Universidad Autónoma del Estado de México
Centro Universitario UAEM Zumpango

Ingeniería en computación

Desarrollo de una App Android para horarios escolares

TESIS

QUE PARA OBTENER EL TÍTULO DE

INGENIERA EN COMPUTACIÓN

PRESENTA:

Janeth Guadalupe Cruz Viveros

DIRECTOR:
Dr. Asdrúbal López Chau

Abril, 2026

Resumen

Hoy en día los celulares son parte de todo. Esto ha impulsado el desarrollo de aplicaciones especializadas para diversos ámbitos, incluyendo el educativo. En la carrera, algo que siempre complica a los estudiantes es hacer el horario sin que las materias se traslapen o equivocarse al inscribirse.

En este trabajo se desarrolló una aplicación móvil que ayuda a los alumnos a ver y organizar sus horarios de forma sencilla. La aplicación procesa los archivos *PDF* oficiales proporcionados por la universidad, extrae la información de horarios mediante análisis de texto posicional y la presenta en una interfaz estructurada.

Para el desarrollo del sistema se utilizó *Ionic Framework con React*, empleando la metodología *Scrum* para ir avanzando poco a poco e ir mejorando sobre la marcha. La aplicación puede detectar cuando dos materias chocan de horario, dejar elegir el turno deseado (matutino o vespertino), organizar todo por semestre y al final generar tu horario personalizado. En general, el proyecto demuestra cómo una aplicación móvil bien diseñada puede facilitar la vida académica del estudiante y hacer más práctico el manejo de sus clases.

Abstract

Nowadays, cell phones are part of everything. This has driven the development of specialized applications for various fields, including education. In college, something that always complicates students is making their schedule without subjects overlapping or making mistakes when enrolling.

In this work, a mobile application was developed that helps students view and organize their schedules easily. The application processes the official *PDF* files provided by the university, extracts the schedule information through positional text analysis, and presents it in a structured interface.

For the development of the system, Ionic Framework with React was used, employing the *Scrum* methodology to gradually make progress and improve along the way. The application can detect when two subjects have a scheduling conflict, allow you to choose the shift you want (morning or afternoon), organize everything by semester, and finally generate your personalized schedule.

Overall, the project demonstrates how a well-designed mobile application can facilitate students' academic life and make managing their classes more practical.

Dedicatoria Dedico este trabajo con todo mi amor y gratitud a mis padres, quienes han sido mi mayor ejemplo de esfuerzo, dedicación y perseverancia. Gracias por su apoyo incondicional, por creer en mí incluso cuando yo misma dudaba, y por enseñarme que los sueños se alcanzan con trabajo y fe. A mi madre, quien ha sido el pilar de toda mi vida, mi guía, mi refugio y mi mayor inspiración. Por su fuerza infinita, por su amor sin medida y por enseñarme a levantarme cada vez que la vida me puso a prueba. Este logro también es tuyo, mamá, porque cada paso que di estuvo impulsado por tu ejemplo y tu amor. A mis hermanas, por ser mis compañeras de vida, por sus palabras de aliento, por su cariño y por recordarme siempre que nunca estoy sola. A mis sobrinos, que con su alegría y ternura llenan mis días de luz y me motivan a seguir adelante con una sonrisa. A mi familia, que ha estado conmigo en cada momento, acompañándome con amor, paciencia y comprensión durante todo este camino. Y finalmente, me dedico este logro a mí misma, por no rendirme, por creer incluso cuando el cansancio me quiso vencer, por cada desvelo, por cada lágrima convertida en fuerza, y por demostrarme que soy capaz de lograr todo aquello que me proponga. Sin ustedes, y sin mi propia determinación, nada de esto habría sido posible.

Agradecimientos Agradezco profundamente a la Universidad Autónoma del Estado de México, especialmente al Centro Universitario UAEM Zumpango y a la Licenciatura en Ingeniería en Computación, por brindarme la formación académica y profesional que hoy me permite culminar esta etapa de mi vida. Extiendo mi agradecimiento a todos los docentes que me guiaron con su conocimiento, paciencia y compromiso, y a quienes me motivaron a seguir adelante en cada semestre. A mis amigos, gracias por su compañía, por compartir risas, desvelos y aprendizajes a lo largo de esta carrera. Finalmente, agradezco a todas las personas que, de una u otra forma, contribuyeron a que este proyecto se hiciera realidad. Cada palabra, gesto y apoyo recibido ha dejado una huella imborrable en mi camino.

Contenido

Lista de figuras

Lista de tablas

Lista de códigos

Capítulo 1

Introducción

El uso de dispositivos móviles se ha extendido de manera significativa en los últimos años, lo que ha impulsado el desarrollo de aplicaciones especializadas para diversos ámbitos, incluyendo el educativo. El desarrollo de software móvil se ha convertido en una área estratégica de la ingeniería y tecnología de la información.

Una de las problemáticas recurrentes en la carrera de Ingeniería en Computación (ICO) del Centro Universitario UAEM Zumpango (CU UAEM Zumpango) es que debido a las particularidades de la carrera, muchos alumnos recursan unidades de aprendizaje en distintos horarios, lo que hace que elegir los horarios sin traslapes sea una tarea complicada para muchos alumnos. Esto se repite cada inicio de semestre, durante cada periodo de inscripción.

1.1 Antecedentes

En el ámbito educativo, sobre todo en la educación superior, los alumnos normalmente reciben sus horarios en formatos como PDFs, Word, Excel o imágenes, lo que impide una claridad activa de información. Se identifico que esto puede complicar la forma en que organizan sus materias, puesto que no pueden ver fácilmente su carga académica ni verificar si hay conflictos de horarios entre las clases. El incremento en el uso de dispositivos móviles ha propiciado el desarrollo de aplicaciones especializadas para la

gestión académica, sin embargo, muchas de estas soluciones no se adaptan a los formatos específicos de cada institución educativa. No obstante, muchas de estas aplicaciones no se ajustan a las características particulares de cada institución y no permiten importar horarios desde un archivo que proporciona la universidad, lo que limita su eficiencia.

1.2 Importancia del problema

Una aplicación para dispositivos móviles creada especialmente para gestionar los horarios escolares, brinda una opción actual y útil para los estudiantes. La posibilidad de elegir asignaturas, ver los horarios y verificar sus disponibilidad desde teléfonos mejora la organización académica y ayuda a ahorrar tiempo. Adicionalmente, si la aplicación se ajusta al formato de los horarios que ofrece la carrera de ICO del CU UAEM Zumpango, se transforma en una herramienta valiosa y adaptada a las necesidades de cada estudiante.

1.3 Planteamiento del problema

Los estudiantes universitarios normalmente reciben sus horarios en archivos PDF, Word, Excel e imágenes, que muestran las materias disponibles junto con sus respectivos días, horas, aulas y profesores. Estos formatos impiden la manipulación dinámica de la información y dificulta la identificación de conflictos de horarios durante el proceso de selección de materias.

1.4 Objetivos

1.4.1 Objetivo general

Diseño e implementación de una aplicación móvil para dispositivos Android, utilizando el *Framework Ionic React*, que permite generar y personalizar horarios escolares a partir

de archivos PDF proporcionados por la universidad, con el fin de facilitar la organización académica del estudiante.

1.4.2 Objetivos específicos

1. Analizar el formato de los archivos PDF utilizados por la universidad para presentar los horarios escolares.

2. Diseñar la interfaz gráfica de usuarios de la aplicación móvil con un enfoque en la claridad visual y facilidad de uso.

3. Diseño e implementación de un módulo que permite procesar archivos PDF e interpretar la información contenida (materias, horarios, días, profesores, claves, aulas).

4. Implementar un sistema de verificación automática de conflictos de horarios entre materias seleccionadas.

5. Validar la aplicación móvil mediante pruebas de funcionalidad y usabilidad con usuarios reales.

1.5 Alcance o delimitación

La presente aplicación estará dirigida a estudiantes universitarios de nivel superior que reciben sus horarios escolares en archivos PDF, Word, Excel o imágenes digitalizadas, sin embargo, el sistema está diseñado para procesar exclusivamente archivos PDF. Las características principales que serán consideradas dentro del desarrollo incluyen:

- Procesamiento local de archivos PDF que contienen horarios académicos.
- Extracción de datos relevantes como nombre de la materia, día, hora y profesor.
- Interfaz gráfica para visualización y personalización de horarios.

- Prevención de cruces de horario al momento de seleccionar materias.
- Despliegue en dispositivos con sistema operativo Android.

No se contemplará en está aplicación:

- La inscripción oficial de materias en el sistema escolar.
- Sincronización con bases de datos institucionales.
- Soporte para sistemas operativos distintos a Android.

1.6 Declaratoria de uso de IA

El presente trabajo es de autoría original; sin embargo, se utilizaron herramientas de inteligencia artificial para el refinamiento estilístico en algunas secciones del documento.

Capítulo 2

Marco teórico

2.1 Desarrollo de Aplicaciones Móviles

2.1.1 Historia y evolución del desarrollo móvil

El desarrollo de las aplicaciones móviles experimentó una transformación significativa con la aparición de los *smartphones*, anteriormente, las aplicaciones eran de carácter básico y se limitaban a funciones como llamadas, mensajes y gestión de agendas personales. [19]

Con la llegada de sistemas operativos como iOS y Android, el desarrollo móvil se diversificó y permitió la creación de aplicaciones con fines educativos, sociales, financieros, entre otros. [15]

Estas plataformas ofrecieron entornos de desarrollo más robustos y accesibles para los programadores. En la actualidad ya existen herramientas que facilitan el desarrollo multiplataforma, sin la necesidad de escribir código completamente distinto para cada sistema operativo, lo que optimiza el tiempo de desarrollo y facilita el mantenimiento de las aplicaciones. [15]

2.1.2 Comparación entre plataformas: Android vs iOS

Android y iOS son las dos principales plataformas móviles más utilizadas en el mundo [14]. Android, es desarrollado por Google, es un sistema operativo de código abierto que permite a los fabricantes y desarrolladores una mayor flexibilidad para modificar e implementar el sistema en una amplia variedad de dispositivos. [15]

iOS, por otro lado, es un sistema cerrado exclusivo de los dispositivos Apple, con estrictas políticas de publicación, pero con alto rendimiento gráfico, estabilidad y optimización del sistema. [17]

Mientras que Android domina el mercado en términos de usuarios y variedad de dispositivos, iOS se destaca por su consistencia y optimización en dispositivos limitados. Ambos sistemas ofrecen herramientas potentes de desarrollo, como Android Studio y Xcode, respectivamente. Las diferencias en arquitectura, lenguajes de programación y distribución de aplicaciones pueden influir en la elección de una plataforma, según el público objetivo y las necesidades del proyecto.

Hasta el 2024, las tres últimas versiones de Android son las siguientes:

- Android 14 (*Upside Down Cake*)- Octubre 2023.
- Android 13 (*Tiramisu*) - Agosto 2022.
- Android 12 () - Octubre 2021.

  Y las tres ultimas versiones de iOS son:

- iOS 17 - Septiembre 2023.
- iOS 16 - Septiembre 2022.
- iOS 15 - Septiembre 2021.

2.1.3 Tipos de aplicaciones: nativas, híbridas, web apps

Para el desarrollo del proyecto se considero la elección entre una aplicación nativa, híbrida o web, dado que las aplicaciones móviles pueden clasificarse en tres tipos

principales como ya se había mencionado: nativas, híbridas y aplicaciones web. [5]

Las aplicaciones nativas se caracterizan por ofrecer un mayor rendimiento, sin embargo, requieren el desarrollo independiente para cada sistema operativo. Cada una de ellas tiene características particulares que las hacen adecuadas para distintos contextos de desarrollo y necesidades del usuario. A continuación se presenta una breve descripción de cada una de ellas.

- **Nativas**: Desarrolladas específicamente para un sistema operativo. Utilizan lenguajes de programación propios del entorno, como Java/Kotlin para Android o Swift para iOS. Estas aplicaciones ofrecen el mejor rendimiento, mayor velocidad y acceso completo a los recursos del *hardware* del dispositivo. Sin embargo, su principal desventaja es que requieren desarrollar versiones por separadas para cada plataforma, lo que aumenta el tiempo, el esfuerzo y los costos de mantenimiento. [15]

- **Híbridas**: Son desarrolladas utilizando tecnologías web (*HTML, CSS, JavaScript*) y se ejecutan dentro de un contenedor nativo. *Frameworks* como *Ionic o React Native o Flutter,* permiten compilar estas aplicaciones para múltiples plataformas desde un solo código base. Su ventaja principal es la eficiencia en tiempo de desarrollo y el bajo costo. No obstante, su rendimiento suele ser inferior al de las aplicaciones nativas, especialmente en procesos que demanda un alto uso de recursos gráficos o funciones especificas del dispositivo. [23]

- **Web Apps**: Son sitios web optimizados para dispositivos móviles, accesibles desde el navegador sin necesidad de ninguna instalación. Tienen la ventaja de ser multiplataforma, fáciles de mantener y actualizaciones en tiempo real. Sin embargo, presentan limitaciones importantes en el acceso al *hardaware* del dispositivo y en funcionalidades que requieren conexión continua o procesamiento intensivo, lo que restringe su uso en aplicaciones más complejas. [5]

2.2 Sistema Operativo Android

2.2.1 Arquitectura de Android

Android está construido sobre una arquitectura en capas que permite modularidad y eficiencia. Se compone de los elementos mostrados en la figura 2.1, los cuales son explicados a continuación:

- *Linux Kernel*: Proporciona la base del sistema operativo, gestionando recursos como la memoria, procesos y seguridad.
- *Hardware Abstraction Layer (HAL)*: Facilita la comunicación entre el *hardware* del dispositivo y el sistema operativo.
- *Android Rountime (ART)*: Ejecuta las aplicaciones y gestiona la memoria.
- *Framework de Android*: Contiene las APIs utilizadas por los desarrolladores para crear aplicaciones.
- Aplicaciones: Son las interfaces visibles para los usuarios, construidas sobre las capas anteriores.

2.2.2 Ciclo de vida de una aplicación Android

El ciclo de vida de una aplicación Android define los estados por los que pasa una actividad, desde su creación hasta su destrucción. Estos estados son controlados mediante métodos como *onCreacte(), onStart(), onResume(), onPause(), onStop() y onDestroy()*. Durante el desarrollo de este proyecto, fue fundamental comprender este ciclo para gestionar correctamente los recursos del sistema y evitar errores como pérdida de datos o consumo innecesario de batería. La implementación correcta de estos métodos resultó esencial para garantizar la estabilidad de la aplicación.

2.2.3 Ventajas de desarrollar para Android nativo

Desarrollar en Android ofrece múltiples ventajas, como la gran base de usuarios, la posibilidad de publicar aplicaciones fácilmente en *Google Play*, acceso a una variedad de dispositivos y herramientas gratuitas de desarrollo como *Android Studio*. Además, su código abierto permite a los desarrolladores explotar y modificar componentes del sistema si así lo requieren.

2.2.4 Ventajas de desarrollar con un Framework híbrido

El desarrollo híbrido permite construir aplicaciones móviles usando tecnologías web, lo que reduce los tiempos de desarrollo y facilita su mantenimiento. *Frameworks* como *Ionic* permite escribir una sola base de código y despegarla en distintas plataformas, lo que resulta ideal para equipos pequeños o proyectos académicos. Asimismo, permite acceso a funcionalidades del dispositivos mediante *plugins nativos*, combinando lo mejor de ambos mundos, web y nativo.

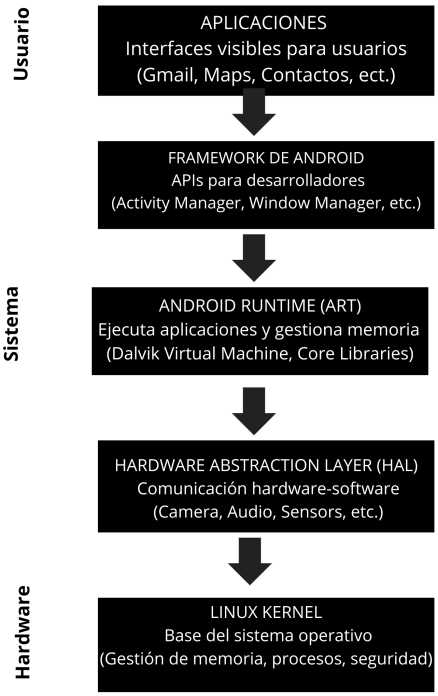

Figura 2.1: Arquitectura de Android.

La arquitectura de Android está organizada en capas que interactúan para ofrecer funcionalidad completa. Como se observa en la Figura 2.1, estas incluyen las aplicaciones y Frameworks, los servicios del sistema, el Android Runtime (ART), la capa de abstracción de hardware (HAL) y finalmente, el núcleo Linux, que gestiona el hardware.

2.3 Framework Ionic

2.3.1 Características relevantes de Ionic

Entre los distintos *Frameworks* disponibles, *Ionic* destaca por su popularidad en los últimos años, gracias a su enfoque práctico y eficiente para el desarrollo de aplicaciones móviles híbridas. Estas se basan en tecnologías web como lo son *HTML, CSS, JavaScript*, y puede integrarse con varios *Frameworks* como *Angular o React*, lo que cual

hace que sea más flexible y accesible para algunos desarrolladores con conocimientos en desarrollo web.

Una de las varias ventajas más notorias de *Ionic* es que permite crear una sola base de código para generar aplicaciones tanto para *Android* como para *iOS*. Esto representa un gran ahorro de tiempo y recursos, ya que no es necesario desarrollar una aplicación distinta para cada sistema operativo. Además mediante el uso de herramientas como capacitor o cordova, se pueden acceder a funciones nativas del dispositivo (un ejemplo puede ser la cámara del teléfono o el GPS) sin tener que escribir código especifico para cada plataforma. También destaca por su catálogo de componentes visuales preconstruidos que siguen los estilos nativos de *Android* y *iOS*, lo que nos ayuda a que la aplicación se vea profesional sin invertir tanto esfuerzo en el diseño en interfaces desde cero.

Como señala Griffith. [9] el uso de *Frameworks* híbridos como *Ionic* representa una solución eficiente y rentable para desarrollos móviles rápidos y adaptables a múltiples sistemas operativos.

2.4 Diseño de Interfaces Ionic

2.4.1 Componentes gráficos en Ionic: Activity, Fragment, RecyclerView, etc.

A diferencia del desarrollo nativo en *Android*, donde se utilizan elementos como *Activity* o *Fragment*, *Ionic* trabaja con páginas y componentes re utilizables que se organizan en rutas. También gracias a su diseño basado en la web, los componentes visuales como botones, menús, listas o tarjetas pueden personalizarse con facilidad y adaptarse automáticamente al estilo del sistema operativo en el que se ejecuta la aplicación. Esto hace que el desarrollo de la interfaz sea mucho más ágil y visualmente consistente entre diferentes dispositivos.

