e implementó una aplicación móvil multiplataforma presente en la *Play Store* y *App Store* que cumple con las necesidades de la comunidad estudiantil. La aplicación, a momento de escritura de este documento, permite a los estudiantes acceder a los servicios más importantes ofrecidos por la aplicación estudiantil actual y les proporciona un mecanismo de identificación seguro y confiable.

En ese sentido, se implementaron con éxito los módulos de información académica para consultar el horario, las notas parciales, las notas definitivas y el histórico de semestres. Además se agregaron funciones como el simulador de notas, las integraciones con servicios de la universidad y un sistema de autenticación de dos factores (2FA) como medida adicional de seguridad para garantizar el acceso seguro a la aplicación.

También se llevaron a cabo pruebas unitarias en el *backend* mediante JUnit, logrando un cubrimiento aproximado del 80 % en el código relacionado a servicios, lo que permitió asegurar el correcto funcionamiento de la aplicación y su integración con los sistemas existentes incluso en casos excepcionales. Del mismo modo, se realizaron pruebas unitarias en el *frontend* mediante *Karma* con un cubrimiento aproximado del 60 %. Lo anterior se hizo para garantizar las buenas prácticas de desarrollo y asegurar una experiencia satisfactoria para el usuario final.

Del mismo modo, las pruebas de estrés permitieron medir un límite de solicitudes por se-

gundo para algunos *endpoints* clave, las cuales se consideraron satisfactorias, llegando a soportar hasta 5000 respuestas por minuto para integraciones con servicios de la universidad, 1700 respuestas por minuto para acceso a acciones permitidas para el usuario, y 900 respuestas por minuto para consultas de horario, todas siendo una fracción notable de la población estudiantil.

9. Trabajo futuro

En primer lugar, se tiene planeado agregar algunas funcionalidades de gran importancia para la comunidad estudiantil, entre las que se encuentran la matrícula de asignaturas, la búsqueda de profesores, la visualización de deudas, la navegación por la universidad mediante un mapa interactivo, la consulta de avance del plan de estudios, servicio de notificaciones, servicio de cambio de contraseña, entre otros.

En segundo lugar, La aplicación desarrollada en este proyecto es la primera de varias que se encuentran proyectadas, entre las que se encuentran una aplicación para los vigilantes de la universidad, otra para la entrega de libros en biblioteca y otra para el servicio de comedores de la universidad, las cuales harán uso de las capacidades de integración e identificación que permite la aplicación estudiantil.

Bibliografía

Adinugroho, T. Y., Reina & Gautama, J. B. (2015). Review of Multi-platform Mobile Application Development Using WebView: Learning Management System on Mobile Platform [International Conference on Computer Science and Computational Intelligence (ICCSCI 2015)]. *Procedia Computer Science*, *59*, 291-297. https://doi.org/https://doi.org/10.1016/j.procs.2015.07.568

Angular. (2023). *Angular Documentation*. Consultado el 21 de mayo de 2023, desde https://angular.io/

Apple. (2023). *Model-View-Controller* [Apple Developer Documentation]. Consultado el 21 de mayo de 2023, desde https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Chinnathambi, K. (2022). *Understanding WebViews*. Consultado el 22 de diciembre de 2022, desde https://www.kirupa.com/apps/webview.htm

Delia, e. a., Galdamez. (2015). Multi-platform mobile application development analysis. (4), 1-2.

Documentation, A. D. (2023). *Xcode Overview*. Consultado el 21 de mayo de 2023, desde https://developer.android.com/studio/intro

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures (Doctoral dissertation)*. Departamento de Ciencias de la Computación de la Universidad de California, Irvine. https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

Gamma, e. a., Helm. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Guo, G. (2018). Design and Implementation of Smart Campus Automatic Settlement PLC Control System for Internet of Things. *IEEE Access*, *6*, 62601-62611. https://doi.org/10.1109/ACCESS.2018.2877023

Ionic. (2023). *Ionic Documentation*. Consultado el 21 de mayo de 2023, desde https://ionicframework.com/docs

JWT. (2022). *Introduction to JSON Web Tokens*. Consultado el 23 de diciembre de 2022, desde https://jwt.io/introduction

Lemonaki, D. (2022). *Frontend VS Backend What's the Difference?* Consultado el 21 de mayo de 2023, desde hhttps://www.freecodecamp.org/news/frontend-vs-backend-whats-the-difference

Network, M. D. (2021). *Lazy Loading*. Consultado el 23 de mayo de 2023, desde https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading

Newman, S. (2015). *Building microservices*. " O'Reilly Media, Inc."

