ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO PARA CONTROL DE ACCESO A AULAS Y REGISTRO DE ASISTENCIA DEL PERSONAL DOCENTE DE LA FIEE

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

BRYAN ARMANDO CARRIÓN TAPIA

MARÍA DOMÉNICA GÓMEZ SÁNCHEZ

DIRECTOR: ING. PABLO WILIAN HIDALGO LASCANO, M.Sc.

Quito, octubre 2020

AVAL

Certifico que el presente trabajo fue desarrollado por Bryan Armando Carrión Tapia y María Doménica Gómez Sánchez, bajo mi supervisión.

_____

Ing. PABLO WILIAN HIDALGO LASCANO, M.Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Nosotros, Bryan Armando Carrión Tapia y María Doménica Gómez Sánchez, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejamos constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

Bryan Armando Carrión Tapia	María Doménica Gómez Sánchez

DEDICATORIA

Este trabajo va dedicado a mis padres, ellos han forjado lo que soy hoy en día, todo se lo debo a ellos.

Bryan Armando Carrión Tapia

Con amor y humildad, a Dios, a mis Padres, a mis Viejitas. Todos y cada uno de mis logros pasados, presentes y futuros son por y para ustedes.

María Doménica Gómez Sánchez

AGRADECIMIENTO

A mis padres, por su incondicional e inagotable apoyo, lo que me ha ayudado a llegar al culmen de mi carrera. Gracias por siempre confiar en mí.

A Doménica, por su esfuerzo, dedicación y motivación para con nuestro proyecto. Gracias por compartir este logro conmigo.

Al Ing. Víctor Reyes, por su tiempo y por estar siempre presto a resolver las dudas de sus estudiantes.

Y al Ing. Pablo Hidalgo, por su guía y colaboración a lo largo de este trabajo.

Bryan Armando Carrión Tapia

AGRADECIMIENTO

A Dios y a la Virgen del perpetuo socorro por guiarme en todo momento y nunca desampararme.

A mis padres, por su incansable lucha por sacarnos adelante a mi hermano y a mí. Por ser mi mayor ejemplo de nobleza, tenacidad, responsabilidad y entrega.

A mis viejitas, por ser esas figuras maternas llenas de sabiduría, paciencia y dulzura, que me han acompañado a lo largo de toda mi vida.

A mis hermanos, y a toda mi familia, por su inquebrantable fe en mí y su constante apoyo.

A mi mayor confidente, mi fortaleza, gracias por tanta paciencia y entendimiento, por todo.
A mis amigos y amigas, gracias por todos los momentos compartidos, por complicidades, risas y llantos, son anécdotas que atesoraré siempre.

A los docentes de la Escuela Politécnica Nacional, quienes contribuyeron a mi preparación profesional y humana. Sin lugar a dudas tuve el honor de aprender mucho de ustedes.

A mi amigo y compañero de tesis, por el excelente trabajo en equipo, por el cariño, tiempo y esfuerzo dedicado a este Trabajo de Titulación, y por el apoyo mutuo que hemos sido desde que empezó este proyecto.

Un agradecimiento especial al Ing. Pablo Hidalgo, gracias por aceptar trabajar con nosotros, y brindarnos su confianza, guía y apoyo a lo largo de este proyecto. Porque además de ser un docente ejemplar, es un ser humano admirable.

María Doménica Gómez Sánchez

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

Actualmente la Facultad de Ingeniería Eléctrica y Electrónica (FIEE) de la Escuela Politécnica Nacional no dispone de ningún tipo de seguridad para controlar el acceso a las aulas de clases. Como consecuencia se puede observar el ingreso arbitrario de personas a éstas, lo que conlleva un peligro a la infraestructura de la Facultad, puesto que individuos no identificados con la comunidad Politécnica pueden ocasionar daños a las instalaciones o no preservar la integridad de las mismas.