Según Griffith [10], *"Esta adaptabilidad permite ofrecer una experiencia uniforme sin*

importar si se ejecuta en un celular Android o un iPhone."

Figura 2.2: Ejemplo de un botón en *Ionic*.

La figura 2.2 muestra un botón desarrollado con *Ionic Framework*, el cual permite ejecutar una acción al ser presionado.

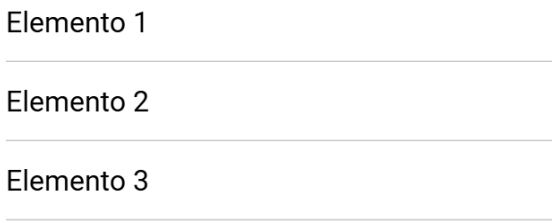

Figura 2.3: Ejemplo de una lista en *Ionic*.

La figura 2.3 presenta una lista *IonList* con varios elementos *IonItem*. Este componente se utiliza comúnmente para organizar opciones o datos de manera ordenada.

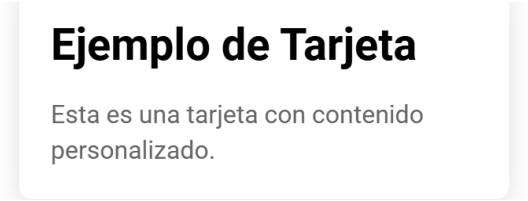

Figura 2.4: Ejemplo de una tarjeta en *Ionic*.

En la figura 2.4 se observa una tarjeta *(IonCard)* con un título y contenido. Este tipo de componente es útil para mostrar información agrupada de forma visualmente atractiva.

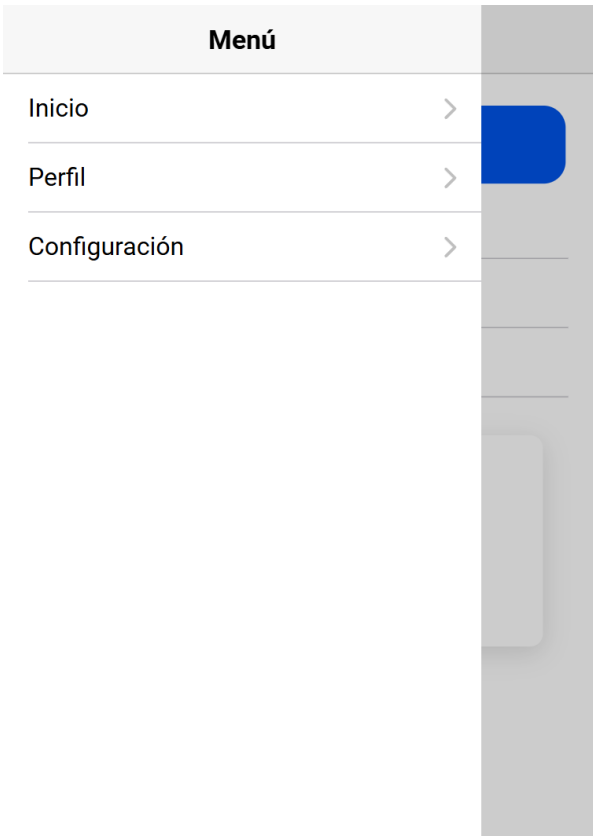

Figura 2.5: Ejemplo de un menú en *Ionic*.

Finalmente, la figura 2.5 muestra el menú lateral *(IonMenu)* que se despliega desde un costado de la pantalla, ofreciendo al usuario diferentes opciones de navegación.

2.4.2 Usabilidad y experiencia del usuario (UX/UI)

En el desarrollo móvil, no solo basta con que una aplicación funcione, también debe ser fácil de usar, que sea atractiva visualmente y clara en su navegación. *Ionic* facilita la creación de interfaces pensadas para el usuario, utilizando principios modernos de diseño centrado en la experiencia del usuario. La estructura modular del *Framework* permite organizar la aplicación de una forma lógica y herramientas como los íconos integrados, la navegación fluida y la posibilidad de aplicar temas personalizados hacen que el desarrollo sea más amigable y enfocado en la usabilidad.

Como bien explica Galitz [7], *"Una buena gráfica no solo debe de ser estética,*

sino funcional y comprensible para el usuario desde su primer contacto." Esta cita refleja perfectamente la importancia de que una aplicación sea intuitiva y práctica, especialmente cuando va dirigida a estudiantes, como en el caso de está aplicación de horarios escolares.

2.4.3 Estructura general de una aplicación Ionic

En general una aplicación desarrollada en *Ionic* sigue una estructura organizada en carpetas que agrupan las páginas, componentes, servicios, temas y recursos. Las rutas definen la navegación entre diferentes pantallas, mientras que los servicios se encargan de la lógica de negocio y la conexión con bases de datos o APIs externas. El proyecto se puede extender fácilmente con bibliotecas y *Plugins*, como *Firebase* para manejar autenticaciones y almacenamiento en la nube, o *Plugins* nativos para acceder a funciones del dispositivo.

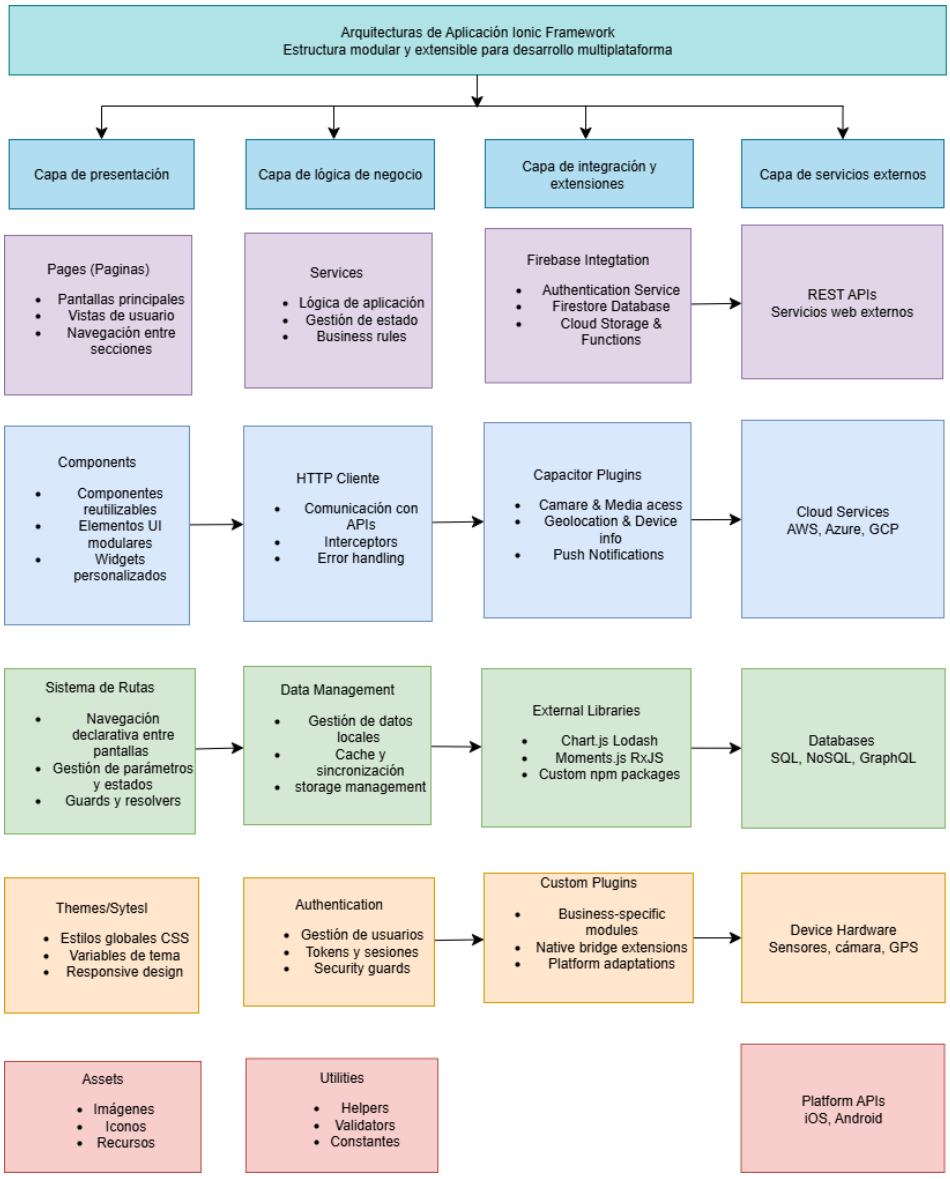

Figura 2.6: Estructura general de una aplicación en *Ionic Framework*.

La figura 2.6 muestra la estructura general de una aplicación desarrollada con *Ionic Framework*, organizada en capas de presentación, lógica de negocio, integración/extensiones y servicios externos, está modularidad permite un desarrollo escalable y adaptable a diferentes plataformas.

2.4.4 Desventajas de Ionic

A pesar de sus muchas ventajas, *Ionic* no es perfecto. En algunos casos, especialmente en dispositivos con recursos limitados, el rendimiento puede verse afectado, sobre todo si se utilizan muchas animaciones o funciones que depende del *hardware*. Además cuando se necesita implementar funcionalidades muy especificas del sistema operativo, puede ser necesario buscar soluciones más técnicas o incluso recurrir al desarrollo nativo.

Estas limitaciones no impiden que *Ionic* sea una opción para proyectos como el de está tesis, donde la prioridad es compatibilidad multiplataforma, la velocidad de desarrollo y una interfaz fácil de usar. [10]

2.5 Lenguaje de programación React

2.5.1 Características relevantes de React

React es una biblioteca de *JavaScript* desarrollada por *Facebook* que permite construir interfaces de usuario dinámicas y eficientes. Su enfoque principal es facilitar la creación de componentes reutilizados que se pueden combinar para construir interfaces complejas. Está arquitectura basada en componentes promueve una mayor organización del código y facilita su mantenimiento a lo largo del ciclo de vida del proyecto.

Una de sus características más destacadas es el uso de *Virtual DOM*, es una representación en memoria del *DOM* real que permite actualizar unicamente los elementos necesarios en pantalla. Gracias a está técnica, con *React* se logra un rendimiento notablemente superior al de métodos tradicionales de manipulación del *DOM*. Además *React* emplea una sintaxis especiales llamada JSX, que permite escribir código *JavaScript* combinado con *HTML* de forma declarativa, lo cual mejora la legibilidad del código y reduce errores durante el desarrollo de interfaces. Cuando se trabaja con *TypeScript* está síntesis se extiende a archivos con extesion *.tsx* lo que permite agregar tipado estático sin perder las ventajas de JSX. Esta combinación no solo ayuda a detectar errores en tiempo de desarrollo, sino que también contribuye

a crear aplicaciones más robustas y escalables, especialmente en proyectos grandes o colaborativos. Según Banks y Porcello [1], *"React transforma la manera en que los desarrolladores construyen interfaces al proporcionar una estructura clara y declarativa, centrada en la reutilización de componentes."*

Un aspecto relevante de *React* es su amplio ecosistema. Al integrarse fácilmente con otras herramientas como *Redux* para la gestión del estado o *React Router* para la navegación, se convierte en una solución muy flexible tanto para aplicaciones web como móviles, especialmente cuando se combina con *Frameworks* como *Ionic* o *React Native*.

2.6 Entorno de Desarrollo VSCODE

2.6.1 Características de VSCODE

Visual Studio Code (VSCode) es un editor de código fuente desarrollado por *Microsoft* que ha ganado gran popularidad entre desarrolladores gracias a su versatilidad, ligereza y extensibilidad. A diferencia de entornos de desarrollo más pesados, *VSCode* permite trabajar con una gran variedad de lenguajes y tecnologías mediante la instalación de extensiones especificas. Una de sus principales fortalezas es la integración nativa con *Git*, lo que puede facilitar el control de versiones sin salir del editor, además ofrece auto completado inteligente, resaltado de sintaxis, depuración en tiempo real y una terminal integrada, lo que permite una experiencia de desarrollo fluida y eficiente.

Otra Característica importante de *VSCode* es su compatibilidad con entornos de desarrollo para aplicaciones móviles, como *React Native o Ionic*, gracias a plugins y extensiones que simplifican la ejecución de emuladores, el uso de *SDKs* y la depuración de aplicaciones móviles. [4]

2.6.2 Herramientas complementarias: emuladores, SDKs, plugins

Durante el desarrollo de aplicaciones móviles, el uso de herramientas complementarias resulta esencial para facilitar el proceso de prueba, depuración y despliegue. Entre las más utilizadas se encuentran los emuladores, que permiten ejecutar una aplicación móvil en un entorno virtual que simula el comportamiento de un dispositivo físico. Esto resulta muy útil para verificar la funcionalidad y el rendimiento de una aplicación sin necesidad de contar con múltiples equipos. Los *SDKs* proporciona bibliotecas y herramientas especificas para cada plataforma, como *Android SDK o iOS SDFK*, necesarias para compilar y ejecutar el código. Estas herramientas permiten al desarrollador accederá a funciones del sistema operativo, como sensores, cámara, notificaciones o almacenamiento.

Por otra parte los *plugins* amplían la funcionalidad de entornos como *VSCode e Ionic*. Algunos permiten integraciones con plataformas externas, como *Firebase*, mientras que otros ofrecen soporte para lenguajes, linters, snippets de código o conectividad con bases de datos. Todo esto mejora significativamente la eficiencia del desarrollo. Según la documentación oficial de *Android Studio [8] (Google Developers)"El uso de herramientas SDKs y emuladores permite simular escenarios reales, ahorrar tiempo y asegurar la calidad del software antes del despliegue."*

2.7 Servicios Web y Conectividad

Las aplicaciones móviles modernas no solo deben funcionar de forma autónoma, sino que también necesitan interactuar con servicios externos para obtener o enviar información: esta comunicación se puede lograr principalmente mediante servicios web que permiten la conexión entre el cliente (Aplicación móvil) y el servidor, donde se residen los datos o funciones adicionales.

2.7.1 Consumo de APIs REST usando Retrofit, Volley

Uno de los enfoques más comunes para establecer esta comunicación es a través de *APIs REST* que permiten acceder a servicios usando el protocolo *HTTP*. Estas *APIs* devuelven datos estructurados, generalmente en formato *JSON*, que la aplicación puede procesar y mostrar al usuario. Para facilitar esta tarea, existen bibliotecas como *Retrofit* que son ampliamente utilizadas en *Android*, que permite definir interfaces que representan los *endpoints* del servicio, haciendo que las llamadas sean más limpias y fáciles de mantener. También, otra de las alternativas puede ser *Volley*, enfocada en el manejo eficiente de peticiones *HTTP*, aunque con enfoque más detallado sobre la gestión de respuestas y errores.

Como señala la documentación oficial de Retrofit [22], las librerías como Retrofit y Volley facilitan significativamente la conexión entre aplicaciones móviles y servicios web, lo que permite integraciones más eficientes y robustas.

2.7.2 Formatos de intercambio: JSON, XML

Los datos que se transmiten entre la aplicación y el servidor suelen estar estructurados en formatos estándar. *JSON (JavaScript Object Notation)* es el más usado actualmente por su ligereza, facilidad de lecturas y compatibilidad con lenguajes modernos. Su sintaxis es sencilla y se adapta al trabajo con objetos en *JavaScript y React*. *XML (Extensible Markup Language)*, aunque más antiguo y más verboso que *JSON*, aún se

utiliza en algunas APLs, sobre todo en sistemas heredados o cuando se requiere una estructura más rígida. La elección entre utilizar *JSON* o *XML* depende del sistema y sus necesidades de interoperabilidad, aunque en la actualidad *JSON* es el formato más utilizado en aplicaciones móviles por su ligereza y facilidad para manejar datos.

2.7.3 Seguridad en la transmisión de datos

Al trabajar con servicios web, es indispensable proteger la información que se transmite, especialmente si incluye datos sensibles como contraseñas o información persona del usuario. Para ello, se utilizan protocolos de seguridad como *HTTPS*, que cifra lo datos en tránsito mediante *SSL/TLS.*

Es recomendable implementar mecanismos de autentificación, como *Tokens* de accesos y validaciones en el servidor para prevenir ataques como la suplantación de identidad o la inyección de código. Al respecto, Jitterbit [11] señala que *"incorporar la seguridad en el ciclo de vida del desarrollo de aplicaciones desde el principio es la mejor manera de garantizar la protección de su aplicación."* Destacando la importancia de considerar la seguridad como un requisito esencial desde las primeras etapas del desarrollo móvil.

2.8 Testing y Validación de Aplicaciones

El proceso de desarrollo de *Software* no puede considerarse completo sin la etapa de pruebas o *testing*, ya que está garantizada que la aplicación cumple con los requerimientos funcionales y no funcionales. En las aplicaciones móviles, donde las condiciones de ejecución varían por dispositivo, sistemas operativo y red, las pruebas adquieren una importación aun mayor.

2.8.1 Tipos de pruebas: unitarias, instrumentadas, funcionales

Las pruebas se pueden clasificar según su enfoque y nivel de profundidad, las pruebas unitarias se centran en verificar bloques pequeños y aislados de código, como funciones o clases. En cambio, las pruebas instrumentarías se ejecutan en un entorno que simula el comportamiento del dispositivo, permitiendo observar cómo actúa la aplicación en condiciones reales.

Las pruebas funcionales validan que la aplicación cumpla con lo especificado por el cliente o usuario, garantizando que la interfaz, lógica y flujo de navegación respondan correctamente ante distintos escenarios.

Beizer [2] subraya que *"las pruebas no deben limitarse a buscar errores, sino a demostrar que el sistema funciona correctamente bajo condiciones esperadas e inesperadas."*

2.8.2 Herramientas comunes: JUnit, Espresso

Para facilitar las pruebas en entornos móviles, se han desarrollado diversas herramientas como *JUnit* esta es una de las más empleadas para pruebas unitarias en *Java*, permitiendo automatizar la validación de métodos y estructuras lógicas. En el caso de aplicaciones *Android*, *Espresso* se ha convertido en un estándar para pruebas de interfaz, ya que permite simular interacciones del usuario como toques, desplazamientos y entradas de texto.

Estas herramientas ayudan a construir suites de pruebas repetibles que permiten verificar el estado funcional de la aplicación tras cada modificación, reduciendo los errores que podrán pasar desapercibidos en pruebas manuales.