Pahl, C., & Jamshidi, P. (2016). Microservices: A systematic mapping study. *IEEE Transactions on Services Computing*, *9*(2), 261-275.

Roland, M. (2012). Software Card Emulation in NFC-enabled Mobile Phones: Great Advantage or Security Nightmare?, 4.

Satanasaowapak, P., Kawseewai, W., Promlee, S., & Vilamat, A. (2021). Residential access control system using QR code and the IoT. *International Journal of Electrical and Computer Engineering (IJECE)*, *11*, 3267. https://doi.org/10.11591/ijece.v11i4.pp3267-3274

Services, A. W. (2022). *Qué es API RESTful?* Consultado el 21 de diciembre de 2022, desde https://aws.amazon.com/es/what-is/restful-api/

Shivakumar, S. K. (2020). *Mobile Web Performance Optimization*. Addison-Wesley Professional. https://doi.org/10.1007/978-1-4842-6528-4_4

Singh, M., & Shobha, G. (2021). Comparative Analysis of Hybrid Mobile App Development Frameworks, 21-22. https://doi.org/10.35940/ijsce.F3518.0710621

Studio, A. (2023). *Introducción a Android Studio*. Consultado el 21 de mayo de 2023, desde https://developer.apple.com/xcode/

Target, T. (2022). *cross-platform mobile development*. Consultado el 22 de diciembre de 2022, desde https://www.techtarget.com/searchmobilecomputing/definition/cross-platform-mobile-development

Team, J. (2021). *JUnit*. Consultado el 23 de mayo de 2023, desde https://junit.org/junit5/

UIS. (2022a). *Aplicación Estudiantes*. Consultado el 21 de diciembre de 2022, desde https://uis.edu.co/uis-app-estudiantes-es/

UIS. (2022b). *15.484 estudiantes de pregrado presencial de la sede uis bucaramanga iniciaron primer Periodo Académico 2022*. Consultado el 28 de mayo de 2023, desde https://uis.edu.

co/15-484-estudiantes-de-pregrado-presencial-de-la-sede-uis-bucaramanga-iniciaron-periodo-academico-2022-1/

Voon, M., Yeo, S., & Voon, N. (2016). Campus Access Control and Management System. https://doi.org/10.1007/978-3-319-27000-5_32

Zhang, R., Wu, X., & Dan, Z. (2013). Design and implementation of mobile digital campus based on mobile internet, 2598-2602. https://doi.org/10.1109/MEC.2013.6885471

Apéndices

Apéndice A. Tabla de Tests Unitarios en Backend

Tabla 4
Test unitarios para el servicio de Notas

NotasServiceImpl	
Método	**Lista de test unitarios**
getNotasParciales	• Verificar un comportamiento adecuado si la solicitud HTTP hecha hacia la API de notas falla • Verificar un comportamiento adecuado si la solicitud HTTP hecha hacia la API de notas es hecha por alguien que no es estudiante • Verificar si el método de notas parciales retorna notas
getNotasDefinitivasHistorico	• Verificar si el método de notas definitivas para el semestre actual retorna notas
getNotasSemestreHistorico	• Verificar si el método de notas de histórico de semestre retorna notas

Tabla 5
Test unitarios para el servicio de Horarios

HorarioServiceImpl	
Método	**Lista de test unitarios**
getHorario	• Verificar un comportamiento adecuado si la solicitud HTTP hecha hacia la API de horario falla • Verificar un comportamiento adecuado si la solicitud HTTP hecha hacia la API de notas es hecha por alguien que no es estudiante • Verificar si el método de horarios retorna un horario