Adicionalmente en la FIEE en la actualidad se maneja el registro de temas dictados y asistencia del personal docente de forma manual, lo cual además de ser un proceso engorroso para el personal docente y administrativo, puede ocasionar un sinnúmero de problemas como: No concordancia entre el registro de asistencia del docente y la ocupación del aula, errores en el registro de datos, olvido en el registro, pérdida de tiempo y recursos, entre otros.

Para solventar esto, se ha desarrollado un prototipo que integra software y hardware, el cual permite realizar el control de acceso a las aulas y el registro de asistencia a clases del personal docente de la FIEE. El sistema propuesto hace uso de lectores biométricos de huella dactilar, cerraduras electromagnéticas, un servidor centralizado que aloja el servicio web, una aplicación para administración, una aplicación móvil Android, y bases de datos alojadas en Firebase y SQL server para el almacenamiento de información.

Este prototipo busca: Mejorar la eficiencia en el proceso de registro, accesar a la información de forma instantánea, agilizar la generación de reportes, disminuir la cantidad de recursos empleados, aumentar la confidencialidad e integridad de la información, y brindar mayor seguridad a las aulas.

En el Capítulo I se expone el marco teórico para el desarrollo del prototipo, en el Capítulo 2 se provee el detalle de la metodología de desarrollo, diseño e implementación del prototipo. Las pruebas de funcionamiento y análisis de resultados son presentadas en el Capítulo 3. Finalmente, en el Capítulo 4 se presentan conclusiones y recomendaciones adquiridas en el desarrollo de este prototipo.

Como anexos se incluyen: Conceptos generales, encuesta, *scripts* de las bases de datos, proyectos de las dos aplicaciones, manual de usuario.

PALABRAS CLAVE: Control de Acceso, Registro de Asistencia, Lector Biométrico, React Native, Redux, Android, Firebase.

ABSTRACT

Currently the Faculty of Electrical and Electronic Engineering (FIEE in Spanish) of the National Polytechnic School does not have any type of security to control access to the classrooms. Therefore, the arbitrary entry of people can't be controlled, which entails a danger for the infrastructure of the Faculty, since individuals not identified with the institution can cause damage to the facilities or not preserve their integrity.

Additionally, the FIEE currently manages the attendance record of the teaching staff manually, which is a cumbersome process for the teaching and administrative staff. This can cause a several problems such as non-concordance between the attendance record of the teacher and the occupation of the classroom, errors in data recording, loss of time and resources, among others.

To solve these problems, a prototype that integrates software and hardware allowing the control of access to the classrooms and the registration of attendance to classes of the teaching staff of the FIEE was developed. The proposed system makes use of biometric fingerprint readers, electromagnetic locks, a centralized server hosting the web service, an administration application, an Android mobile application, and databases hosted on Firebase and SQL server for information storage.

This prototype seeks to: Improve efficiency in the registration process, obtain access to information instantly, speed up the generation of reports, reduce the amount of physical and human resources employed, increase the confidentiality and integrity of the information, and provide more safety to classrooms.

Chapter 1 exposes the theoretical framework for the development of the prototype. Chapter 2 provides the detail of the methodology of development, design and implementation of the prototype. Chapter 3 shows the analysis of results and performance tests. Finally, Chapter 4 presents conclusions and recommendations acquired through the development of the prototype.

The attachments included are: General concepts, the survey, database scripts, project files for both applications and the user manual.

KEYWORDS: Access Control, Attendance Record, Biometric Reader, React Native, Redux, Android, Firebase.

1. INTRODUCCIÓN

En este Capítulo se presenta una compilación de la información relevante y necesaria para el desarrollo del Sistema propuesto. Adicionalmente, se exponen los objetivos del proyecto, y su alcance, donde se especifican los componentes y funcionalidades del mismo.

En las siguientes subsecciones, se explican conceptos relacionados a las tecnologías y herramientas utilizadas, así como también la metodología de desarrollo SCRUM, en la que se basará el desarrollo del prototipo.