Según Kaner, Falk y Nguyen [12]. *"Las herramientas de prueba automatizada no reemplazan al ingeniero de pruebas, pero amplifican su capacidad para detectar defectos y evaluar la calidad del software."*

2.9 Desarrollo

El desarrollo de esta aplicación móvil para la gestión de horarios escolares se llevó a cabo mediante un enfoque iterativo e incremental, apoyando en la metodología *Scrum*, una de las más utilizadas en el desarrollo ágil de *Software*. Esta metodología permitió dividir el proyecto en entregas pequeñas y funcionales denominadas *sprints*, lo que facilitó la planificación, el seguimiento y la mejora continua a lo largo de todo el ciclo de vida del proyecto.

2.9.1 Metodología de trabajo *Scrum*

Scrum es un marco de trabajo ágil que facilita el desarrollo de productos complejos mediante entregas parciales que agregan valor progresivamente. En este proyecto se organizaron los *sprints* a lo largo de un periodo total de duración de cinco meses, con reuniones de planificación, revisión y retrospectiva al finalizar cada ciclo. Esto permitió recibir retroalimentación constante, realizar ajustes oportunos y mantener una visión clara de los objetivos a corto y largo plazo.

De acuerdo con Sommerville [21], *"las metodologías ágiles como Scrum permiten a los equipos responder de forma flexible a los cambios de requerimientos y entregan software funcional de forma temprana."*

Durante cada *sprints* se asignaron tareas específicas como el diseño de la interfaz, la integración de componentes, la lectura del PDF con los horarios, la detección de conflictos y la validación de los datos. Al finalizar cada *sprints* se evaluaba el estado de avance con base en los objetivos definidos en la planificación.

2.9.2 Análisis de requerimientos

El primer paso fue comprender a fondo las necesidades del usuario, se realizaron sesiones de análisis para identificar los requerimientos funcionales (acciones que la aplicación debe ejecutar, como carga de un archivo *PDF*, visualizar horarios, seleccionar materias)

y los no funcionales (rendimiento, usabilidad, compatibilidad, almacenamiento local.)

Este análisis fue clave para definir el alcance del proyecto y establecer prioridades. Según Pressman [16], *"los requerimientos constituyen la base para todas las actividades posteriores de diseño, codificación y prueba. Una omisión a ambigüedad en esta fase puede propagarse con efectos negativos en todo el sistema."*

2.9.3 Diseño del sistema

Una vez comprendidos los requerimientos, se procedió a diseñar la arquitectura de la aplicación. Se adoptó una arquitectura modular, dividiendo el sistema en componentes independientes que se comunican entre sí. Está decisión permitió facilitar el mantenimiento y la escalabilidad. El diseño incluyó tanto la estructura lógica (componentes, servicios, rutas, estados) como la estructura visual (interfaz gráfica con *Ionic*). También se definieron las rutas de navegación, las interfaces para la selección de materias y la manera de mostrar los horarios por día y turno. Según Booch. [3] *"Un buen diseño no es aquel que solo funciona, sino aquel que puede crecer, adaptarse y sobrevivir en entornos cambiantes."*

2.9.4 Implementación

Durante la etapa de implementación, se comenzó la codificación del sistema utilizando *React* junto con *Ionic*, lo que permitió combinar la potencia de una biblioteca moderna de *JavaScript* con un *Framework* especializado en desarrollos para móvil. El desarrollo se realizó en *TypeScript*, considerando sus beneficios para el control de tipos y la escalabilidad del proyecto.

PERIODO	SEXTO TM				LUN.	MAR.	MIER.	JUE.	VIER.	Lab/Aula
CLAVE MATERIA	UNIDAD DE APRENDIZAJE	CRED.	HORAS							
			H.T	H.P.						
LINC23	Compiladores (TM)	7	3	1		9-11		9-11		LPelS/B 6
PROFESOR	M. en C. María Guadalupe Domínguez Urban	RFC	DOUG840323							
		N.E.								
CLAVE MATERIA	UNIDAD DE APRENDIZAJE	CRED.	HORAS							
			H.T	H.P.						
LINC39	Sistemas analógicos (TM)	7	3	1			9-11		9-11	LEyAC/ B6
PROFESOR	Ing. Alejandro Quintero García	RFC	QUGA810422							
		N.E.								
CLAVE MATERIA	UNIDAD DE APRENDIZAJE	CRED.	HORAS							
			H.T	H.P.						
LINC36	Protocolos de Comunicaciones de Datos (TM)	7	3	1	11-13		11-13			LEyAC/ B6
PROFESOR	Dr. en T.I.E. Jorge Bautista López	RFC	BALJ751215							
		N.E.								
CLAVE MATERIA	UNIDAD DE APRENDIZAJE	CRED.	HORAS							
			H.T	H.P.						
LINC27	Ingeniería de Software II (TM)	7	3	1			7-9		7-9	LPelS/B 6
PROFESOR	Dra. Rosa Erendira Reyes Luna	RFC	RELR850625							
		N.E.								
CLAVE MATERIA	UNIDAD DE APRENDIZAJE	CRED.	HORAS							
			H.T	H.P.						
LINC17	Administración de recursos informaticos (TM)	7	3	1		11-13		11-13		LRySD/ B6
PROFESOR	Ing. Nadia Yanet Julia López Villegas	RFC	LOVN850209							
		N.E.								
CLAVE MATERIA	UNIDAD DE APRENDIZAJE	CRED.	HORAS							
			H.T	H.P.						
LINC34	Procesamiento de imágenes digitales (TM)	7	3	1	9-11			7-9		LPelS/B 6
PROFESOR	M. en C. Edith Cristina Herrera Luna	RFC	HELE831015							
		N.E.								

Figura 2.7: Ejemplo de archivo *PDF* con horarios escolares.

La Figura 2.7 muestra un ejemplo del archivo *PDF* original que contiene los horarios escolares correspondientes al sexto semestre en turno matutino, el cual es utilizado como entrada para el procesamiento de la información en la aplicación.

Entre los componentes desarrollados destacan:

- **Carga y procesamiento de archivos PDF**: Permite al usuario subir un archivo de horarios desde su dispositivo. La aplicación interpreta su contenido y extrae las materias, días y horas de clases. Se desarrollo una lógica que permitiera leer, interpreta y estructurar la información contenida en el *PDF.*

- **Interfaz de selección de materias**: Muestra todas las materias disponibles organizadas por semestres, permitiendo seleccionarlas con un *clic*. Al seleccionar una, la aplicación resalta automáticamente los horarios correspondientes con un color diferente.

- **Validación de conflictos**: Al seleccionar una materia, la aplicación verifica si existe un cruce con otra ya seleccionada. En caso de conflicto, se muestra un mensaje emergente al usuario indicado el problema.

- **Persistencia de datos**: Se utilizaron soluciones de almacenamiento local como *localStorage* para guardar temporalmente las selecciones del usuario. Esto garantiza que las materias elegidas permanezcan visibles incluso si el usuario cambia de semestre o recarga la aplicación.

2.9.5 Pruebas de funcionalidad y corrección

Una vez implementadas las principales funciones, se realizaron diversas pruebas para verificar que la aplicación cumpliera con los objetos definidos. Estas incluyeron:

- Pruebas unitarias para asegurar que cada componente del sistema funcione de forma aislada.
- Pruebas funcionales para evaluar si la aplicación responde correctamente a las acciones del usuario (Subir *PDF*, seleccionar materias, detectar conflictos).
- Pruebas de compatibilidad ejecutando la aplicación en distintos dispositivos *Android* para garantizar un comportamiento estable.

Se tomaron en cuenta las buenas prácticas de pruebas descritas por *McConnell* [13], quien señala que *"la prueba sistemática es un proceso iterativo que permite descubrir errores en las etapas más tempranas, evitando costos mayores en el futuro."*

2.9.6 Iteración y mejora continua

Gracias al enfoque iterativo de *Scrum*, se fue mejorando la aplicación en cada ciclo. La retroalimentación obtenida al finalizar cada *sprints* permitió ajustar la interfaz, mejorar la lógica de procesamiento *PDF*, corregir errores en la validación de horarios y optimizar el rendimiento en móviles de gama media y baja. Este modelo de trabajo ágil resultó especialmente efectivo para el tipo de aplicaciones que se desarrollo, ya que se ajustó con facilidad a los cambios y mantuvo siempre un producto funcional al finalizar cada iteración.

Para ilustrar de forma concreta cómo se implementó parte de la funcionalidad de la aplicación, en esta sección se describe el desarrollo del componente encargado de la extracción y visualización de los horarios a partir de un archivo *PDF*. Esta funcionalidad representa uno de los pilares de la aplicación, ya que permite al usuario transformar automáticamente el contenido del documento en un horario digital personalizado.

2.9.7 Carga y lectura del archivo PDF

El proceso comienza con la interfaz gráfica que permite al usuario seleccionar un archivo *PDF* desde su dispositivo. Este archivo contiene los horarios organizados por semestre, incluyendo materias, docentes, días de la semana, horas y aulas. Para asegurar una experiencia fluida, se diseñó una interfaz intuitiva, utilizando los componentes de *Ionic* para móviles.

Figura 2.8: Carga y lectura del archivo *PDF*.

La figura 2.8 muestra la pantalla inicial de la aplicación, donde el estudiante puede subir su archivo *PDF* que contiene el horario. La interfaz incluye un ícono representativo acompañado del texto "Sube tu PDF de ICO" y una instrucción que indica al usuario que debe cargar el archivo para comenzar. Asimismo, incorpora un botón de selección de archivo y un mensaje de estado que informa si aún no se ha elegido ningún documento.

Una vez cargado el archivo, se ejecuta un proceso de análisis que interpreta el contenido del *PDF*. A diferencia de una lectura plana o secuencial, el sistema implementa un algoritmo de detección de patrones espaciales, que analiza las

coordenadas del texto para determinar en que columna (día) y fila (hora) se encuentra cada clase. Esta lógica fue desarrollada teniendo en cuenta que los archivos *PDF* no almacena la información en una estructura tabular visible, sino como bloques de texto con coordenadas X e Y.

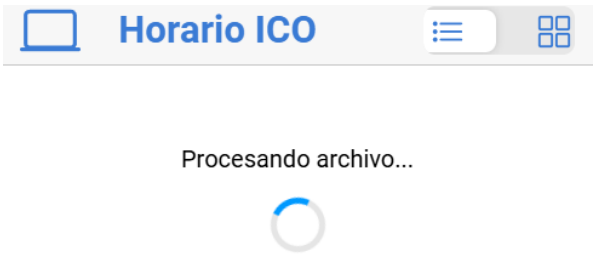

Figura 2.9: Procesamiento del archivo *PDF*.

La figura 2.9 muestra el estado de la aplicación durante el procesamiento del archivo PDF. En esta etapa, el sistema interpreta la información contenida en el documento, prepara la extracción de los horarios y organiza los datos para que puedan visualizarse en el calendario interactivo. Está retroalimentación visual permite al usuario identificar que su archivo está siendo analizado correctamente antes de continuar con el uso de la aplicación.

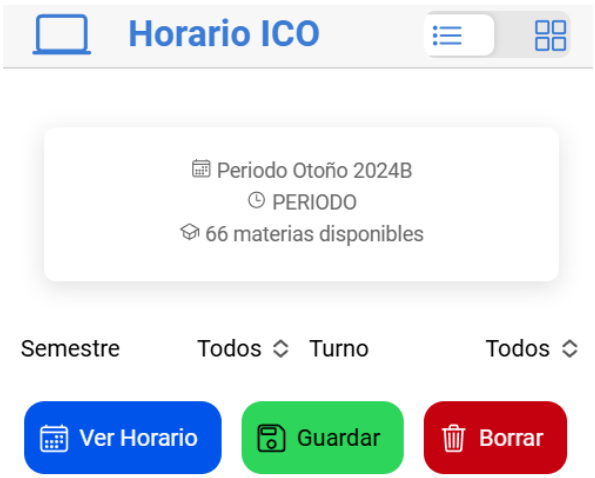

Figura 2.10: Interfaz principal del sistema de horarios académicos.

La figura 2.10 muestra la interfaz principal del sistema de horarios académicos de la carrera de Ingeniería en Computación (ICO). En está pantalla se visualiza el título *"Horario ICO"* acompañado de un ícono alusivo, así como la información del período académico (*Otoño 2024B*) y la cantidad total de materias disponibles. Además, se incluyen filtros para seleccionar el semestre y el turno, junto con botones de acción que permiten ver el horario, guardar los cambios o borrar la selección. Está interfaz constituye el punto central de interacción con el sistema después de haber procesado el archivo *PDF*.

2.9.8 Organización y validación de datos

Tras la lectura de las tablas, el sistema organiza la información en estructuras de datos internas, agrupando cada materia con sus respectivos días y horarios. El formato tabular reduce el margen de error, se implementa un filtro adicional que valida que cada materia solo tenga un máximo de dos horarios semanales, evitando asignaciones duplicadas o erróneas. Se eliminaron espacios vacíos o campos no relevantes que puedan interferir con la correcta visualización del horario. Todo esto se realiza de manera automática, con el objetivo de que el usuario solo tenga que cargar su archivo una vez y obtener un horario limpio y organizado.

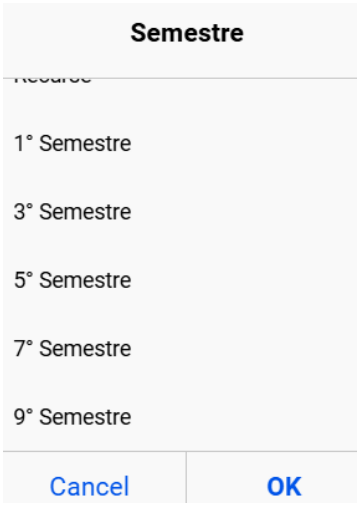

Figura 2.11: Organización y validación de datos.

La figura 2.11 presenta la interfaz destinada a la selección de semestre en el sistema de horarios de la carrera de Ingeniería en Computación (ICO). Está ventana emergente muestra un listado de opciones que incluye los semestres impares y la alternativa *"Todos"*, lo que permite filtrar las materias de acuerdo con el nivel académico del estudiante. Asimismo, la interfaz incorpora dos botones de acción, *"Cancel"* y *"OK"*, que confirman o descartan la elección del usuario. Esta funcionalidad resulta esencial para la organización y validación de los datos, ya que posibilita ajustar la vista del horario según el semestre seleccionado.

Como menciona Sommerville. [21] *"La validación de entradas y estructuras intermedias es una practica fundamental para asegurar la integridad del sistema en tiempos de ejecución."*

2.9.9 Visualización y selección de materias

Después de revisar y organizar la información, el sistema genera automáticamente una tabla visual usando el componente Horario.tsx. La tabla emplea una estructura matricial donde las columnas representan los días de la semana y las filas corresponden a las franjas horarias, facilitando la identificación visual de cada sesión de clase. Cuando el usuario selecciona una materia, la celda corresponde se llena con el nombre y se

resalta con un color para que sea fácil de ubicar. Este diseño busca que el usuario vea de forma clara que espacios ya están ocupados y cuales siguen disponibles. Por ejemplo si se elige una materia por la mañana, el sistema quita automáticamente esa misma materia del turno de la tarde para evitar confusiones o duplicados. Además, si el usuario intenta seleccionar una asignatura que se cruza con otra ya elegida, aparece un aviso que informa del conflicto y bloquea la selección, así mismo le pide que seleccione una de las dos materias. Esta lógica ayuda a que todo fluya de manera más clara y evita errores comunes al armar horarios. Como bien menciona Galitz. [6] *"La claridad y la prevención de errores son elementos cruciales en toda interfaz gráfica y eficiente."* Este mensaje esta alineado con principios fundamentales de diseño de interfaces que resalta en distintas secciones del libro donde enfatiza que una interfaz debe ser clara, comprensible y diseñada para evitar errores del usuario desde su concepción.

Figura 2.12: Visualización y selección de Materias.

La figura 2.12 muestra la interfaz principal del horario semanal. En la parte superior se presenta el título *"Horario ICO"*, acompañado de un ícono representativo. Las materias se visualizan inicialmente en tonos de color claros, lo que indica que se encuentran disponibles para selección. Al seleccionar una materia, el sistema mantiene el mismo color, pero en un tono más intenso, permitiendo al usuario identificar de forma visual y clara las materias que han sido agregadas al horario.

Figura 2.13: Selección de Materias.

La figura 2.13 muestra una vista expandida del horario semanal del sistema *"Horario ICO"*, en la que se visualizan los días martes, miércoles, jueves y viernes. En la parte

superior se presenta el título de la aplicación acompañado de un ícono representativo. Las materias se muestran inicialmente en tonos de color claros, tal como se observa en la figura 2.12, lo que indica que se encuentran disponibles en distintos días y horarios; al seleccionar una materia, esta conserva el mismo color, pero en un tono más intenso, permitiendo identificar de forma clara las asignaturas que han sido agregadas al horario.

2.9.10 Persistencia y cambio de semestre

Para mejorar la experiencia de uso, el sistema guarda las materias seleccionadas en almacenamiento local (*localStorage*). Esto permite que incluso si el usuario cambia de semestre, las materias seleccionadas previamente se sigan mostrando en su horario.

Figura 2.14: LocalStorege.

La figura 2.14 muestra un cuadro de diálogo modal titulado *"Confirmar acción"*, en el cual se presenta la pregunta "¿Qué deseas hacer?" junto con cuatro opciones: Cancelar, Subir nuevo PDF, Borrar selección y Reiniciar todo. Este

componente permite al usuario gestionar los datos almacenados localmente, ya sea cancelando la operación, eliminando únicamente las materias seleccionadas, reiniciando completamente el sistema o cargando un nuevo archivo PDF.

Cuando el usuario cambia de semestre, el sistema respeta los horarios previamente seleccionados y únicamente ubica las materias nuevas en el horario al momento de ser seleccionadas, sin modificar las materias previamente registradas. Esto evita conflictos y da al estudiante un mayor control sobre su carga académica, esta funcionalidad responde al principio de iteraciones constante que forma parte del enfoque de desarrollo utilizado.

2.9.11 Diagrama de clases

La figura 2.15 presenta un diagrama de clases UML del sistema de horarios ICO, mostrando las clases UsuarioSeleccion, GestorSemestres, Semestre, Materia y Horario, junto con sus atributos, métodos y relaciones entre ellas, como utiliza, gestiona, selecciona, agrupar y contiene. Este diagrama permite visualizar la estructura y la interacción entre los componentes principales del sistema.