Tabla 6
Test unitarios para el servicio de QR

QRServiceImpl	
Método	**Lista de test unitarios**
verifyToken	• Verificar un comportamiento adecuado si un token de QR es inválido • Verificar un comportamiento adecuado si un token de QR es válido
verifyQR	• Verificar un comportamiento adecuado si un QR es inválido • Verificar un comportamiento adecuado si un QR es válido
generateQR	• Verificar cuando se intenta generar el QR de un usuario inválido • Verificar cuando se intenta generar el QR de un usuario válido
verifyGateQR	• Verificar un comportamiento adecuado si se intenta verificar un QR inválido el endpoint de integración de torniquetes • Verificar un comportamiento adecuado si se intenta verificar un QR válido por el endpoint de integración de torniquetes
auditConfigurationQR	• Verificar cuando el servicio de auditoría de QR encuentra un QR que no puede ser decodificado • Verificar cuando el servicio de auditoría de QR encuentra que el usuario que lo envió no es válido • Verificar cuando el servicio de auditoría de QR encuentra un QR y usuario válido
generateEncryptedCommands	• Verificar que el método de generar comandos funcione correctamente

Tabla 7
Test unitarios para el servicio de Autenticación

AuthenticationServiceImpl	
Método	**Lista de test unitarios**
getUser	• Verificar un comportamiento adecuado si el usuario a autenticar es válido • Verificar un comportamiento adecuado si el usuario a autenticar es inválido • Verificar un comportamiento adecuado si el usuario a autenticar es estudiante
authenticate	• Verificar la autenticación de un usuario nuevo • Verificar la autenticación de un usuario habitual en el mismo dispositivo • Verificar la autenticación de un usuario cuando este cambia de dispositivo • Verificar la autenticación de un usuario cuando mediante 2FA con un código inválido • Verificar la autenticación de un usuario mediante 2FA con un codigo válido • Verificar la autenticación de un usuario cuando este ya está logeado • Verificar la autenticación de un usuario cuando la contraseña es errónea
getSideMenu	• Verificar que el menu de usuario sea retornado correctamente

Tabla 8
Test unitarios para el servicio de Sesión

SesionServiceImpl	
Método	**Lista de test unitarios**
checkUniqueSessionPrivate	• Verificar cuando un usuario tiene una sesión privada registrada • Verificar cuando un usuario no tiene una sesión privada registrada
checkUniqueSessionPublic	• Verificar cuando un usuario tiene una sesión pública registrada • Verificar cuando un usuario no tiene una sesión pública registrada
updateDeviceToken	• Verificar la actualización de dispositivo
getDeviceToken	• Verificar cuando se encuentra dispositivo un usuario • Verificar cuando no se encuentra dispositivo un usuario

Tabla 9
Test unitarios para el servicio de Documentos de identificación

DocumentTypeServiceImpl	
Método	**Lista de test unitarios**
findAllOrderByOrden	• Verificar que los tipos de documento se listen correctamente

Tabla 10
Test unitarios para el servicio de Autenticación de dos factores (2FA)

TwoFactorAuthServiceImpl	
Método	**Lista de test unitarios**
pairDevice	• Verificar que no se envie un codigo cuando ya se emparejó un dispositivo recientemente • Verificar que no se envie un codigo cuando ya se envió otro código recientemente • Verificar que no se envie un codigo cuando la base de datos no pueda invalidar códigos anteriores • Verificar que no se envie un codigo cuando este no se pueda registrar en la base de datos • Verificar que no se envie un codigo cuando no se pueda encontrar el usuario • Verificar un comportamiento correcto si el servicio de enviar correo electrónico falla • Verificar que se envíe una confirmación al cliente si el código se envía correctamente
verify2FACode	• Verificar un comportamiento correcto si el código de 2FA es inválido • Verificar un comportamiento correcto si el código de 2FA es válido

Tabla 11
Test unitarios para el servicio de fotos

PhotoServiceImpl	
Método	**Lista de test unitarios**
findAllOrderByOrden	• Verificar que se retorne la foto si esta se encuentra en la base de datos de hoja de vida • Verificar que se retorne la foto si esta se encuentra en la base de datos del sistema de personas • Verificar un comportamiento correcto si no se encuentra foto

Apéndice B. Tabla de test unitarios en Frontend

Tabla 12
Test unitarios para el módulo de Autenticación

LoginComponent	
Método	**Lista de test unitarios**
buildForm	• Verificar que el formulario tiene dos campos: usuario y contraseña • Verificar que el campo de usuario y contraseña sean obligatorios
login	• Verificar que se realiza la petición de inicio de sesión si el usuario presiona enter en el campo de contraseña • Verificar que la sesión del usuario se inicia si las credenciales son correctas • Verificar que se muestra el modal para vincular dispositivo • Verificar que se muestra el modal para esperar debido a muchos inicios de sesión • Verificar que se muestra la autenticación en dos pasos • Verificar que aparecen mensajes de error si ingrsa mal el usuario y/o contraseña