1.1 OBJETIVOS

El objetivo general de este Proyecto Técnico es:

- Desarrollar un prototipo para control de acceso a aulas y registro de asistencia del personal docente de la FIEE.

Los objetivos específicos del Proyecto Técnico son:

- Analizar el funcionamiento de los elementos de hardware y software necesarios para el desarrollo del proyecto propuesto.
- Diseñar los módulos que se incorporarán en el prototipo.
- Implementar los módulos del prototipo.
- Analizar los resultados en base a las pruebas ejecutadas.

1.2 ALCANCE

El proyecto propuesto integra distintos componentes de hardware y software para su realización. Este trabajo genera un prototipo conformado por lectores biométricos de huella dactilar de la marca ZKTeco, cerraduras electromagnéticas, un servidor centralizado que aloja un servicio web, una aplicación para administración que consume dicho servicio, una aplicación móvil, el servicio de Firebase y SQL server para el almacenamiento de datos. Debido a que las aulas del séptimo piso del edifico de Química/Eléctrica no tienen cableado estructurado, el prototipo se implementa solo en un aula del séptimo piso y se emula otra aula en una maqueta.

Para el inicio de cada semestre académico, el Subdecanato de la FIEE dispone la información de cada carrera, en la que se detallan materias, créditos, grupos, aulas, horarios, profesores, entre otros. El sistema desarrollado enviará automáticamente esta

información a la base de datos, para actualizar los dispositivos cada semestre y poder realizar las validaciones de control de acceso y registro de asistencia.

El procedimiento que el personal deberá realizar se detalla a continuación:

El docente se presenta en el aula en su horario de clases, coloca su huella en el lector biométrico, el cual contiene la información de los horarios respectivos para el aula de clases. Se realiza la verificación internamente en el sistema; si el docente está en su horario de clases se registra la asistencia con la hora exacta y la cerradura electromagnética se desbloquea permitiendo el acceso al aula, caso contrario la cerradura no se desbloquea impidiendo el acceso al aula.

Cabe recalcar que el docente puede acceder al aula en cualquier momento durante su horario de clases; en el sistema únicamente se registrará una inasistencia si el docente no se presentó.

Se tiene un servidor centralizado que controlará los lectores y el acceso a los datos, además una aplicación para la administración del sistema. Adicionalmente se dispone de una aplicación móvil destinada al sistema operativo Android para la visualización de información por parte del usuario.

El servidor se encuentra colocado dentro de la intranet de la EPN, éste tomará la información de los terminales biométricos y validará el acceso a las aulas; también a través de éste se manejará de manera remota a los dispositivos (terminales biométricos). Además éste brindará un servicio Web para el acceso a la base de datos. El servidor se encargará de guardar los datos en SQL Server, y una porción de ellos en Firebase para que puedan ser accesados por la aplicación móvil.

En la aplicación para administración que hace uso del servicio Web, el usuario puede acceder solo dentro de la intranet de la EPN. Se puede interactuar con el sistema mediante diversos módulos del software, que se mencionan a continuación:

- Módulo Usuario: En este módulo se tienen los submódulos: Crear Usuarios, Editar Usuarios. Un usuario tiene como información sus nombres y apellidos, cédula de ciudadanía, fotografía, huella dactilar, estado (Activo/Inactivo), nivel de acceso (Administrador/Docente).
    - Submódulo Crear Usuarios: Al crear un usuario se debe insertar toda la información antes mencionada.
    - Submódulo Editar Usuarios: Por lo general servirá para agregar la huella dactilar de los usuarios a la base de datos; sin embargo, también se pueden realizar modificaciones en los datos antes mencionados. En este módulo

también se puede inhabilitar usuarios, es decir poner en estado Inactivo al usuario debido a varias situaciones que se pueden presentar como renuncias, retiros inusuales, etc. No se pueden eliminar usuarios puesto que eliminar *data* no es una buena práctica, solo se podrán inhabilitarlos con la opción a habilitarlos posteriormente. [1]