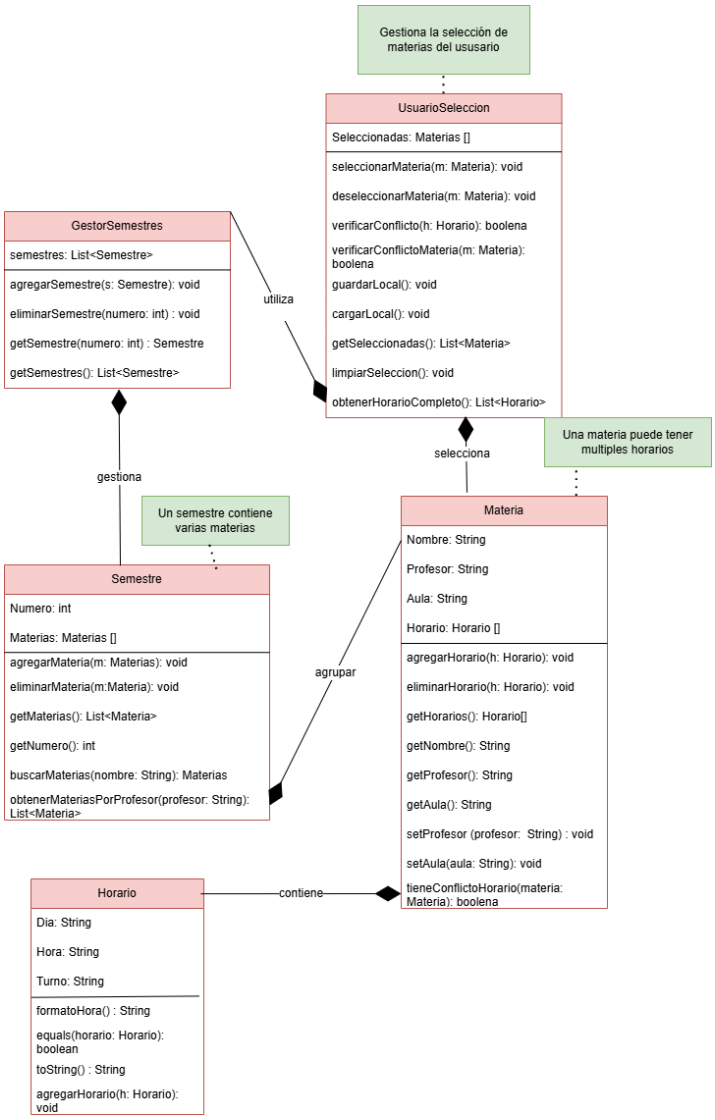

Figura 2.15: Diagrama de Clases.

Explicación del diagrama:

- **Materia** Representa una asignatura, contiene el nombre, el profesor y el aula. Cada materia puede tener uno o más horarios, ya que algunas materias tienen clases en diferentes días u horarios. Se pensó así porque hay materias que se repiten en distintos bloques, entonces se necesitaba que una misma materia tuviera varios horarios posibles.

- **Horario**: Indica el día, hora y turno en que se imparten una clase, esta asociado

a una materia. Esta clase esta relacionada directamente con la clase Materia, inicialmente se había hecho con *strings* sueltos, pero luego se observo que era mejor agrupar esta información en una clase separada para tener más control sobre los datos del horario.

- **Semestre**: Agrupa todas las materias de un periodo escolar, permite añadir o eliminar asignaturas de forma dinámica, según lo que el usuario seleccione, esta parte del desarrollo fue complicada por que al inicio las materias no se actualizaban bien al cambiar de semestre, pero después de revisar como se estaba manejando el estado, se soluciono y quedo funcionando correctamente.

- **UsuarioSeleccion**: Administra las materias seleccionadas por el usuario, también se encarga de que estas selecciones se guarden correctamente en el almacenamiento local para que cuando el usuario vuelva a entrar a la aplicación, sus materias sigan ahí. En esta parte se complico un poco implementar la persistencia porque se presentaron problemas al usar *localStorage* con algunos objetos, pero después, se resolvió convirtiéndolos a *JSON* antes de guardarlos.

- **GestorSemestres**: Administra las materias seleccionadas por el usuario entre los diferentes semestres, permite agregar y eliminar materias, y verificar que no haya duplicados o conflictos entre los horarios. Fue una clase que se añadió después, por que al inicio todo estaba muy junto y se dificultaba controlar los cambios de un semestre a otro, entonces se decidió separar esta lógica en una nueva clase.

Este diseño modular permite un mantenimiento más ágil del sistema, ya que cada clase tiene responsabilidades bien definidas. La relación entre clases esta basadas en una estructura jerárquica que sigue el principio de responsabilidad única.

Según Rumbaugh et al, [18] *"el modelado de clases no solo permite comprender la estructura del sistema, sino anticipar su comportamiento e interacción antes de codificar."* Esta idea representa con fidelidad los principios del *Object Modeling Technique (OMT)* presentados en su libro, donde se enfatiza que el modelado facilita

la visualización del comportamiento del sistema y la simulación de interacciones como etapas previas a la implementación.

Capítulo 3

Desarrollo del sistema

En este capitulo se va a describir como se desarrolla la aplicación, desde que se pensó en la idea hasta que ya estuvo lista para usarse. La finalidad es mostrar de forma clara las partes que se fueron haciendo y como se revolvieron algunos problemas que aparecieron en el camino, no se trata de poner el código, sino explicar por que se hicieron las cosas así, que herramientas se usaron y también las decisiones que se tomaron.

La aplicación que se hizo es para que los estudiantes puedan subir su horario en *PDF* y que el sistema lo organice, detecte conflictos y permita elegir materias de forma fácil. Entonces aquí se va a ir contando paso a paso, sin olvidar las pruebas que se hicieron para ver que todo funciona como debería.

3.1 Metodología de desarrollo

Se optó por la metodología ágil *Scrum* debido a que permite dividir el desarrollo del sistema en iteraciones pequeñas y controladas, facilitando la revisión periódica del avance del proyecto. *Scrum* es una metodología ágil ampliamente utilizada cuando se requiere trabajar de manera incremental y realizar evaluaciones constantes, ya que posibilita detectar errores de forma temprana y realizar ajustes oportunos sin necesidad de esperar a la finalización completa del proyecto.

Scrum trabaja con iteraciones llamadas *sprints*, que en este caso fueron cortos,

por que en el desarrollo de la aplicación tenia que avanzar rápido. En cada *sprint* se revisaban los avances y se ajustaban las tareas, esto permitió que la aplicación se fuera desarrollando poco a poco, desde la interfaz, hasta la lógica para leer el *PDF* y manejar el horario.

Según Schwaber [20], esta metodología *"proporciona un marco de trabajo en el que las personas pueden abordar problemas complejos adaptativos, al mismo tiempo que entreguen productos de la más alta calidad posible de forma productiva y creativa."* Esto fue muy útil porque el proyecto no estaba del todo definido al inicio y se fueron descubriendo detalles sobre la marcha.

3.2 Herramientas y tecnologías utilizadas

Durante el desarrollo de la aplicación se seleccionaron diversas herramientas y tecnologías con el objetivo de facilitar la implementación del sistema. Algunas de ellas ya eran conocidas, mientras que otras se eligieron debido a que se ajustaban a los requerimientos funcionales del proyecto. Si bien no todas las herramientas resultaron sencillas de utilizar inicialmente, su adopción permitió optimizar el proceso de desarrollo y fortalecer las competencias técnicas necesarias para la correcta implementación de la aplicación.

La tecnología principal fue *Ionic React*, es un *Framework* que permite hacer aplicaciones móviles con *React*, se selecciono por que *Ionic* permite codificar solo una vez y funciona tanto en *Android* como en *iOS*, no es necesario escribir un código separado para cada sistema. Además *Ionic* tiene muchos componentes, como botones, listas y modales, lo que realmente ayuda a construir una interfaz solida. *React* hace que sea más fácil reutilizar y administrar componentes según el estado de la aplicación, que es muy útil ya que el horario se adapta a las opciones de usuario.

También se usa *TypeScript* que es una experiencia de *JavaScript* que le permite especificar tipos para variables y funciones, esto fue muy útil para detectar esos errores astutos en *JavaScript* regular gracias a *TypeScript*, durante la implementación del

código para la lectura de archivos *PDF* y la gestión de los horarios, se detectaron errores antes de la ejecución de la aplicación, lo que permitió un ahorro considerable de tiempo durante el proceso de desarrollo.

Para el almacenamiento de la información no fue necesario utilizar una base de datos compleja; en su lugar, se empleó *LocalStorage*, una característica de navegador incorporada que almacena datos en pares de valores clave. Así los usuarios permanecen con el periodo y la fecha actual, entonces cuando se vuelve a abrir la aplicación, su horario aparece listo para comenzar, no es necesario configurarla nuevamente al menos que el usuario lo quiera hacer.

El código 3.1 presenta un fragmento del componente *Horario.tsx*, en el cual se implementa el almacenamiento local de la información del horario mediante *LocalStorage*.

```
localStorage.setItem(STORAGE_KEY, JSON.stringify(materias));
localStorage.setItem(PERIODO_KEY, periodoActual);
localStorage.setItem(FECHA_KEY, fechaActual);
alert("Horario guardado correctamente!");
```

Código 3.1: Fragmento de Horario.tsx
LocalStorage.

Este fragmento de código 3.1 permite guardar localmente las materias seleccionadas, el periodo académico y la fecha actual, utilizando pares clave–valor. Al finalizar el proceso, se muestra un mensaje de confirmación al usuario, indicando que el horario ha sido almacenado correctamente. Este mecanismo elimina la necesidad de conexión a internet y agiliza el acceso a la información al reabrir la aplicación.

Otra tecnológica clave era la biblioteca para leer archivos *PDF*, se necesitaba extraer los horarios de manera automática, no manual, por lo que se busco una biblioteca que pudiera procesar tablas dentro del *PDF*, esta parte fue un poco más difícil por que en las pruebas algunos *PDFs* tenían formatos distintos y fue necesario escribir lógica adicional para organizar correctamente las materias, los días y las horas. Esta sección es clave en todo el trabajo del sistema, sin ella, no se podría hacer la extracción de datos

bien, en otras palabras, si no se configuraban correctamente la biblioteca, el sistema simplemente no podía mostrar ni procesar el *PDF*.

El código 3.2 presenta la configuración del *worker* de *pdf.js* utilizada dentro del componente *Horario.tsx*, necesaria para el procesamiento adecuado de archivos *PDF* en la aplicación.

```
pdfjsLib.GlobalWorkerOptions.workerSrc = pdfWorker;
GlobalWorkerOptions.workerSrc = `//cdnjs.cloudflare.com/ajax/libs/pdf.js/${
    (window as any).pdfjsLib?.version || '2.11.338'
}/pdf.worker.min.js`;
```

Código 3.2: Fragmento de Horario.tsx

Worker.

Esta configuración permite establecer correctamente el archivo *worker* encargado de ejecutar el procesamiento de los documentos *PDF* en segundo plano. Gracias a ello, la aplicación puede cargar e interpretar los archivos de forma eficiente, evitando bloqueos en la interfaz de usuario y asegurando el correcto funcionamiento de las funcionalidades posteriores relacionadas con la extracción de información.

Se utilizó *Visual Studio* como entorno de desarrollo para la implementación del sistema y *Git* para realizar un seguimiento de los cambios de versiones. *VS Code* es fácil de usar, tiene muchas extensiones que ayudan a programar en *Ionic* y *TypeScript*, *Git* permitió llevar un historial de cambios, fue útil porque en varias ocasiones hubo que volver a versiones anteriores del código.

Entonces, básicamente el uso de estas tecnologías y herramientas hizo que el proyecto fuera mucho más organizado, eficiente y seguro. Todos tenían un papel clave que jugar y sin ellos, conseguir que la aplicación funcione sin problemas seria mucho más complicado.

3.3 Proceso de desarrollo

El desarrollo de la aplicación siguió un enfoque iterativo dividido en cuatro etapas principales, las cuales son: análisis de requisitos, diseño de interfaz, implementación de funcionalidades core y validación del sistema.

El desarrollo del sistema se realizó de manera incremental, iniciando con la implementación de las funcionalidades básicas y posteriormente incorporando los componentes de mayor complejidad. Durante este proceso se identificaron diversos inconvenientes que requirieron ajustes en la implementación; sin embargo, estas modificaciones contribuyeron a mejorar la estabilidad y el funcionamiento general del sistema.

La primera etapa del desarrollo fue el análisis de requisitos, el cual consistió en identificar las funcionalidades que debía cumplir la aplicación. Entre los aspectos más relevantes se estableció que el sistema debía ser capaz de leer archivos de horarios en formato *PDF*, mostrar las materias de manera organizada, permitir la selección de asignaturas sin generar conflictos de horario y almacenar el resultado para su posterior consulta. Estos requisitos sirvieron como base y guía para las etapas posteriores del desarrollo.

Posteriormente, se abordó la etapa de diseño de la interfaz de usuario, en la cual se definieron las pantallas principales de la aplicación, tales como la vista para la carga de archivos *PDF*, la visualización del horario organizado por días y horas, así como los controles para seleccionar o eliminar materias. Para esta etapa se utilizó el *Framework Ionic*, el cual proporciona componentes gráficos predefinidos y personalizables, lo que facilitó el desarrollo de la interfaz y permitió optimizar el tiempo de implementación al adaptar estilos existentes a los requerimientos del sistema.

Una vez definida la interfaz de usuario, se procedió a la implementación del proceso de extracción de datos desde los archivos *PDF*, el cual representó una de las etapas más complejas del desarrollo. Esto se debió a que los horarios no siempre presentaban una estructura uniforme, ya que algunos documentos contenían formatos irregulares

que dificultaban la identificación correcta de días y horas. Para atender esta situación, fue necesario implementar lógica adicional orientada a la limpieza y validación de los datos, asegurando que cada materia contara únicamente con dos días de clase y que los horarios no presentaran inconsistencias. Esta etapa resultó fundamental, ya que una extracción incorrecta de la información afectaría directamente la correcta generación del horario y la coherencia del sistema.

Una vez que la información del archivo *PDF* pudo visualizarse correctamente, se continuó con la implementación de la lógica de validación de horarios. En esta etapa se programó que, si el usuario intenta seleccionar una materia que se superpone con otra previamente elegida, el sistema muestre una ventana emergente notificando la existencia de un conflicto. Esta validación evita que el estudiante genere un horario incompatible o imposible de cursar. Asimismo, se implementó una restricción adicional que impide seleccionar una misma materia en distintos turnos: al elegirla en el turno matutino, se desmarca automáticamente en el turno vespertino, y viceversa, garantizando que cada materia se curse únicamente en un turno.

Otra parte importante fue la de almacenamiento, para eso se uso *LocalStorage*, guardando las materias seleccionadas, el periodo y la fecha, de esta forma, aunque el usuario cerrara la aplicación, podía volver después y seguir viendo su horario, esto le da un sentido más practico por que no se pierde la información, al menos que el usuario decida que si.

Finalmente, se realizaron pruebas básicas con el objetivo de verificar el correcto funcionamiento del sistema, las pruebas consistieron en cargar diferentes horarios en archivos *PDF*, seleccionar materias de varios semestres, confirmar que no hubiera choques ni errores al guardar o mostrar el horario, algunas veces aparecían problemas como el que una materia se asignaba más días de los que debía, pero poco a poco se corrigieron esos detalles hasta que el sistema quedo estable. En conclusión el proceso de desarrollo fue un camino de varias etapas.

- Análisis.

- Diseño.
- Extracción de datos.
- Validación de horarios.
- Almacenamiento y pruebas.

No siempre fue lineal, por que en ocasiones se tenia que regresar en cosas que ya se habían realizado, pero eso forma parte normal del trabajo de programación, al final se logro tener una aplicación que cumple con lo que se había planteado desde el inicio.

3.4 Implementación

La implementación del sistema se fue haciendo en módulos, cada uno encargado de una parte importante de la aplicación, no se hizo todo junto por que era difícil de manejar y mejor se fue dividiendo, a continuación se describen las partes relevantes y como se programaron.

3.4.1 Carga del archivo *PDF*

Lo primero que se implementó fue la función para que el usuario pudiera subir su horario en formato *PDF*, esta parte fue esencial por que sin el archivo no había nada que procesar, se tuvo que configurar la librería *pdf.js* para que pudiera leer los archivos directamente en el navegador y luego se preparo el código que detecta cuando el usuario seleccione un archivo desde su dispositivo.

El código 3.3 muestra el fragmento de código que hace la detección del archivo se muestra a continuación.

```
1 const handleFileChange = (event: React.ChangeEvent<HTMLInputElement>) => {
2     const selectedFile = event.target.files?.[0];
3     if (selectedFile) {
4         setFile(selectedFile);
```

```
        setIsLoading(true);
        processPDF(selectedFile);
    }
};
```

Código 3.3: Fragmento de Horario.tsx

Detección del archivo.

Este código muestra como se verifica si el usuario ha seleccionado un archivo, en caso de que sea afirmativo, se almacena en el estado correspondiente y se inicia el proceso de lectura del *PDF* mediante la función *processPDF*.

En este instante, la función *handFileChange* se pone en marcha solo cuando el usuario elige un archivo en el espacio para subir documentos, dentro de ella se guarda el archivo en la parte del programa que guarda datos *setFile*, se activa una luz para decir que el archivo se esta cargando *setLoading(true)*, y después se lleva el archivo a la función *processPDF* para empezar el análisis.

El código 3.4 muestra el *input* en *TSX* que deja al usuario escoger el archivo *PDF*.

```
<input
    type="file"
    accept=".pdf"
    onChange={handleFileChange}
    className="file-input"
/>
```

Código 3.4: Fragmento de Horario.tsx

Permite al usuario elegir el archivo.

En este código 3.4, el *input* esta configurado para aceptar unicamente archivos *PDF* para que se active la función *handleFileChange* cuando el usuario selecciona un archivo, asegurando que el proceso de análisis se inicie correctamente.

Una vez detectado el archivo, se debe cargar en memoria y leerlo con la librería *pdf.js*, para esto se utilizo la función *processPDF*, que se encarga de recibir el archivo, convertirlo en un *ArrayBuffer* y luego abrirlo con *getDocument*, esto permite que el

sistema pueda recorrer las paginas del *PDF* y empezar a extraer la información.

El código 3.5 muestra la lógica principal de esta fase.

```
const processPDF = async (pdfFile: File) => {
    const reader = new FileReader();
    reader.onload = async (e) => {
        const arrayBuffer = e.target?.result as ArrayBuffer;
        const pdf = await getDocument({ data: arrayBuffer }).promise;
        // Estructuras iniciales para almacenar la informacin
        const materiasExtraidas: Materia[] = [];
        const semestresList: Set<string> = new Set(["Todos", "Recurse"]);
        // Recorre cada pgina del documento
        for (let i = 1; i <= pdf.numPages; i++) {
            const page = await pdf.getPage(i);
            const textItems = await obtenerTextos(page);
            // Aqu inicia la deteccin de materias, horarios y das
        }
        // Se guardan los resultados y se actualiza el estado de la aplicacin
        setMaterias(materiasExtraidas);
    };
    reader.readAsArrayBuffer(pdfFile);
};
```

Código 3.5: Fragmento de Horario.tsx

Gracias a este fragmento de código se logra la carga inicial del *PDF*, primero se lee el archivo con *FileReader*, luego se convierte en un *buffer* de datos que puede entender *pdf.js* y finalmente se obtiene una instancia del documento *PDF* para trabajar con su contenido. El código completo también integra funciones auxiliares que permiten.