Tabla 13
Test unitarios para el módulo de Autenticación

UserService	
Método	**Lista de test unitarios**
getAvatar	• Verificar que devuelve el avatar cuando la petición es exitosa • Verificar que no devuelve el avatar si la petición no es válida
getUser	• Verificar que devuelve los datos del usuario si la petición es exitosa • Verificar que se devuelve *null* si el servidor falla • Verificar que se devuelve *null* si el internet falla

Tabla 14
Test unitarios para el módulo de Autenticación

AuthService	
login	• Verificar que se muestra un *snackbar* de error cuando el servidor falla para la petición del login • Verificar que la sesión del usuario es iniciada correctamente • Verificar que se muestra un *snackbar* de sin conexión cuando el usuario intenta hacer la petición del login sin internet
logout	• Verificar que la sesión del usuario se cierra correctamente
checkToken	• Verificar que la validación del token de autenticación falla si no se le pasó ningún token • Verificar que la validación del token de autenticación falla si el token no se puede decodificar • Verificar que la validación falla si el token está expirado • Verificar que la validación falla si hay algo incorrecto en el token • Verificar que la validación del token es exitosa si el token es válido
autoLogin	• Verificar que al usuario no se le inicia la sesión automáticamente si no hay token de autenticación • Verificar que al usuario no se le inicia la sesión automáticamente si no el token de autenticación es inválido • Verificar que al usuario se le inicia la sesión automáticamente si hay token de autenticación y si tiene id dispositivo
pairDevice	• Verificar que se devuelve *null* cuando el servdiro falla • Verificar que se devuelve *null* cuando no hay internet • Verificar que se devuelve el identificador del dispositivo si la petición es correcta

Tabla 15
Test unitarios para el módulo de Carnet Virtual

HomePage	
Método	**Lista de test unitarios**
ngOnInit	• Verificar que la vista se crea • Verificar que el QR se genera correctamente • Verificar que se muestra la carga esqueleto mientras no hay datos • Verificar que se oculta la carga esqueleto una vez hay datos

Tabla 16
Test unitarios para el módulo de Notas

GradePageComponent	
Método	**Lista de test unitarios**
ngOnInit	• should create • Verificar que se expande/colapsa los acordeones para los notas • Verificar que se asignan los datos en la vista cuando el backend devuelve datos • Verificar que se hace la llamada al backend por datos cuando hay un usuario • Verificar que no se hace la llamada al backend por datos cuando no hay un usuario

Tabla 17
Test unitarios para el módulo de Horario

SchedulePage	
Método	**Lista de test unitarios**
constructor	• Verificar que si aún no teniendo internet la cabecera tiene un titulo • Verificar que la vista no se muestre en modo *offline* si tiene internet, y que si la condicionalidad del estudiante es mayor que 3 (estudiante no activo) no se haga la llamada al backend • Vericar que se traen los datos del horario si el usuario es válido
ionViewWillEnter	• Verificar que se actualiza la cabecera cuando se entra a la vista • Verificar que se refrescan los datos de la vista cuando el estatus de internet cambia a conectado • Verificar que se muestra la carga esqueleto cuando se entra a la vista
getSchedule	• Verificar que se muestra la vista cuando el backend devuelve datos (debería ocultar la carga esqueleto) • Verificar que si no hay un usuario válido se muestre el estado vacío de la vista • Verificar que si no hay internet, ni datos, se muestre la pantalla de sin conexión • Verificar que el modal para la abreviatura de los edificios se muestra correctamente • Verificar que si es la primera vez que el usuario entra a la vista se muestra el banner superior de información
validateURL	• Verificar que la dirección de la clase virtual es válida • Verificar que si la dirección de la clase virtual es inválida no se muestre
VirtualClassInfoComponent	
copyToClipboard	• Verificar que el id de la reunión se copia correctamente al portapapeles • Verificar que la contraseña de la reunión se copia correctamente al portapapeles • Verificar que si no se peude copiar al portapapeles alguna credencial no hayan fallos

Tabla 18
Test unitarios para el módulo de Notas

ScheduleService	
getSchedule	• Verificar que el backend devuelve los datos correctamente • Verificar que se muestra un *snackbar* de error si el servidor • Verificar que se muestra un *snackbar* de error si no hay conexión a internet