- Módulo Aulas: En este módulo se tienen los submódulos: Agregar Aulas, Editar Aulas, Mostrar Aulas, Cambios y Reservas. Un aula tiene como información nombre, capacidad, y los grupos que contendrá en un semestre específico.
    - Submódulo Mostrar Aulas: Se presenta la lista de aulas, en la cual se debe seleccionar un aula, y se puede visualizar la información del aula, es decir la capacidad, y el horario permanente del aula.
    - Submódulo Cambios: Permite realizar cambios permanentes; este submódulo se crea para casos en que los docentes por distintos motivos quieren hacer un cambio para todo el periodo académico en las aulas en las que deben dar clases o en el horario en el cual se dictará la clase. En este caso el aula en la que ellos deberían dar clase originalmente en ese horario quedaría disponible, y se les asigna una nueva aula que esté disponible en el horario requerido; o dado el caso que no deseen cambiar el aula sino modificar el horario en el que dictarán la clase en esa aula, de igual manera se les asigna un nuevo horario que esté disponible en el aula en cuestión.
    - Submódulo Reservas: Permite reservar un aula para una fecha o intervalo de tiempo específico en el período académico. Para este caso el docente únicamente debe detallar, en caso de que dicte varias materias, la materia de la que desea recuperar o adelantar clases, el sistema inmediatamente devolverá el GR (Grupo) del docente en dicha materia, obteniendo toda la información del GR que necesita, en este caso la cantidad de estudiantes que éste tiene.

      Para este submódulo se presentan dos formas de realizar las reservas; en el primer caso el docente puede solicitar un aula en específico, por tanto al docente se le mostrará en qué horario dicha aula estará disponible para proceder a realizar la reserva; en el segundo caso el docente presentará un horario en específico en el que desea reservar un aula, en este caso el sistema devolverá una lista de aulas con la capacidad suficiente para la cantidad de estudiantes del GR que recuperará o adelantará clases y que estén disponibles

en el horario escogido, para que el docente o el administrador escojan el aula a criterio propio.

- Módulo Dispositivos: En este módulo se tienen los submódulos: Agregar Dispositivos, Manejo de Dispositivos. Un dispositivo tiene como información la dirección IP, dirección MAC, código de dispositivo, fabricante, tipo de dispositivo, número serial y el aula en la que se le instalará.
    - Submódulo Agregar Dispositivos: Se deberá insertar la dirección IP que se le asignará al dispositivo, y el aula a la que pertenecerá, puesto que el resto de datos el sistema los recupera directamente del dispositivo ya que es información que contiene por defecto el dispositivo.
    - Submódulo Manejo de Dispositivos: Se presentará una lista de dispositivos en la cual se seleccionará un dispositivo con la información actual de dicho dispositivo; posteriormente se podrán configurar parámetros y realizar acciones de manera remota en este dispositivo como actualizar fecha y hora, modificar IP, verificar estado del dispositivo, sincronizar dispositivo con la base de datos (en caso de que haya existido un error interno), habilitar/deshabilitar dispositivo, reiniciar dispositivo, verificación 1:1 (dispositivo-base de datos), resetear dispositivo, y desbloquear la cerradura del dispositivo seleccionado.
- Módulo Reportes: En este módulo se tienen los submódulos: Reportes de Asistencia, Reportes de Aulas.
    - Submódulo Reportes de Asistencia: Permite generar los reportes discriminando por profesor y por materias; se podrán observar los registros de asistencia con fecha, hora de entrada y hora de salida, visualizar la totalidad de horas asistidas, y obtener estadísticas.
    - Submódulo Reportes de Aulas: Presenta la lista de aulas, de las cuales se podrá visualizar el porcentaje de utilización.

En ambos submódulos los reportes serán visualizados en formato PDF, para posteriormente imprimir si se lo requiere.