- Detectar periodos y fechas.
- Extraer nombres de materias, créditos y aulas.
- Clasificar las materias según semestre y turno.
- Guardar los resultados en memoria local para mantener la persistencia de datos.

Por su extensión, no se muestra completo en esta sección.

Esta función 3.5 recorrer todas las páginas del *PDF*, para extraer la información de materias, horarios y días, almacenando los resultados en memoria local para asegurar la persistencia de los datos del usuario.

3.4.2 Extracción del contenido del *PDF*

Una vez que el archivo *PDF* se logra cargar correctamente en la aplicación, el siguiente paso fue la extracción del contenido, esta parte es probablemente la más importante de toda la implementación, ya que sin poder leer los datos del *PDF* no seria posible construir el horario del estudiante, fue necesario apoyarse en la librería *pdfs-dist*, que permite obtener el texto y sus coordenadas directamente desde cada pagina del documento. El proceso inicia desde la función *processPDF*, que después de abrir el archivo crea un ciclo para recorrer cada pagina. Y analiza sus elementos de texto, en este ciclo se realiza la identificación de las claves de materia, los horarios, los turnos (matutinos o vespertinos), así como otros detalles como créditos y aulas, después los horarios se asignan a los días de la semana tomando como referencia la posición horizontal (Coordenadas X) de cada elemento.

El código 3.6 muestra un fragmento representativo de esta lógica.

```
for (let i = 1; i <= pdf.numPages; i++) {
    const page = await pdf.getPage(i) as PDFPageProxy;
    const textItems = await obtenerTextos(page);
    // Deteccin de claves de materia y horarios
    for (const item of textItems) {
        if (esClaveMateria(item.text)) {
            const clave = item.text;
            const baseY = item.y;
            const horariosCandidatos = textItems
                .filter(item => esHorarioValido(item.text) &&
                 Math.abs(item.y - baseY) <= 100);
            // Asignacin de horarios a los das de la semana
```

```
            const asignacionesDias = horariosCandidatos.map(horario => {
                // Se compara la posicin X del horario con las columnas de los das
            });
            // Registro de la materia en la lista final
            materiasExtraidas.push({ clave, horarios: asignacionesDias });
        }
    }
}
```

Código 3.6: Fragmento de Horario.tsx

El código 3.6 muestra un fragmento representativo de la lógica utilizada para la extracción de información. En dicho código se recorren las páginas del archivo *PDF* con el objetivo de identificar las materias junto con sus respectivos horarios.

El resto del programa contiene funciones auxiliares que permiten:

- Definir periodos y fechas.
- Reconocer los nombres de las materias y detectar posibles recursadoras (R).
- Extraer los creídos y las aulas.
- Clasificar las materias según semestre y turno.

Debido a la extensión del código, este no se presenta de manera completa en la presente sección.

Para poder obtener el texto de cada pagina, se utilizo una función auxiliar llamada *obtenerTextos*, esta función no solo extrae el texto, si no que también guarda sus posiciones (x, y), lo cual fue necesario por que muchos horarios dependen de la alineación en la tabla.

El código 3.7 muestra la implementación de esta función.

```
const obtenerTextos = async (page: PDFPageProxy) => {
    const textContent = await page.getTextContent();
    const viewport = page.getViewport({ scale: 1.0 });
```

```
    const pageHeight = viewport.height;
    return textContent.items
        .filter(
            (item: PDFTextItem): item is PDFTextItem =>
                "str" in item && typeof item.str === "string"
        )
        .map(
            (item: PDFTextItem): TextItem => ({
                text: item.str.trim(),
                x: item.transform[4],
                y: pageHeight - item.transform[5],
                width: item.width || 0,
                height: item.height || 0,
            })
        )
        .filter((item: TextItem) => item.text !== "");
};
```

Código 3.7: Fragmento de Horario.tsx

Obtener Textos.

Con este procedimiento se logra tener una lista de todos los textos en cada página junto con su ubicación, a partir de ahí, se desarrollo la lógica para identificar cuando un texto corresponde a una clave de materia, para esto se programó una función llamada *esClaveMateria*, que detecta si una cadena cumple con el formato de clave, cuando esto ocurre, el sistema ya sabe que ahí empieza el bloque de datos de una materia

El código 3.8 muestra un ejemplo de esta detección.

```
if (esClaveMateria(item.text)) {
    const clave = item.text;
}
```

Código 3.8: Fragmento de Horario.tsx

Clave materia.

En el código 3.8 la función verifica si el texto cumple con el formato de clave, y en

caso afirmativo, almacena la clave para continuar con la extracción de la información de la materia correspondiente.

Una vez localizada la clave, se busco en el texto cercano los datos relacionados: nombre de la materia, horarios, aula, grupo y turno, para eso se usaron funciones auxiliares que comparaban las posiciones y de los textos (es decir, si estaban en la misma linea o muy cerca) y filtraban solo los candidatos validos, por ejemplo para encontrar a los horarios se hizo lo siguiente.

El código 3.9 muestra el procedimiento utilizado para identificar los horarios candidatos asociados a cada materia.

```
1 const clave = item.text
2 const baseY = item.y;
3 const horariosCandidatos = textItems
```

Código 3.9: Fragmento de Horario.tsx

Horarios Candidatos.

Como se observa en el código 3.9, se filtran los textos que cumplen con el formato de horarios y que se encuentran próximos a la clave de la materia. Posteriormente, estos elementos se ordenan según su proximidad vertical, lo que permite identificar los horarios correctos de cada asignatura.

De manera similar, para ubicar el nombre completo de la materia se busco dentro de la misma linea o alrededor, verificando si contenía etiquetas como (TM) o (TV), o si simplemente era un texto largo que no correspondía a una clave.

El código 3.10 muestra el procedimiento utilizado para identificar los nombres candidatos de las materias.

```
1 nombre = claveObj.alias;
2 } else {
3 const nombresCandidatos = encontrarTextoEnLinea(
4 textItems, baseY, 80,
```

Código 3.10: Fragmento de Horario.tsx

Nombres Candidato.

Como se observa en el código 3.10, se buscan textos ubicados en la misma línea o en posiciones cercanas a la clave de la materia que cumplen con ciertos criterios definidos. Este proceso permite identificar de manera confiable los nombres completos de las asignaturas.

Gracias a este procedimiento se pudo reconstruir cada unidad de aprendizaje con toda su información relevante, a pesar de que algunos *PDF* venían con formatos distintos, fue necesario hacer varias pruebas y ajustes, por que ciertos casos los textos no aparecían bien alienados o el nombre de la materia estaba separado en varias partes. Sin embargo, después de depurar la lógica se consiguió una extracción confiable de claves, nombres y horarios y aulas.

Esta etapa fue fundamental por que sin ella no se podría construir el horario del usuario, es el corazón de todo el sistema, leer el *PDF*, detectar las materias y extraer sus datos para que después se puedan mostrar y manipular en la aplicación.

3.4.3 Validación de horarios y resolución de conflictos

Una parte clave de la mejora fue manejar los choques de horario entre clases, en la vida diaria los alumnos a veces quieren anotarse en dos materias que coinciden en el mismo día y hora, y si el sistema no ve esto, el horario sería inadecuado y no seria útil.

Para arreglar esto, se uso un efecto con *React useEffect* que mira todo el tiempo las clases elegidas por el usuario, en este proceso se miran todas las clases con sus días de clase formando eventos con hora para inicio y fin, si hay traslapes, se activa una alerta sobre un choque. El código 3.11 muestra la implementación del efecto encargado de detectar conflictos de horarios dentro de la aplicación. En este proceso, se recorren todas las materias seleccionadas y se generan eventos correspondientes a cada clase, considerando el día y el rango de horas asignado.

```
1 const materiasSeleccionadas = materias.filter(m => m.seleccionada);
2 const eventos: CalendarEvent[] = [];
3 materiasSeleccionadas.forEach((materia, index) => {
4     Object.entries(materia.diasClase).forEach(([dia, horario]) => {
```

```
5         if (horario) {
6             const [horaInicio, horaFin] = horario.split('-')
7             .map(h => parseInt(h, 10));
8             const eventoConflicto = eventos.find(e =>
9                 e.dia === dia &&
10                ((horaInicio >= e.horaInicio && horaInicio < e.horaFin) ||
11                    (horaFin > e.horaInicio && horaFin <= e.horaFin) ||
12                    (horaInicio <= e.horaInicio && horaFin >= e.horaFin))
13            );
14            eventos.push({
15                id: `${materia.id}-${dia}-${horario}`, title: materia.nombre, dia,
16                horaInicio, horaFin, color: coloresMaterias
17                [index % coloresMaterias.length]
18                , materia
19            });
20            if (eventoConflicto) {
21                // Se detecta y registra el conflicto
22            }
23        }
24    });
25});
```

Código 3.11: Fragmento de Horario.tsx

useEffect.

Durante la ejecución de este efecto, los eventos generados se comparan entre sí para identificar posibles traslapes en días y horarios. Cuando se detecta un conflicto, el sistema registra dicha condición y notifica al usuario mediante una alerta, garantizando que el horario final no contenga choques entre materias.

A partir de esta implementación, se realizan las siguientes acciones:

- Almacenar los eventos correspondientes a las materias seleccionadas.
- Detectar conflictos de horario mediante la comparación de rangos de tiempo.
- Actualizar los estados de la aplicación y mostrar alertas al usuario.

Cuando se detecta un conflicto entre dos asignaturas, el sistema muestra una notificación visual en pantalla mediante el componente *IonAlert* de *Ionic*. Este componente permite desplegar cuadros de diálogo personalizables que informan al usuario sobre el conflicto detectado y le ofrecen opciones para decidir cómo proceder.

A través de esta alerta, el usuario puede realizar las siguientes acciones:

- Cancelar la acción y no realizar cambios en el horario.
- Mantener una de las dos asignaturas en conflicto.
- Reiniciar el horario para realizar una nueva selección.

El código 3.12 muestra la implementación del aviso de conflicto de horarios mediante el componente *IonAlert* de *Ionic*, el cual se activa cuando se detecta un traslape entre asignaturas seleccionadas.

```
<IonAlert
    isOpen={showConflictAlert}
    onDidDismiss={() => setShowConflictAlert(false)}
    header="Conflicto de horario!"
    message={conflictMessage}
    buttons={[
        {
        text: conflictMaterias ? `Quedarme con "
        ${conflictMaterias.materia1.nombre}"` : '',
        handler: () => conflictMaterias && handleConflictResolution
        (conflictMaterias.materia1),
        },
        {
        text: conflictMaterias ? `Quedarme con "
        ${conflictMaterias.materia2.nombre}"` : '',
        handler: () => conflictMaterias && handleConflictResolution
        (conflictMaterias.materia2),
        },
    ]}
```

```
/>
```

Código 3.12: Fragmento de Horario.tsx

IonAlert.

En este fragmento se configuran las opciones del cuadro de diálogo, permitiendo al usuario seleccionar cuál de las asignaturas en conflicto desea conservar o, en su caso, cerrar la alerta. Cada botón ejecuta un manejador específico que actualiza el estado de la aplicación, asegurando una resolución controlada del conflicto y una interacción clara con el sistema.

El siguiente fragmento de código es representativo, ya que ejemplifica la definición de la alerta que se muestra cuando se detecta un choque de horarios. Durante la ejecución del sistema, se genera un mensaje que incluye los nombres de las materias involucradas y se proporciona al usuario la posibilidad de tomar una decisión inmediata. Por motivos de extensión, en esta sección no se incluye el código completo.

Finalmente, se implemento un método llamado *handleConflictResolution*, el cual permite al usuario a conservar la asignatura deseada y deseleccionar automáticamente la materia en conflicto. Adicionalmente, se actualiza el almacenamiento local mediante *localStorage*, asegurando que los cambios realizados queden registrados de forma inmediata.

El código 3.13 muestra la implementación de esta función.

```
const handleConflictResolution = (materiaToKeep: Materia) => {
if (conflictMaterias) {
const nuevasMaterias = [...materias];
const materiaToRemove = materiaToKeep.id === conflictMaterias.materia1.id ?
conflictMaterias.materia2 : conflictMaterias.materia1;
const indiceToRemove = nuevasMaterias.findIndex(m => m.id === materiaToRemove.id);
if (indiceToRemove >= 0) {
nuevasMaterias[indiceToRemove].seleccionada = false;
setMaterias(nuevasMaterias);
localStorage.setItem(STORAGE_KEY, JSON.stringify(nuevasMaterias));}
setShowConflictAlert(false);
```

```
12 setConflictMaterias(null);}};
```

Código 3.13: Fragmento de Horario.tsx
handleConflictResolution.

Como se observa en el código 3.13, la función identifica la asignatura que debe ser removida a partir de la selección del usuario, actualiza el estado de las materias en *React* y persiste los cambios en *localStorage*. Este procedimiento garantiza que el horario permanezca actualizado y libre de conflictos después de la resolución del choque detectado.

Con esto se hizo que el sistema no solo diga si hay problema sino que ademas ofrezca solución rápida al usuario para que pueda arreglar al situación rápido.

3.4.4 Almacenamiento y persistencia de datos

Uno de los principales retos del sistema fue garantizar que los horarios y las selecciones realizadas por el usuario se conservaran al cerrar la aplicación o recargar la página. Para esto, se implemento el uso de *localStorage*, lo que permite almacenar información de forma local en el navegador y mantener la persistencia de los datos entre sesiones.

El sistema almacena información en distintos momentos del uso de la aplicación, los cuales se describen a continuación:

1. Guardar después de procesar un *PDF*: Cuando se sube y procesa un nuevo documento, las materias extraídas junto con el momento y la fecha se almacenan en *localStorage*. El código 3.14 actualiza los estados de *React* y guarda las materias extraídas junto con la información de semestres, periodo y fecha en *localStorage*, asegurando que el horario del usuario permanezca disponible incluso después de cerrar o recargar la aplicación.

```
1 setMaterias(materiasExtraidas);
2 setSemestres(["Todos", ...Array.from(semestresList).filter
3     (sem => sem !== "Todos")]);
4 setPeriodoActual(periodoDetectado);
```

```
setFechaActual(fechaDetectada);
setIsLoading(false);
localStorage.setItem(STORAGE_KEY, JSON.stringify(materiasExtraidas));
localStorage.setItem(PERIODO_KEY, periodoDetectado);
localStorage.setItem(FECHA_KEY, fechaDetectada);
```

Código 3.14: Fragmento de Horario.tsx
Materias Extraídas.

Este fragmento garantiza que la información persista en el navegador y que el usuario pueda continuar su actividad sin perder los datos previamente seleccionados.

2. Guardado al seleccionar o deselecionar materias: Cada vez que el usuario marca o desactiva algo, la lista actualizada se guarda automáticamente. El código 3.15 implementa la función *toggleMateriaSeleccion*, que actualiza el estado de selección de una materia y persiste los cambios en *localStorage* cada vez que el usuario marca o desmarca una materia.

```
const toggleMateriaSeleccion = (index: number) => {
const nuevasMaterias = [...materias];
nuevasMaterias[index].seleccionada = !nuevasMaterias[index].seleccionada;
setMaterias(nuevasMaterias);
localStorage.setItem(STORAGE_KEY, JSON.stringify(nuevasMaterias));};
```

Código 3.15: Fragmento de Horario.tsx
toggleMateriaSeleccion.

Este fragmento garantiza que las elecciones del usuario se mantengan actualizadas y disponibles incluso después de cerrar o recargar la aplicación.

3. Guardado manual mediante botón: El sistema incluye un botón que permite a los usuarios guardar manualmente su horario. Además, se genera un respaldo descargable en formato *.json*, el cual está disponible únicamente cuando la

aplicación se utiliza en equipos de escritorio o portátiles, ya que la descarga de archivos no se encuentra habilitada en dispositivos móviles.

El código 3.16 implementa la función *guardarHorario*, que permite al usuario guardar manualmente su horario. La función filtra las materias seleccionadas, actualiza *localStorage* y genera un archivo descargable en formato *.json* con toda la información relevante.

```
const guardarHorario = () => {
    const materiasSeleccionadas = materias.filter(m => m.seleccionada);
    if (materiasSeleccionadas.length === 0) {
        alert("No has seleccionado ninguna materia");
        return;
    }
    localStorage.setItem(STORAGE_KEY, JSON.stringify(materias));
    localStorage.setItem(PERIODO_KEY, periodoActual);
    localStorage.setItem(FECHA_KEY, fechaActual);
    alert("Horario guardado correctamente!");
    const blob = new Blob(
        [JSON.stringify({
            periodo: periodoActual,
            fecha: fechaActual,
            materias: materiasSeleccionadas,
            creditosTotales: calcularCreditosTotales()
        }, null, 2)],
        { type: 'application/json' }
    );
    const url = URL.createObjectURL(blob);
    const link = document.createElement('a');
    link.href = url;
    link.download = `horario_${new Date()
        .toISOString().split('T')[0]}.json`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
```

```
28 };
```

Código 3.16: Fragmento de Horario.tsx
Guardar Horario.

Este fragmento asegura que el usuario pueda conservar su horario de manera externa y segura, permitiendo descargarlo como archivo *.json* mientras mantiene actualizados los datos en *localStorage*.

4. Guardado al resolver conflictos: Cuando el usuario selecciona qué asignatura conservar durante un conflicto de horario, el sistema registra automáticamente dicha elección. El código 3.17 muestra cómo, mediante la función *handleConflictResolution*, se actualiza el estado de las materias y se persisten los cambios en *localStorage*, garantizando que la decisión del usuario se mantenga incluso después de cerrar o recargar la aplicación.

```
1 setMaterias(nuevasMaterias);
2 localStorage.setItem(STORAGE_KEY, JSON.stringify(nuevasMaterias));
3 setShowResetAlert(false);
```

Código 3.17: Fragmento de Horario.tsx
handleConflictResolution.

Recuperación de datos

Cuando la aplicación se inicia, se verifica la existencia de información previamente almacenada en *localStorage*. En caso de que estos datos estén disponibles, el sistema recupera automáticamente el periodo, la fecha y las materias guardadas, permitiendo restaurar el horario del usuario sin necesidad de una nueva configuración.