Tabla 19
Test unitarios para el módulo de Simulador de Notas

GradeSimulatorPageComponent	
Método	**Lista de test unitarios**
getGrades	• Verificar que solo se muestren las materias cuyas notas sean de tipo numéricas • Verificar que si hay datos de notas simuladas almacenadas localmente se carguen • Veriticar que se muestra el estado vacío de la vista si no hay datos • Verificar que los datos de las notas/materias simuladas se almacenan correctamente • Verificar que los datos de las notas se cargan correctamente • Verificar que si el usuario agrega o elimina una nota se actualice en el almacenamiento local correctamente
convertToDecimalIfNeeded	• Verificar que se convierte correctamente a decimal si es necesario • Verificar que se devuelve el mismo número si ya es decimal • Verificar que se devuelve lo mismo si lo que se pasó no es un número
subjectName	• Verificar que las preposiciones no se conviertan a *title case* • Verificar que no se modifiquen palabras que estén dentro de parentecis • Verificar que se mantienen las preposiciones en minúscula • Verificar que si es un número romano se pasa a mayúscula • Verificar que la primera letra de cada palabra se convierte a mayúscula

Tabla 20
Test unitarios para el módulo de Histórico de Semestres

SemesterHistoryPage	
Método	**Lista de test unitarios**
ngOnInit	• Verificar que la vista se crea correctamente y si no hay datos se muestra la carga esqueleto
getData	• Verificar que los datos del backend se procesan correctamente en la vista • Verificar que se muestra la selección en la parte superior para la carrera de los estudiantes con multiples carreras • Verificar que si el estudiatne solo ha cursado una carrera no se muestra la selección de carreras • Verificar que los datos de la vista se actualicen si el estudiante selecciona otra carrera • Verificar que si se devuelve de la vista secundaria a la principal, los argumentos en la URL se quiten • Verificar que si no hay datos ni conexión a internet, se muestra la pantalla de sin conexión
GradesHistoryPage	
getData	• Verificar que si no hay datos se muestra el estado vacío • Verificar que si no hay datos ni conexión a internet se muestre la vista de sin conexión • Verificar que si hay internet nuevamente y se está mostrando la vista de sin conexión, se actualice y se cambie • Verificar que los datos se asignan correctamente y se asigna el periodo en la cabecera
ionViewWillEnter	• Verificar que cuando entra a la vista en la cabecera aparece la acción para devolverse
ionViewWillLeave	• Verificar que cuando se sale de la vista en la cabecera se quita la acción para volver y se coloca la navegación normal
gradesAccordionChange	• Verificar que los indices de los acordeones asociados a las notas se mantienen correctamente

Tabla 21
Test unitarios para el patrón Repository

RepositoryService	
Método	**Lista de test unitarios**
getCachedRequest	• Verificar que los datos almacenados en cache se borran cuando ya han expirado
objectEquals	• Verificar que la función para determinar si dos objetos son iguales funciona correctamente
clear	• Verificar que todas las llaves de los datos almacenados se borran correctamente cuando algunas están en cache y otras no • Verificar que todas las llaves de los datos almacenados se borran correctamente cuando algunas están en cache y otras no y se excluyen algunas de la eliminación
DataService	
getCacheRequest	• Verificar que si se hace una petición, y hay datos de ésta en cache, se devuelvan los datos del cache • Verificar que si los datos del backend son los mismos que están en cache, se descarten • Verificar que si los datos del backend son distintos a los que están en cache, se actualicen • Verificar que si una petición al backend falla, pero hay datos en cache, se devuelvan dichos datos • Verificar que si una petición al backend falla, y no hay datos en cache, se devuelva *null*

Tabla 22
Test unitarios para el interceptor de peticiones HTTP

AuthHttpInterceptor	
Método	**Lista de test unitarios**
intercept	• Verificar que si el backend devuelva una respuesta de estatus 401 o 403 la sesión del usuario sea cerrada automáticamente • Verificar que si no hay datos del usuario disponibles, no se asignen headers a las peticiones • Verificar que el header de "Authorization" se asigna cuando hay token de autenticación • Verificar que los headers de idusuario, rolusuario, idpersona se agregan si hay datos disponibles • Verificar que si el token de autenticación es inválido, la sesión del usuario es cerrada
constructor	• Verificar que se asigna correctamente el id dispositivo