En la Aplicación móvil, el docente deberá realizar el proceso de ingreso, mediante su usuario y contraseña, desde cualquier lugar a través de Internet; posteriormente se podrá interactuar con el sistema mediante los siguientes módulos:

- Módulo Reportes: En el cual el docente puede generar y visualizar su reporte de asistencia de cada asignatura o de forma general de todas sus asignaturas dentro

de un periodo de tiempo determinado por el usuario; por ejemplo, se podrán realizar reportes diarios, semanales, mensuales, y semestrales.

- Módulo Aulas: En el cual al docente se le presenta una lista de aulas de la FIEE y un calendario; el docente seleccionará un aula y una fecha; posteriormente podrá visualizar el horario del aula actualizado (horario permanente del aula y las reservas) para la semana de la fecha seleccionada.
- Módulo Solicitud de Reserva: En el cual el docente ingresará datos sobre el aula que desea reservar tales como materia, grupo, fecha, y hora.

Cabe recalcar que se ha contemplado el posible caso de un fallo de energía en la facultad, por lo cual el lector biométrico contará con una batería interna de respaldo para continuar operando por aproximadamente 5 horas.

El desarrollo del presente trabajo de titulación está basado en la metodología SCRUM. Se realizan encuestas a las autoridades de la FIEE con el fin de recopilar información para poder obtener las historias de usuario. [2]

El proyecto tiene un producto final demostrable. El esquema del prototipo propuesto se presenta en la Figura 1.1, mismo que está direccionado únicamente a ser instalado en un aula de clase.