El código 3.18 muestra cómo se implementa este proceso de verificación y carga inicial de datos mediante el uso del gancho *useEffect*.

```
1 useEffect(() => {
2     const savedPeriodo = localStorage.getItem(PERIODO_KEY);
3     const savedFecha = localStorage.getItem(FECHA_KEY);
```

```
4       const savedMaterias = localStorage.getItem(STORAGE_KEY);
5
6       if (savedPeriodo) {
7           setPeriodoActual(savedPeriodo);
8       }
9       if (savedFecha) {
10          setFechaActual(savedFecha);
11      }
12      if (savedMaterias) {
13          try {
14              const parsedMaterias = JSON.parse(savedMaterias);
15              setMaterias(parsedMaterias);
16              setDatosGuardados(true);
17
18              const semestresList = new Set(["Todos", "Recurse"]);
19              parsedMaterias.forEach((materia: Materia) => {
20                  if (materia.semestre) {
21                      semestresList.add(materia.semestre);
22                  }
23              });
24              setSemestres([
25                  "Todos",
26                  ...Array.from(semestresList).filter
27                  (sem => sem !== "Todos")
28              ]);
29          } catch (error) {
30              console.error("Error loading saved data:", error);
31          }
32      }
33  }, []);
```

Código 3.18: Fragmento de Horario.tsx

useEffect inicial.

Tal como se muestra en el código 3.18, el sistema recupera la información almacenada

en *localStorage* y actualiza los estados correspondientes en *React*, permitiendo que el horario del usuario se restaure automáticamente al iniciar la aplicación.

También agregaron una forma de mantener las mismas selección de materias, si se sube un nuevo *PDF* con los mismos cursos. El código 3.19 muestra cómo se verifica y actualiza el estado de selección de las materias utilizando la información almacenada anteriormente.

```
if (datosGuardados) {
    const materiasGuardadas = [...materias];
    materiasExtraidas.forEach(nuevaMateria => {
        const materiaGuardada = materiasGuardadas.find
        (m => m.id === nuevaMateria.id);
        if (materiaGuardada) {
            nuevaMateria.seleccionada =
             materiaGuardada.seleccionada;
        }
    });
```

Código 3.19: Fragmento de Horario.tsx

Datos Guardados.

Tal como se muestra en el código 3.19, cuando existen datos previamente almacenados, el sistema compara las materias extraídas del nuevo archivo con las guardadas anteriormente y conserva el estado de selección del usuario, garantizando una experiencia continua incluso al actualizar el archivo PDF.

Eliminación de datos

Para gestionar de forma adecuada la información almacenada en *localStorage*, el sistema implementa dos mecanismos que permiten conservar o descartar las selecciones de materias previamente realizadas por el usuario:

- *Conservación o eliminación de selecciones previas*: El código 3.20 muestra cómo el sistema utiliza la información previamente almacenada para decidir si las selecciones de materias deben mantenerse cuando se procesa un nuevo archivo PDF.

```
if (datosGuardados) {
    const materiasGuardadas = [...materias];
    materiasExtraidas.forEach(nuevaMateria => {
        const materiaGuardada = materiasGuardadas.find
        (m => m.id === nuevaMateria.id);
        if (materiaGuardada) {
            nuevaMateria.seleccionada = materiaGuardada.seleccionada;
        }
    });
}
```

Código 3.20: Fragmento de Horario.tsx

Borrar Horario.

Tal y como se refleja en el código 3.20, al detectar que existen datos previamente guardados (*datosGuardados*), el sistema compara cada nueva materia con las guardadas anteriormente y actualiza el estado seleccionada, permitiendo al usuario decidir si mantener o eliminar las selecciones previas.

- *Reinicio completo de la aplicación:* Este mecanismo permite eliminar toda la información previamente almacenada y restablecer el sistema a su estado inicial.

  El código 3.21 muestra cómo se realiza este reinicio completo de la aplicación.

```
const reiniciarAplicacion = () => {
    localStorage.removeItem(STORAGE_KEY);
    localStorage.removeItem(PERIODO_KEY);
    localStorage.removeItem(FECHA_KEY);
    setMaterias([]);
    setPeriodoActual("");
    setFechaActual("");
    setFile(null);
    setSemestreSeleccionado("Todos");
    setTurnoSeleccionado("Todos");
    setSemestres(["Todos", "Recurse"]);
    setDatosGuardados(false);
```

```
13     setShowResetAlert(false);
14 };
```

Código 3.21: Fragmento de Horario.tsx

Reiniciar Aplicación.

Como se detalla en el código 3.21, el método *reiniciarAplicacion* elimina todas las claves almacenadas en *localStorage* y reinicia los estados de *React*, incluyendo las materias, el periodo, la fecha, el archivo cargado, así como el semestre y turno seleccionados. Esto asegura que la aplicación vuelva a un estado limpio, lista para una nueva configuración.

Claves utilizadas en el almacenamiento

Para la persistencia de la información del usuario, se definieron tres claves específicas dentro de *localStorage*, las cuales permiten almacenar de manera ordenada los datos asociados al horario escolar.

El código 3.22 muestra las claves empleadas para guardar las materias seleccionadas, el periodo académico y la fecha correspondiente.

```
1 const STORAGE_KEY = 'horario_ICO_materias';
2 const PERIODO_KEY = 'horario_ICO_periodo';
3 const FECHA_KEY = 'horario_ICO_fecha';
```

Código 3.22: Fragmento de Horario.tsx

Claves utilizadas.

Como se observa en el código 3.22, el uso de claves diferenciadas permite almacenar y recuperar la información del estudiante de forma confiable, garantizando que el horario pueda restaurarse correctamente sin necesidad de reprocesar el archivo *PDF* cada vez que se reinicia la aplicación.

Gracias a esto, el sistema permite que el estudiante mantenga su progreso y puede retomar donde lo dejo en cualquier momento, sin necesidad de rehacer un *PDF* desde el primer momento.

3.4.5 Interfaz de usuario *(Frontend* con *Ionic/React)*

La primera funcionalidad incorporada a la interfaz del sistema fue permitir al usuario cargar su horario mediante un archivo en formato *PDF* desde la aplicación. Este paso es fundamental, ya que el sistema requiere dicho archivo como insumo inicial para poder comenzar la generación y configuración del horario académico, ya que sin un archivo cargado, no es posible continuar con el proceso.

Para implementar esta funcionalidad, se utilizó el *Framework Ionic* junto con *React*, haciendo uso de componentes visuales como *IonCard*, *IonCardHeader* o *IonCardTitle*, los cuales permiten crear un espacio visual claramente delimitado donde el usuario puede seleccionar su archivo. Este componente muestra un mensaje de orientación y un campo de entrada (*input*) configurado para aceptar únicamente archivos con extensión *.pdf*. Una vez que el usuario selecciona un archivo válido, se ejecuta la función *handleFileChange*, la cual almacena temporalmente el archivo y da inicio al proceso de lectura y análisis del documento. El código 3.23 muestra la implementación de esta función.

```
const handleFileChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    const selectedFile = event.target.files?.[0];
    if (selectedFile) {
        setFile(selectedFile);
        setIsLoading(true);
        processPDF(selectedFile);
    }
};
```

Código 3.23: Fragmento de Horario.tsx

handleFileChange.

Como se observa en el código 3.23, la función *handleFileChange* obtiene el archivo seleccionado desde el evento de cambio, lo guarda temporalmente mediante *setFile*, activa un indicador de carga con *setIsLoading(true)* y posteriormente invoca la función *processPDF*, encargada de iniciar el procesamiento del contenido del archivo PDF.

Durante el proceso de carga y lectura del archivo PDF, se implementó un indicador visual de progreso circular, representado por una rueda animada, con el propósito de informar al usuario que el archivo se encuentra en proceso de análisis. Este indicador evita que el usuario perciba la aplicación como inactiva mientras se ejecutan las operaciones internas.

El código 3.24 muestra la implementación de este indicador de carga.

```
{isLoading && (
    <div className="loading-container">
        <p>Procesando archivo...</p>
        <div className="spinner"></div>
    </div>
)}
```

Código 3.24: Fragmento de Horario.tsx

isLoading.

Tal como se observa en el código 3.24, cuando la variable de estado *isLoading* se encuentra en *true*, se renderiza un contenedor que incluye un mensaje informativo con el texto *"Procesando archivo..."* junto con un componente visual tipo *spinner*. Dicho elemento permanece visible mientras el sistema realiza la lectura y procesamiento del archivo *PDF*.

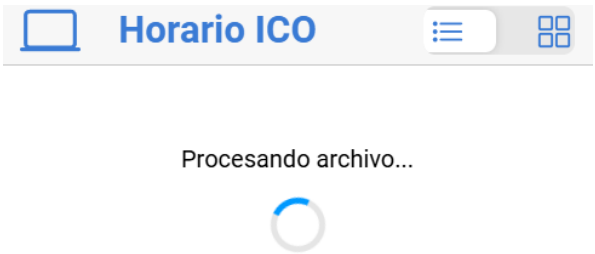

Figura 3.1: Indicador visual de carga durante el procesamiento del archivo PDF.

La Figura 3.1 muestra el resultado visual del código 3.24. Cuando la variable *isLoading* es verdadera, la aplicación despliega un mensaje de procesamiento acompañado de un indicador gráfico de carga, informando al usuario que el archivo *PDF* está siendo analizado.

Este mecanismo mejora la experiencia de usuario al proporcionar retroalimentación visual inmediata sobre el estado del sistema, Además de permitir que la aplicación conserve correctamente las selecciones previas del usuario en caso de que ya se haya cargado un archivo anteriormente.

El archivo *PDF* se abre en la memoria y va a la función *processPDF* (hablada en el apartado 3.4, que busca la información importante de cada pagina, como nombres de temas, horas de clases y lugares de la aula. En esta sección se describe la interacción del usuario con la interfaz gráfica y como los estados de *React(isLoading, file)* dejan

que la aplicación responda rápido a sus acciones.

También, la forma en que se ve y se usa tiene las elecciones anteriores del usuario guardadas, si ya había ubicado un *PDF* antes, las clases escogidas previamente se señalan solo al poner un nuevo papel, lo que hace mejor la experiencia y evita que falle algo.

Este enfoque facilita al estudiante la carga de su horario e integra elementos visuales que mejoran la integración con archivos *PDFs*, ofreciendo una experiencia de uso más intuitiva dentro de la aplicación.

3.4.6 Visualización de horarios y selección de materias

Una vez que el sistema logra extraer las materias desde el archivo *PDF*, se desarrolló un módulo encargado de visualizar el horario en forma de un calendario interactivo. Este módulo permite al estudiante seleccionar o deseleccionar las materias de su interés, facilitando la construcción y personalización de su horario académico.

Para lograrlo, se desarrollaron diversas funciones encargadas de organizar la información, generar las tablas correspondientes y permitir la interacción del usuario con el calendario.

Renderizacion de horarios

Una vez que el sistema extrae las materias del archivo *PDF*, la siguiente etapa del desarrollo consistió en implementar un módulo que permite visualizar el horario en forma de un calendario interactivo, así como habilitar la selección y deselección de materias por parte del estudiante.

Para ello, se desarrollaron diversas funciones encargadas de organizar la información, generar las tablas correspondientes y permitir la interacción del usuario con el calendario. El calendario principal se construye automáticamente a partir de la lista de eventos generados a partir de las materias. Cuando una clase coincide con un intervalo de tiempo específico, se representa visualmente en dicho espacio, utilizando un color distintivo y un tamaño proporcional a la duración de la sesión.

Interacción con el calendario

Esto permite ver de manera claro lo que sucede en cada franja horaria, al hacer clic en un tema, aparece una ventana emergente con todos los datos importantes como el nombre de la asignatura, el aula y las horas. Además de esta vista detallada, se preparo un calendario general que mezclan algunas materias seleccionadas y disponibles, lo que permite al estudiante elegir directamente en el calendario interactivo cuales desea cursar.

El código 3.25 muestra la función *renderCalendarioCompleto* que genera este calendario dinámico.

```
const renderCalendarioCompleto = () => {
    return (
        <div className="calendario-container">
            <div className="calendario-header">
            {/* Cabecera con das de la semana */}
            </div>
            <div className="calendario-body">
            {horasCalendario.map(hora => (
            <div key={hora} className="calendario-fila">
            {/* Celdas del calendario */}
            {diasSemana.map(dia => (
            <div key={dia} className="calendario-celda">
            {allCalendarEvents
            .filter(ev => ev.dia === dia && ev.horaInicio <=
                hora && ev.horaFin > hora)
            .map(ev => (
            <div
                key={ev.id}
                className={`evento-calendario
                    ${ev.materia.seleccionada ?
                'evento-seleccionado' : 'evento-disponible'}`}
                style={{ backgroundColor: ev.color }}
                onClick={() => toggleMateriaSeleccion(index)}>
                <div className=
                "evento-titulo">{ev.title}</div>
```

```
26                  <div className=
27                  "evento-subtitulo">{ev.materia.aula}</div>
28                  </div>
29              ))}</div>))}</div>))}</div></div>
30      );
31  };
```

Código 3.25: Fragmento de Horario.tsx
renderCalendarioCompleto.

Como se aprecia en el código 3.25, la función *renderCalendarioCompleto* recorre todas las horas del calendario y los días de la semana, generando celdas que representan cada franja horaria. Las materias seleccionadas se muestran con un estilo diferenciado respecto a las disponibles, y al hacer clic sobre ellas se activa la función *toggleMateriasSeleccion*, permitiendo al usuario seleccionar o deseleccionar asignaturas directamente desde el calendario.

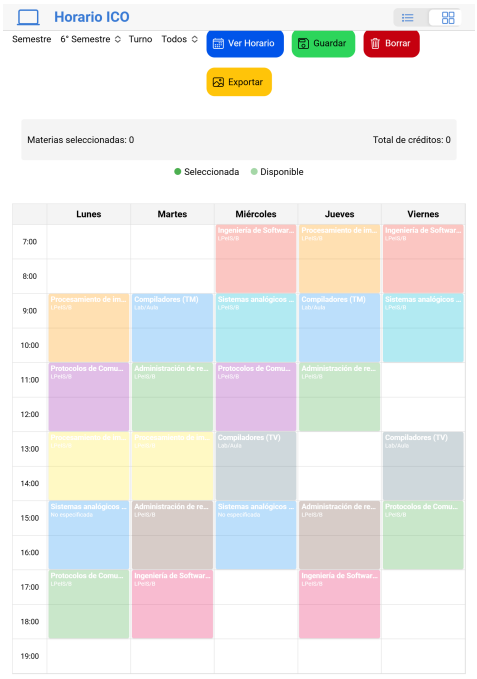

Figura 3.2: Calendario dinámico generado por la función *renderCalendarioCompleto*.

La Figura 3.2 muestra el resultado visual de la función *renderCalendarioCompleto*,

donde se presenta el calendario académico organizado por días de la semana y franjas horarias. En esta vista, las materias seleccionadas se distinguen visualmente de las disponibles, permitiendo al usuario interactuar directamente con el calendario para seleccionar o deseleccionar asignaturas.

Selección y deseccion de materias

El sistema tiene este cambio de interruptor (*toggle*) donde el estudiante puede marcar si ha elegido una materia o no, la función *toggleMaterialSeleccion*, la cual mantiene sincronizado el estado local de la aplicación con el almacenamiento del navegador. El código 3.26 muestra la implementación de esta función.

```
const toggleMateriaSeleccion = (index: number) => {
    const nuevasMaterias = [...materias];
    nuevasMaterias[index].seleccionada =
    !nuevasMaterias[index].seleccionada;
    setMaterias(nuevasMaterias);
    localStorage.setItem(STORAGE_KEY,
        JSON.stringify(nuevasMaterias));
};
```

Código 3.26: Fragmento de Horario.tsx
toggleMateriaSeleccion

Como se aprecia en el código 3.26, cada vez que el usuario marca o desmarca una materia, el estado de la aplicación se actualiza mediante `setMaterias`, y los cambios se guardan automáticamente en *localStorage*. Esto asegura que la selección del estudiante se mantenga aunque recargue la página o cierre el navegador.

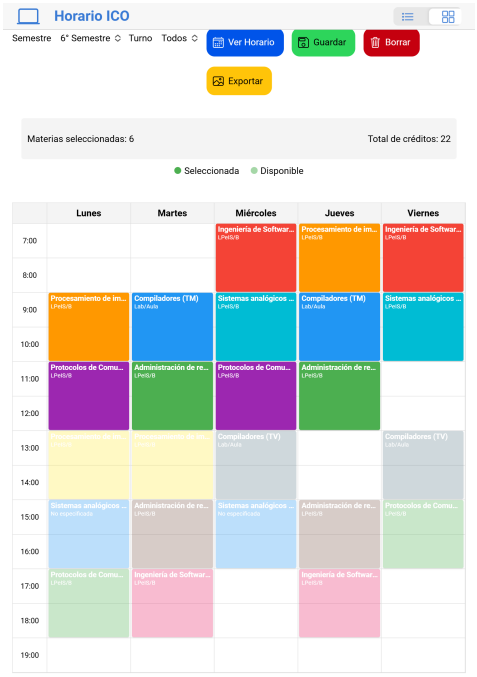

Figura 3.3: Resultado visual de la función *toggleMateriaSeleccion* en el calendario académico.

En la Figura 3.11 se observa el calendario académico donde las materias seleccionadas se distinguen visualmente de aquellas que no lo están. Esta retroalimentación visual permite al estudiante identificar fácilmente su horario actual y realizar ajustes de manera intuitiva.

Para la visualización de cada materia dentro de la lista, se utiliza el componente *IonItem*, el cual permite integrar un control de tipo *toggle* que facilita la selección y deselección de asignaturas. Este componente muestra la información principal de cada materia, incluyendo nombre, clave, semestre, turno, créditos, aula y horarios.

El código 3.27 presenta la implementación de esta visualización, donde cada elemento de la lista se genera de manera dinámica a partir de los datos de la materia correspondiente. El estado del *toggle* se encuentra sincronizado con el atributo *seleccionada*, lo que permite reflejar de forma inmediata la interacción del usuario.

```
1 const renderMateriaItem = (materia: Materia, index: number) => {
2     return (
3         <IonItem
```

```
            key={materia.id}
            className={materia.seleccionada ?
            'materia-seleccionada' : ''}>
            <IonToggle
                checked={materia.seleccionada}
                onIonChange={() => toggleMateriaSeleccion(index)}
                slot="start" />
            <IonLabel>
                <h2>{materia.nombre}</h2>
                <p>Clave: {materia.clave} |
                    {materia.semestre} | {materia.turno}</p>
                <p>Crditos: {materia.creditos}
                    | Aula: {materia.aula}</p>
                <p>
                    {Object.entries(materia.diasClase)
                        .filter(([, horario]) => horario)
                        .map(
                            ([dia, horario]) =>
                                `${dia.charAt(0).toUpperCase()
                                    + dia.slice(1)}: ${horario}`
                        )
                        .join(" | ")
                    }
                </p></IonLabel></IonItem>
    );
};
```

Código 3.27: Fragmento de Horario.tsx

renderMateriaItem.

Asimismo, la selección realizada se mantiene actualizada tanto en el estado de la aplicación en *React* como en *localStorage*, garantizando la persistencia de la información incluso si el usuario recarga la página o cierra la aplicación.

Identificación visual de materias

El usuario tiene que distinguir visualmente el estado de cada materia, así que se

desarrollaron unos estilos que marcan la diferencia entre las materias que ya están elegidas, las materias que están disponibles para agregar y las que generan conflicto de horario, por ejemplo, las seleccionadas se ven en un color solido, las disponibles son más neutras y las que chocan aparecen en rojo para alertar inmediatamente.

El fragmento de código 3.28 muestra la implementación de estos estilos.

```
.materia-seleccionada {
    --background: rgba(56, 128, 255, 0.1);
}
.evento-conflicto {
    border: 2px solid red;
    box-shadow: 0 0 5px rgba(255, 0, 0, 0.5);
}
.evento-seleccionado {
    opacity: 1;
}
.evento-disponible {
    opacity: 0.6;
}
```

Código 3.28: Fragmento de Horario.tsx estilosSeleccionarDisponiblesConflicto.

También se agrego una pequeña leyenda al principio del listado, con colores y ejemplos, para que cualquiera entienda que significa cada estilo sin tener que adivinarlo, así evitamos confusiones y la interfaz resulta más amigable.

El fragmento de código 3.29 muestra cómo se implementó esta leyenda:

- **Seleccionada:** materia que el usuario ha marcado.
- **Disponible:** materia que aún no ha sido seleccionada.

```
<div className="leyenda">
    <span className="leyenda-item seleccionada">Seleccionada</span>
    <span className="leyenda-item disponible">Disponible</span>
```

```
</div>
<div className="leyenda">
    <span className="leyenda-item seleccionada">Seleccionada</span>
    <span className="leyenda-item conflicto">Conflicto</span>
</div>
```

Código 3.29: Fragmento de Horario.tsx
leyendaVisual.

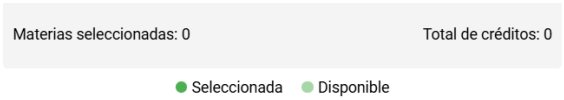

Figura 3.4: Leyenda visual de estados de las materias y resumen del horario académico.

En la Figura 3.4 se muestra la leyenda visual utilizada en el sistema, junto con el total de materias seleccionadas y la cantidad de créditos acumulados. Esta información permite al estudiante tener una visión clara y rápida de su avance en la construcción del horario académico.

Control de turnos

Respecto al control de turnos, se incluyo un filtro que deja al usuario elegir si quiere ver solo materias del turno matutino, solo el vespertino o ambos, se uso *IonSelect* que va actualizando la lista al momento según lo que elija la persona. Este filtro actualiza la lista de manera inmediata según la elección del usuario, como se muestra en el código 3.30.

```
<IonSelect
    label="Turno"
    value={turnoSeleccionado}
    onIonChange={e => setTurnoSeleccionado(e.detail.value)}
>
    {turnos.map(turno => (
        <IonSelectOption key={turno} value={turno}>
            {turno}
```

```
        </IonSelectOption>
    ))}
</IonSelect>
```

Código 3.30: Fragmento de Horario.tsx

IonSelect.

Para determinar automáticamente el turno de cada clase, se implemento una función que mira la hora de inicio de la materia.

```
const horarioATurno = (horario: string): string => {
    if (!horario) return "No definido";
    const hora = parseInt(horario.split('-')[0], 10);
    return hora < 12 ? "Matutino" : "Vespertino";
};
```

Código 3.31: Fragmento de Horario.tsx

funcioHorarioATurno.

Como aparece en el código 3.31, la función horarioATurno determina automáticamente el turno de cada clase según la hora de inicio, asignando "Matutino" si inicia antes de las 12 y "Vespertino" si inicia después, evitando depender de datos preclasificados.

Y por si acaso, si alguien intenta meter materias de distintos turnos que se traslapan, el sistema también lo detecta, salta una alerta que obliga al usuario a decidir cual de las dos quiere conservar. De esta forma, se asegura que no se tenga conflictos con los horarios y que todo quede claro y bien organizado.

Al integrar todos estos elementos, el calendario que se genera automáticamente, la lista que ofrece referencias visuales sobre el estado de las materias, la leyenda explicativa y el filtro para seleccionar el turno, se obtiene una herramienta clara y fácil de usar. De esta manera, los estudiantes pueden planear su semestre con mayor confianza, evitando el riesgo de traslapar materias sin percatarse de ello.

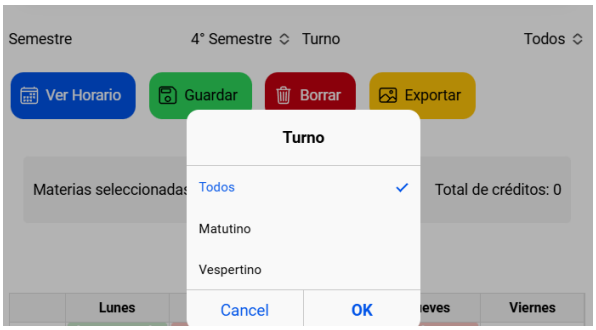

Figura 3.5: Filtro de selección de turnos académicos en la interfaz del sistema.

En la Figura 3.5 se muestra el filtro de turnos disponible en la aplicación, donde el usuario puede elegir entre las opciones matutino y vespertino para ajustar la visualización de las materias de acuerdo con sus preferencias.

3.4.7 Gestión de conflictos de horario y notificaciones emergentes

Uno de los problemas más comunes a la hora de realizar un horario es que, sin querer los estudiantes eligen dos materias que chocan a la misma hora, es frustrante por que después se dan cuenta que no pueden cursarlas al mismo tiempo y que se tiene que volver a empezar.

Para evitar eso, se desarrollo una lógica al sistema que detecta automáticamente esos choques de horario y le avisa al usuario en el momento, antes de que guarde todo definitivamente.

La validación se ejecuta dentro de un efecto *useEffect*, el cual se mantiene atento a los cambios en las materias seleccionadas. En este proceso, se analizan los días y horarios de cada asignatura y se comparan con los eventos ya registrados en el calendario, con el objetivo de detectar posibles traslapes. Cuando se identifica un conflicto de horario, el sistema despliega una alerta clara e identifica las materias involucradas en el traslape, permitiendo al usuario reconocer de forma precisa cuáles asignaturas deben revisarse.

El siguiente fragmento de código muestra la parte central de este proceso:

```
useEffect(() => {
    const materiasSeleccionadas = materias.filter(m => m.seleccionada);
    const eventos: CalendarEvent[] = [];
    let hayConflicto = false;
    let mensajeConflicto = "";
    let materiasEnConflicto: { materia1: Materia, materia2: Materia } | null = null;
    materiasSeleccionadas.forEach((materia) => {
        Object.entries(materia.diasClase).forEach(([dia, horario]) => {
            if (horario) {
                const [horaInicio, horaFin] = horario.split('-')
                .map(h => parseInt(h, 10));
                const eventoConflicto = eventos.find(e =>
                    e.dia === dia &&
                    ((horaInicio >= e.horaInicio && horaInicio < e.horaFin) ||
                        (horaFin > e.horaInicio && horaFin <= e.horaFin))
                );
                eventos.push({
                    id: `${materia.id}-${dia}-${horario}`,
                    title: materia.nombre, dia, horaInicio, horaFin, materia
                });
                if (eventoConflicto) {
                    hayConflicto = true;
                    mensajeConflicto = `Hay un conflicto entre "${materia.nombre}
          " y "${eventoConflicto.materia.nombre}"`;
                    materiasEnConflicto = { materia1: materia, materia2:
                         eventoConflicto.materia };
                }
            }
        });
    });
    if (hayConflicto && materiasEnConflicto) {
        setConflictMessage(mensajeConflicto);
        setConflictMaterias(materiasEnConflicto);
        setShowConflictAlert(true);
```

```
    }
}, [materias, showConflictAlert]);
```

Código 3.32: Fragmento de Horario.tsx

materiasSeleccionadas.

En este código 3.32, useEffect se encarga de validar continuamente las materias seleccionadas, revisando sus días y horarios para detectar conflictos. Cuando encuentra un empalme, genera un mensaje claro y registra las materias en conflicto, permitiendo al usuario identificar rápidamente qué clases requieren su atención.

Cuando se identifica un conflicto de horario entre dos materias, el sistema despliega un cuadro de diálogo interactivo que guía al usuario en la resolución del problema. A través de este mecanismo, el estudiante puede seleccionar la asignatura que desea conservar, asegurando una experiencia de uso clara, controlada y sin interrupciones innecesarias.

```
<IonAlert
    isOpen={showConflictAlert}
    onDidDismiss={() => setShowConflictAlert(false)}
    header="Conflicto de horario!"
    message={conflictMessage}
    buttons={[{
        text: Cancelar,
        role: cancel,
        handler: () => {
            setShowConflictAlert(false);
            setConflictMaterias(null);
        }}, {
        text: conflictMaterias ? Quedarme con
        "${conflictMaterias.materia1.nombre}" : ,
        handler: () => {
            if (conflictMaterias) {
                handleConflictResolution(conflictMaterias.materia1);
            }
        }}, {
```

```
        text: conflictMaterias ? Quedarme con
        "${conflictMaterias.materia2.nombre}" : ,
        handler: () => {
            if (conflictMaterias) {
                handleConflictResolution(conflictMaterias.materia2);
            }
        }]} cssClass="conflict-alert"
/>
```

Código 3.33: Fragmento de Horario.tsx

IonAlertConflict.

En el código 3.33, cuando el sistema detecta un conflicto entre materias, se muestra un *IonAlert* que permite al usuario decidir cuál materia conservar o cancelar la acción, ofreciendo así una solución inmediata y manteniendo el flujo de trabajo sin interrumpir la experiencia de uso.

Internamente, el método *handleConflictResolution* se encarga de gestionar la resolución del conflicto, desmarcando automáticamente la materia que el usuario decidió no conservar y actualizando el estado del sistema de forma inmediata, de modo que el conflicto desaparezca al instante.

El fragmento de código 3.34, muestra cómo, al detectar un conflicto de horario, las materias involucradas se resaltan visualmente dentro del calendario mediante un borde rojo punteado y una animación suave. Este recurso visual facilita la identificación inmediata de los cursos en conflicto, permitiendo al usuario localizar el problema de manera clara y precisa.

```
.evento-conflicto {
    border: 2px solid red;
    box-shadow: 0 0 5px rgba(255, 0, 0, 0.5);
}
```

Código 3.34: Fragmento de Horario.tsx

En conjunto, este mecanismo ofrece una experiencia más clara y práctica, los estudiantes reciben una advertencia visual y textual, pueden resolver el conflicto en

el momento y el sistema asegura que al guardar el horario no existan duplicidades ni choques que afecten su organizaron.

Validación al guardar el horario

Para evitar que se almacenen horarios incompletos o con materias que presenten conflictos entre sí, se implementó un mecanismo de validación previo al proceso de guardado. Dicho mecanismo revisa la información antes de almacenarla en *localStorage* y únicamente permite guardar horarios que no presentan traslapes ni inconsistencias.

De esta manera, se garantiza que, al momento de guardar el horario, los datos sean coherentes y estén correctamente estructurados, evitando conflictos inadvertidos en el calendario y asegurando un funcionamiento estable de la aplicación.

Primero, se revisa que el estudiante haya seleccionado al menos una materia, si no es así, se muestra un mensaje de alerta indicado que no ha seleccionado nada, después se comprueba si aun existen conflictos sin resolver, solo cuando ambas condiciones se cumplen, el sistema permite guardar el horario.

```
const guardarHorario = () => {
    const materiasSeleccionadas = materias.filter(m => m.seleccionada);
    if (materiasSeleccionadas.length === 0) {
        alert("No has seleccionado ninguna materia");
        return;
    }
    if (eventosConflicto.length > 0) {
        alert("Tienes conflictos de horario sin resolver. " +
              " Por favor, resulvelos antes de guardar."
        );return;}
    localStorage.setItem(STORAGE_KEY, JSON.stringify(materias));
    localStorage.setItem(PERIODO_KEY, periodoActual);
    localStorage.setItem(FECHA_KEY, fechaActual);
    alert("Horario guardado correctamente!");
    const blob = new Blob(
        [
            JSON.stringify(
                {   periodo: periodoActual,
```

```
                    fecha: fechaActual,
                    materias: materiasSeleccionadas,
                    creditosTotales: calcularCreditosTotales()
                },null, 2)],{ type: 'application/json' });
    const url = URL.createObjectURL(blob);
    const link = document.createElement('a');
    link.href = url;
    link.download = `horario_${new Date().toISOString().split('T')[0]}.json`;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
};
```

Código 3.35: Fragmento de Horario.tsx
guardarHorarioConflicto.

En el código 3.35, la función *guardarHorario* valida que el usuario haya seleccionado al menos una materia y que no existan conflictos pendientes antes de guardar el horario, garantizando coherencia en los datos y evitando inconsistencias en el calendario.

De esta manera, el usuario recibe retroalimentación inmediata si guarda un horario que no cumple con las reglas y el sistema mantiene siempre la integridad de los datos, así mismo al guardar se preserva la selección de materias y el periodo correspondiente, lo que permite continuar trabajando en sesiones posteriores sin perder la información.

3.4.8 Persistencia de datos y manejo de semestres

Con el fin de preservar el progreso del usuario, el horario se almacena automáticamente en el navegador mediante el uso de *localStorage*. De este modo, la información seleccionada, como las materias elegidas, el periodo académico y la fecha de generación del horario, permanece disponible incluso después de cerrar la aplicación o apagar el equipo.

Cada vez que la aplicación se inicia, el sistema verifica la existencia de información previamente almacenada. En caso de encontrar un horario guardado, se recuperan los

datos correspondientes y se reconstruyen la lista de materias y los semestres asociados, permitiendo continuar con el uso de la aplicación sin necesidad de repetir el proceso desde el inicio.

```
useEffect(() => {
    const savedPeriodo = localStorage.getItem(PERIODO KEY);
    const savedFecha = localStorage.getItem(FECHA KEY);
    const savedMaterias = localStorage.getItem(STORAGE KEY);
    if (savedPeriodo) {
        setPeriodoActual(savedPeriodo);
    }
    if (savedFecha) {
        setFechaActual(savedFecha);
    }
    if (savedMaterias) {
        try {
            const parsedMaterias = JSON.parse(savedMaterias);
            setMaterias(parsedMaterias);
            setDatosGuardados(true);
            const semestresList = new Set(["Todos", "Recurse"]);
            parsedMaterias.forEach((materia: Materia) => {
                if (materia.semestre) {
                    semestresList.add(materia.semestre);
                }
            });
            setSemestres(["Todos", ...Array.from(semestresList)
                .filter(sem => sem !== "Todos")]);
        } catch (error) {
            console.error("Error loading saved data:", error);
        }
    }
}, []);
```

Código 3.36: Fragmento de Horario.tsx

savedHorario.

Como señala el código 3.36, al iniciar la aplicación, se verifica si existen datos previamente guardados en *localStorage*. De encontrarlos, se reconstruye automáticamente la lista de materias y semestres, permitiendo al usuario retomar su horario exactamente donde lo dejó.

Sincronización al procesar nuevo *PDF*

Para evitar la pérdida del progreso del trabajo previo, la aplicación conserva las materias seleccionadas cuando se vuelve a cargar un archivo *PDF*. Este comportamiento es posible gracias a la comparación de los identificadores únicos de las materias previamente almacenadas con las del documento cargado.

Es importante señalar que esta conservación del progreso únicamente se presenta cuando el archivo *PDF* corresponde al mismo conjunto de materias, es decir, cuando se trata del mismo horario o de un documento con las mismas asignaturas. En caso de que se cargue un *PDF* diferente, que contenga materias distintas, el sistema no podrá restaurar las selecciones previas, ya que no existen coincidencias entre los identificadores de las asignaturas.

```
if (datosGuardados) {
    const materiasGuardadas = [...materias];
    materiasExtraidas.forEach(nuevaMateria => {
        const materiaGuardada = materiasGuardadas.find
        (m => m.id === nuevaMateria.id);
        if (materiaGuardada) {
            nuevaMateria.seleccionada = materiaGuardada.seleccionada;
        }
    });
}
```

Código 3.37: Fragmento de Horario.tsx

DatosGuardadosHorario.

En el código 3.37, cuando se carga un nuevo archivo *PDF*, la aplicación compara los identificadores únicos de las materias previamente almacenadas con los del documento cargado. Si existen coincidencias, se conserva el estado de selección de dichas materias,

evitando que se pierda el trabajo realizado anteriormente.

Este mecanismo funciona únicamente cuando el *PDF* corresponde al mismo conjunto de asignaturas; en caso de que el archivo contenga materias distintas, no es posible restaurar el progreso previo, ya que no se encuentran coincidencias entre los identificadores.

Eliminación y reinicio de datos

Además, se tiene el control total, se puede limpiar selecciones especificas o reiniciar toda la aplicación cuando se necesite, empezado desde cero.

```
const borrarHorario = () => {
    const nuevasMaterias = materias.map(materia => ({
        ...materia,
        seleccionada: false
    }));
    setMaterias(nuevasMaterias);
    localStorage.setItem(STORAGE_KEY, JSON.stringify(nuevasMaterias));
    setShowResetAlert(false);
};
const reiniciarAplicacion = () => {
    localStorage.removeItem(STORAGE_KEY);
    localStorage.removeItem(PERIODO_KEY);
    localStorage.removeItem(FECHA_KEY);
    setMaterias([]);
    setPeriodoActual("");
    setFechaActual("");
    setFile(null);
    setSemestreSeleccionado("Todos");
    setTurnoSeleccionado("Todos");
    setSemestres(["Todos", "Recurse"]);
    setDatosGuardados(false);
    setShowResetAlert(false);
```

```
23 };
```

Código 3.38: Fragmento de Horario.tsx

EliminacionyReinicio.

En el código 3.38, la aplicación permite tanto limpiar las selecciones específicas de materias como reiniciar completamente el estado del horario y los datos almacenados, garantizando que el usuario tenga control total sobre su planificación académica.