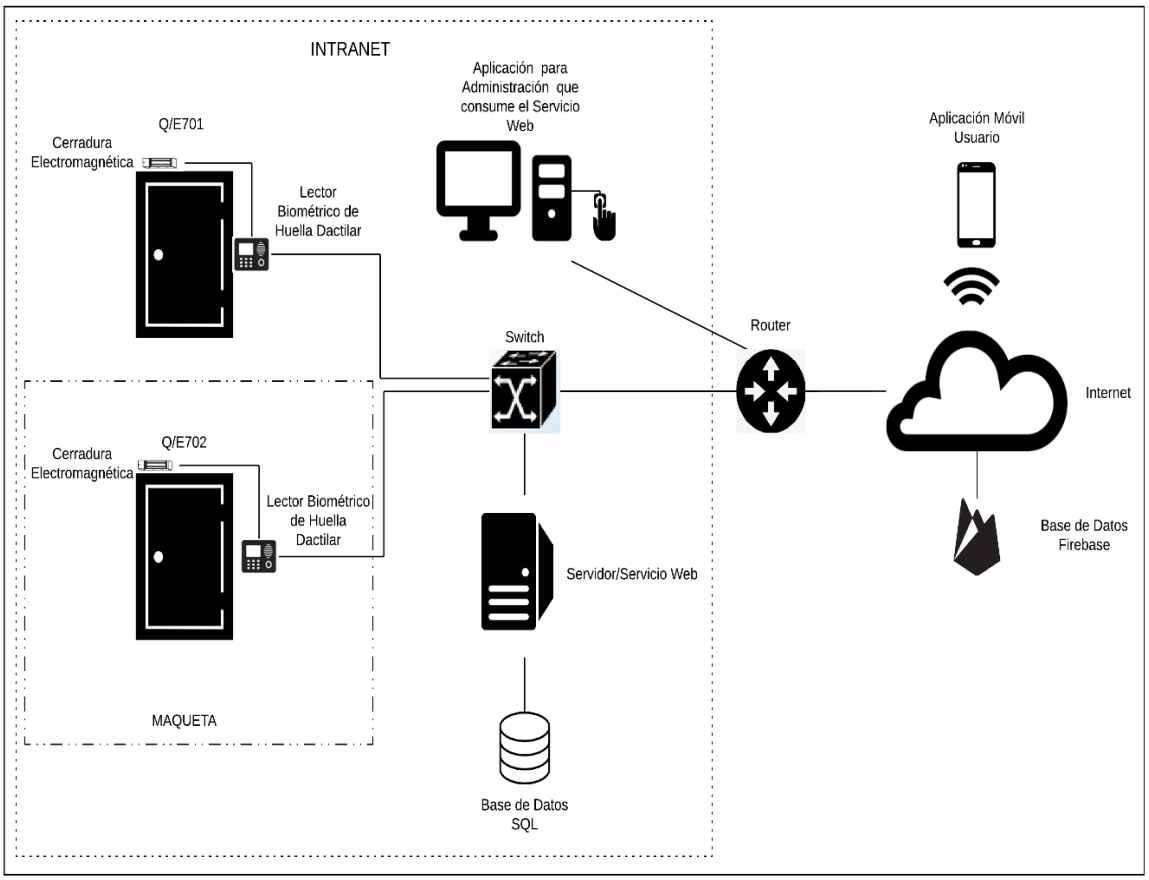


Figura 1.1. Esquema del prototipo

1.3 MARCO TEÓRICO

En esta sección se detallan conceptualmente el conjunto de herramientas y tecnologías de hardware y software utilizadas para el diseño y desarrollo del prototipo. Adicionalmente se explica brevemente la metodología a emplear para el producto software.

1.3.1 BIOMETRÍA

El concepto de biometría proviene de las palabras bio (vida) y metría (medida), es decir la aplicación de matemáticas y estadísticas al análisis de características morfológicas o conductuales únicas de un individuo, con el fin de la identificación del mismo [3].

La biometría se clasifica en:

- Biometría Estática: Ésta se encarga de los rasgos morfológicos únicos que tiene un individuo para ser identificado (por ejemplo: Huellas dactilares, rasgos de la mano o de la cara, patrones del iris).
- Biometría Dinámica: Ésta se encarga de estudiar los rasgos conductuales del individuo con el fin de identificar los comportamientos únicos que lo diferencian del resto (por ejemplo: Patrones de voz, patrones de firma ológrafa, patrones de tipeo) [4] [3] [5].

Un rasgo morfológico tiene la cualidad de ser relativamente estable en el tiempo. Los rasgos conductuales son menos estables, pues depende de la disposición psicológica de la persona. Esta es la razón por la cual no cualquier característica puede ser utilizada con éxito como patrón para un sistema biométrico [6].

Para encontrar información más detallada, ver sección 2 del anexo A.

1.3.2 SISTEMAS BIOMÉTRICOS

Un sistema biométrico es un método automático de reconocimiento, verificación o identificación de un individuo utilizando características únicas e intransferibles del mismo, ya sean físicas o conductuales.

El método que utilizarán los dispositivos biométricos que serán integrados en el sistema es de identificación. El proceso de identificación puede ser llevado a cabo mediante tres indicadores de identidad:

- Conocimiento: Se realiza la autenticación mediante algo que se sabe (clave, PIN).
- Posesión: La autenticación se ejecuta mediante algo que se posee (una tarjeta).

- Característica: El individuo se autentica mediante una característica biométrica inherente al mismo, que puede ser verificada (huellas dactilares, iris).

Estos indicadores pueden ser combinados con el fin de incrementar el nivel de seguridad.

Los procesos basados en conocimiento y en posesión requieren que la persona que se va a autenticar sepa el código o lleve consigo el dispositivo. En el primer caso el individuo es susceptible de olvidar el código o de divulgarlo; en el segundo caso el dispositivo puede extraviarse, ser sustraído y/o duplicado. De igual forma en ambos casos el vínculo entre la información y su verificación es débil, lo que facilita la usurpación de identidad.

Esta es una de las razones por las cuales la identificación mediante características biométricas es la más utilizada y recomendada, puesto que el indicador es único e intransferible.