Gracias a estas funciones, el horario se conserva entre sesiones, se recupera automáticamente y se decide como gestionar las materias y semestres en todo momento.

3.5 Pruebas y validación

Para garantizar que la aplicación funcionara como se desea, se llevo a cabo un grupo de pruebas y verificaciones de manera ordenada. El objetivo era asegurar de que lo fundamental como elegir materias, identificar conflictos, horarios, y almacenar la información adecuadamente operar sin inconvenientes, y que los alumnos puedan utilizar la aplicación de manera confiable y fácil.

Figura 3.6: Pantalla de Inicio.

Como se muestra en la Figura 3.6, la pantalla de inicio de la aplicación educativa

está diseñada para estudiantes de Ingeniería en Computación, con un diseño limpio, profesional y una paleta de colores azul que transmite confianza y profesionalismo.

3.5.1 Tipos de pruebas realizadas

Se consideraron los siguientes tipos de pruebas:

1. Pruebas unitarias: Se evaluaron funciones individuales del código, como fue la validación de conflictos de horario, la selección de materias y la generación de IDs únicos, esto permitió detectar errores de lógica antes de integrarlas al sistema completo.

Figura 3.7: Diálogo de conflicto de horario activo.

La figura 3.7, se observa como el sistema detecta automáticamente los conflictos de horario. La ventana emergente muestra las materias en conflicto, el día y el horario exacto, así como las opciones disponibles para resolver el conflicto.

2. Pruebas de integración: Se verificó que los distintos componentes de la aplicación, como el calendario, la selección de unidades de aprendizaje y el almacenamiento en *localStorage*, trabajaran de manera coordinada, por ejemplo al seleccionar una materia en un semestre, la aplicación debía reflejar correctamente la selección en la interfaz y actualizar la persistencia de datos.

3. Pruebas funcionales o de usuario: Se realizaron escenarios completos en la aplicación para asegurar que la experiencia fuera intuitiva, algunos de los casos probados incluyen:

   - Selección de unidades de aprendizaje y visualización en el calendario.

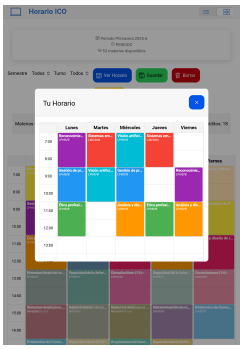

Figura 3.8: Ventana modal de visualización del horario generado.

En la Figura 3.8, el usuario puede revisar el horario generado antes de guardarlo o exportarlo, editarlo, garantizando que no existan conflictos y que la distribución de materias sea adecuada.

   - Detección de conflictos de horario y alertas emergentes.
   - Cambio de semestre y mantenimiento de las unidades de aprendizaje seleccionadas previamente.

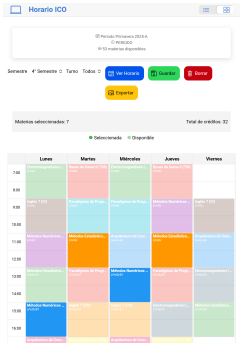

Figura 3.9: Cambio de semestre con materias seleccionadas.

Como se puede observar en la figura 3.9, al cambiar de semestre el sistema mantiene las unidades de aprendizaje seleccionadas previamente

y los créditos asociados, garantizando continuidad y consistencia en la planificación académica del estudiante.

- Exportación de horario y restaurante de los mismo desde un archivo.

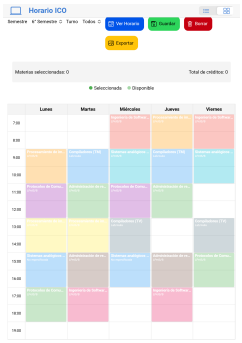

Figura 3.10: Pantalla principal de la aplicación mostrando el calendario.

Como se puede observa en la Figura 3.10, la pantalla principal de la aplicación de horarios muestra toda la información relevante para la gestión académica, incluyendo filtros por semestre y turno, botones de acción, panel de información y un calendario semanal con unidades de aprendizaje codificadas por color para fácil identificación.

Figura 3.11: Materias seleccionadas activamente.

En la Figura 3.11, la aplicación muestra las unidades de aprendizaje seleccionadas activamente, con colores saturados que indican claramente qué las unidades de aprendizaje han sido añadidas al horario y actualizan el panel de información con la cantidad de materias y créditos correspondientes.

4. Pruebas de usabilidad: Se evaluó que la interfaz fuera comprensible y que los mensajes de alerta fueran claros, especialmente en casos de conflictos de horario o al intentar guardar un horario sin materias seleccionadas.

3.5.2 Procedimiento de validación

El proceso de validación se realizo en varias etapas:

1. Preparación de escenarios: Se crearon horarios de pruebas con diferentes combinaciones de unidades de aprendizaje y horarios para simular situaciones reales de un semestre académico.

Figura 3.12: Escenario de prueba en el 5° semestre sin materias seleccionadas.

En esta Figura 3.12, al iniciar el 5° semestre sin seleccionar unidades de aprendizaje, todas las materias aparecen en colores pasteles claros, permitiendo al usuario identificar visualmente que están disponibles para su selección y sirviendo como base para los distintos escenarios de prueba del sistema.

2. Ejecución de acciones: Se interactuó con la aplicación como un usuario final, seleccionado y de seleccionando unidades de aprendizaje, cambiando semestres y verificando que las alertas emergentes aparecieran correctamente.

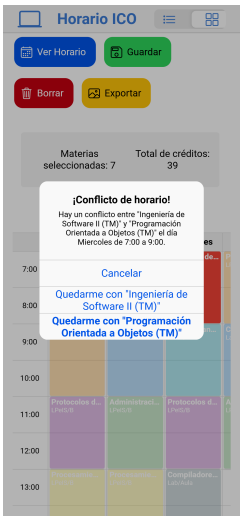

Figura 3.13: Conflicto de horario.

Como se muestra en la Figura 3.13, la aplicación presenta un diálogo de conflicto de horario cuando se detecta que dos unidades de aprendizaje se traslapan, permitiendo al usuario decidir cuál mantener y asegurando la coherencia del calendario.

3. Verificación de resultados: Se comprobó que:

   - La información se guarda en *localStorage*.
   - La exportación genera correctamente en el archivo final con los datos del horario.

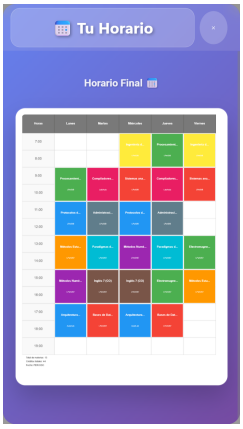

Figura 3.14: Vista del "Horario Final" de la aplicación.

En esta Figura 3.14, la aplicación muestra el "Horario Final" listo para captura de pantalla, con un diseño visual atractivo, colores diferenciados para cada unidad de aprendizaje y una tabla clara que permite ver toda la semana académica de un vistazo.

- Las unidades de aprendizaje conflictivas fueran detectadas y resaltadas en el calendario.
- Los cambios de semestre no provocaran perdida de datos ni inconsistencias.

3.5.3 Resultados obtenidos

Las pruebas realizadas permitieron validar que:

- Selección y visualización: Las unidades de aprendizaje se muestran correctamente en el calendario según el día y horario asignado.
- Detección de conflictos: La lógica de validación de horarios detecta superposiciones correctamente de manera precisa, mostrando alertas emergentes al usuario.
- Persistencia de datos: Al cerrar y abrir la aplicación, los horarios se mantienen correctamente gracias al uso del *localStorage*.
- Carga de nuevo *PDF*: La aplicación permite subir nuevamente el archivo *PDF* de horarios, conservando el progreso únicamente cuando se trata del mismo documento previamente cargado; en caso contrario, al detectar un *PDF* distinto con materias diferentes, el sistema reinicia la selección para evitar inconsistencias en los datos.

Figura 3.15: Borrar selección, reiniciar aplicación y subir nuevo *PDF*.

En la Figura 3.15, la aplicación presenta un cuadro de diálogo de confirmación al presionar el botón "Borrar", permitiendo al usuario decidir si desea eliminar únicamente las unidades de aprendizaje seleccionadas o reiniciar completamente la aplicación. Asimismo, en la interfaz mostrada se incluye la opción para subir un nuevo archivo *PDF*, lo que permite al usuario actualizar la información de las materias de manera controlada. Esta funcionalidad garantiza un mayor control sobre los datos almacenados, los filtros aplicados y la gestión del horario durante la planificación académica.

- Exportación de horarios: El archivo final refleja fielmente el horario seleccionado, incluyendo materias, días, horas y créditos totales.

- Interacción sin errores: No se detectaron fallas en la interfaz al cambiar semestres, filtrar materias o reiniciar el horario.

3.5.4 Conclusión de pruebas

El proceso de validación permitió verificar que la aplicación cumple con los requisitos establecidos. Se identificaron áreas menores de mejora relacionadas con la actualización

visual de componentes, las cuales fueron documentadas para futuras iteraciones. Se reviso todo, desde seleccionar unidades de aprendizaje, cambiar de semestre, validar los choques de horarios, hasta la parte de exportar el horario, y la mayoría de las funciones respondieron bien, aunque en algunas ocasiones los botones no hacían exactamente lo que uno esperaba el color de las materias no se actualizaban rápido.

También se realizaron pruebas en distintos escenarios de uso, como estudiantes con pocas materias, con una carga académica elevada y combinando asignaturas de diferentes semestres, verificando en todos los casos que la información se almacenara y recuperara correctamente. Durante estas pruebas se identificaron principalmente errores menores de visualización o de presentación del texto, los cuales no afectaron el funcionamiento general del sistema. No obstante, este proceso permitió comprender mejor el comportamiento de los usuarios dentro de la aplicación y fortalecer los mecanismos de validación, con el fin de prevenir la selección de unidades de aprendizaje con conflictos de horario y la pérdida de información al cambiar de semestre.

En resumen, la aplicación desarrollada es bastante útil, aunque no perfecta, y podemos decir que los estudiantes pueden usarla con confianza, que sus horarios se guardan y se muestran bien, pero igual con el tiempo seguro se encontraran detalles que se podrían mejorar. Se aprendió mucho sobre pruebas, sobre como revisar todo varias veces y la importancia de probar con situaciones reales y diferentes, porque uno cree que todo funciona y luego aparece un detalle que nadie había pensado.

Conclusiones y sugerencias

En esta tesis se presenta el desarrollo de una aplicación para dispositivos móviles con sistema operativo *Android*, que atiende una problemática frecuente en el programa educativo de Ingeniería en Computación en el proceso de inscripción. Este problema es el de elegir unidades de aprendizaje, evitando colisiones de horarios. Para ello, se usó *Ionic Framework* con *React* para crear la aplicación.

Usando como entrada al sistema un archivo *PDF* que contiene todos los horarios para el próximo semestre, y con el formato usado en los últimos años por la Coordinación del programa educativo, se extraen los nombres de las unidades de aprendizaje y los horarios. El sistema creado tiene la funcionalidad de detectar colisiones de horarios, notificando esto al usuario. Se realizaron pruebas de funcionamiento, encontrándose que el sistema realiza correctamente la carga, guardado de horarios creados y detección de traslape de horas en las clases elegidas.

A continuación, se presentan las conclusiones finales derivadas del desarrollo de este trabajo. La implementación de la aplicación para la organización de horarios no se limitó únicamente al desarrollo de funcionalidades técnicas, sino que también implicó el análisis de la experiencia del usuario, la correcta presentación de la información y la coherencia en cada uno de los procesos del sistema. Durante las pruebas realizadas, se comprobó que la selección de materias resulta sencilla e intuitiva, evitando conflictos de horario de manera efectiva. El uso de colores distintivos y alertas visuales facilita la identificación inmediata de conflictos entre asignaturas, permitiendo al usuario tomar decisiones informadas sin dificultad. Asimismo, se validó que la información se conserva

correctamente aun cuando la aplicación se cierra o se carga nuevamente un archivo *PDF* compatible, lo cual evita la pérdida de progreso y elimina la necesidad de repetir el proceso desde el inicio. De igual forma, la opción de exportar el horario final representa una funcionalidad relevante, ya que permite almacenar o compartir el resultado de la planificación académica. Finalmente, la aplicación presenta un desempeño ligero y estable, con botones funcionales, mensajes claros y sin errores al cambiar de semestre o reiniciar el horario. Estas características contribuyen a que el sistema sea accesible y fácil de utilizar, cumpliendo el objetivo de ofrecer una herramienta práctica que pueda ser utilizada por cualquier estudiante sin requerir conocimientos técnicos avanzados.

Como trabajo futuro, se identifican diversas mejoras que podrían incorporarse en versiones posteriores del sistema, entre las cuales se encuentran:

- Generar el horario final en formato *PDF*, permitiendo su descarga.
- Incorporar la opción de agregar notas rápidas en cada unidad de aprendizaje, con el objetivo de facilitar la organización académica del estudiante.
- Realizar pruebas en un mayor número de dispositivos, con el fin de garantizar un funcionamiento consistente del sistema en distintos entornos y configuraciones.
- Mejorar la gestión de las materias de recursamiento dentro de la aplicación, mediante una validación más precisa al momento de generar el horario. Asimismo, se propone que dichas materias se visualicen en un calendario específico de recursamiento o cuenten con una marca o anotación distintiva que las diferencie claramente de las demás unidades de aprendizaje, con el propósito de evitar inconsistencias y mejorar la experiencia del usuario.

En conclusión, el desarrollo de este proyecto permitió comprender que una aplicación funcional no solo debe cumplir con los requerimientos técnicos, sino también ofrecer una experiencia de uso clara, intuitiva y agradable. El cumplimiento de estos criterios demuestra que el sistema desarrollado logra un equilibrio entre utilidad y usabilidad, contribuyendo de manera efectiva a la organización académica de los estudiantes

Glosario de términos

API (Application Programming Interface): Interfaz que permite la comunicación entre aplicaciones mediante solicitudes estructuradas.

Aplicación web: Sistema accesible a través de un navegador que permite al usuario interactuar con información y servicios sin instalación local.

Calendario interactivo: Representación visual dinámica de eventos organizados por fecha y hora.

DOM (Document Object Model): Modelo que representa la estructura de un documento HTML como un árbol de objetos manipulable mediante código.

Framework: Conjunto de herramientas y convenciones que proporcionan una estructura base para el desarrollo de aplicaciones.

HTML (HyperText Markup Language): Lenguaje de marcado utilizado para estructurar el contenido de páginas web.

Ionic: Framework que permite desarrollar aplicaciones híbridas utilizando tecnologías web como HTML, CSS y JavaScript.

JSON (JavaScript Object Notation): Formato ligero de intercambio de datos utilizado para almacenar y transferir información estructurada.

localStorage: Mecanismo de almacenamiento del navegador que permite guardar información de forma persistente.

PDF (Portable Document Format): Formato de archivo utilizado para representar documentos de manera independiente del software o hardware.

Plugin: Componente adicional que extiende funcionalidades de un sistema sin modificar su estructura base.

React: Biblioteca de JavaScript utilizada para construir interfaces de usuario basadas en componentes reutilizables.

SDK (Software Development Kit): Conjunto de herramientas y bibliotecas que facilitan el desarrollo de aplicaciones para una plataforma específica.

Spinner: Indicador visual animado que señala que un proceso se encuentra en ejecución.

TypeScript: Lenguaje que extiende JavaScript incorporando tipado estático para mejorar la mantenibilidad del código.

UI (User Interface): Elementos visuales que permiten la interacción entre el usuario y la aplicación.

UX (User Experience): Experiencia general del usuario al interactuar con un sistema, considerando facilidad de uso y satisfacción.

Referencias

[1] Banks, A., and Porcello, E. *Learning React: Modern Patterns for Developing React Apps*. O'Reilly Media, 2020.

[2] Beizer, B. *Software Testing Techniques*, 2nd ed. Van Nostrand Reinhold, 1990. Segunda edición, a veces citada como 1995.

[3] Booch, G. *Object-Oriented Analysis and Design with Applications*, 2nd ed. Benjamin/Cummings, 1994. p. 72.

[4] Corporation, M. React native tools for visual studio code, 2023. Documentación oficial.

[5] Firtman, M. *Programming the Mobile Web*, 2nd ed. O'Reilly Media, 2013.

[6] Galitz, W. O. *The Essential Guide to User Interface Design*, 2nd ed. Wiley, 2002. enfatiza claridad y prevención de errores en interfaces gráficas.

[7] Galitz, W. O. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*, 3nd ed. Wiley, Indianapolis, IN, 2007.

[8] Google Developers. *Guide to Android Emulators*. Android Studio, 2024.

[9] Griffith, C. *Ionic in Action: Hybrid Mobile Apps with Ionic and Angular*. Manning Publications, Shelter Island, NY, 2017.

[10] Griffith, C. *Ionic Cookbook*. Packt Publishing, 2018.

[11] Jitterbit. Web application security best practices, 2024. Último acceso: 2024-10-18.

[12] Kaner, C., Falk, J., and Nguyen, H. Q. *Testing Computer Software*, 2nd ed. Wiley, 1999. p. 131.

[13] McConnell, S. *Code Complete: A Practical Handbook of Software Construction*, 1st ed. Microsoft Press, 1993.

[14] Mednieks, Z., Dornin, L., Meike, G. B., and Nakamura, M. *Programming Android: Java Programming for the New Generation of Mobile Devices*. O'Reilly Media, Sebastopol, CA, 2012. 2nd edition.

[15] Meier, R. *Professional Android 4 Application Development*. Wiley / Wrox, Indianapolis, IN, 2012. Versión digital (eBook).

[16] Pressman, R. S. *Ingeniería del software: un enfoque práctico*, 5th ed. McGraw-Hill, 2002. p. 135.

[17] Ray, J. *iOS 8 Application Development in 24 Hours, Sams Teach Yourself*. Pearson Education, Indianapolis, IN, 2015. Edición impresa y Kindle.

[18] Rumbaugh, J. E., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. *Object-Oriented Modeling and Design*. Prentice Hall, 1991. modelado anticipa comportamiento e interacción antes de codificar.

[19] Scharl, A., and Tochtermann, K. *The Geospatial Web: How Geobrowsers, Social Software and the Web 2.0 are Shaping the Network Society*. Springer, London, 2007. Versión digital en formato PDF.

[20] Schwaber, K. *Agile Project Management with Scrum*. Microsoft Press, 2004.

[21] Sommerville, I. *Ingeniería del Software*, 7th ed. Pearson Educación, 2005.

[22] Square, I. *Retrofit Documentation*, 2024. Documentación oficial.

[23] Yamacli, S. *Beginner's Guide to Android App Development: A Practical Approach for Beginners*. Createspace Independent Publishing Platform, 2017.