1.3.2.1 Huella dactilar

Partiendo de la selección del rasgo morfológico, misma que fue desarrollada a detalle en la sección 2.1.2 del anexo A, se escogió la huella dactilar como patrón a utilizar.

La huella dactilar es la representación de la morfología superficial de la epidermis de un dedo y es un patrón único en cada individuo; aparece en la etapa fetal en el sexto mes y no cambia a lo largo de la vida del mismo. Está conformada por varios grupos de líneas que llevan el nombre de crestas (trazos oscuros) y valles (trazos claros), tal como se muestra en la Figura 1.2.

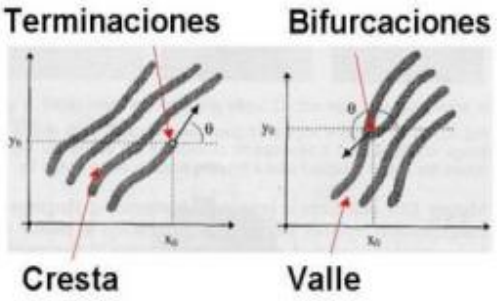


Figura 1.2. Minucias [7]

1.3.2.2 Selección del sistema biométrico

Una vez seleccionado el rasgo morfológico más adecuado para la aplicación, se debe complementar la decisión a tomar, con un análisis comparativo para la selección del sistema que se encargará de recibir y procesar el patrón.

Los distintos escenarios que pueden presentarse requerirán diferentes tipos de sistemas biométricos; los criterios de evaluación pueden variar levemente de un sistema a otro. Con

el fin de realizar un análisis comparativo entre los sistemas biométricos más desarrollados, la Tabla 1.1 compara las diferentes propiedades de los sistemas biométricos basada en las características descritas en las secciones 2.1.2 y 2.1.3 del anexo A [8].

Tabla 1.1. Comparación entre sistemas biométricos [8]

Sistema biométrico / **Característica**	**Iris**	**Huella dactilar**	**Geometría de la mano**	**Voz**	**Rostro**
Universalidad	Alta	Alta	Media	Media	Media
Fiabilidad	Alta	Alta	Alta	Alta	Alta
Facilidad de uso	Media	Alta	Alta	Alta	Alta
Prevención de ataques	Alta	Alta	Alta	Media	Media
Aceptabilidad	Media	Media	Media	Alta	Alta
Permanencia	Alta	Alta	Media	Media	Media
Costo	Alto	Medio	Alto	Bajo	Medio

Se puede observar, que el sistema biométrico de huella dactilar seleccionado, es el que cumple de mejor manera con las características biométricas enlistadas en la Tabla 1.1, en contraste al resto de sistemas. Adicionalmente posee un promedio de costos razonable teniendo en cuenta sus prestaciones, debido a esto es la opción más adecuada para este proyecto.

Para encontrar información más detallada, ver sección 2.1.3 del Anexo A.

1.3.2.3 Arquitectura del sistema biométrico de huella dactilar

El sistema biométrico de huella dactilar está compuesto fundamentalmente de cinco módulos (Figura 1.3):

1. Módulo de Captura: Módulo que permite la captura del rasgo morfológico del individuo.

2. Módulo de extracción de características: La información obtenida es procesada con el objetivo de extraer las características principales de la misma, y crear una plantilla o *template* (representación de la huella dactilar) de la huella dactilar.

   Cabe mencionar que una plantilla está conformada por un subconjunto de la información obtenida de la biometría; por tanto, no es posible recuperar una huella dactilar a partir de una plantilla, incrementando así el nivel de seguridad.

3. Módulo de coincidencia: Se obtiene una puntuación al realizar la comparación de la plantilla de entrada con las plantillas existentes en la base de datos.

4. Módulo de base de datos: En este módulo las plantillas de usuarios son almacenadas en la base de datos.
5. Módulo de toma de decisiones: Se retorna una respuesta afirmativa o negativa al usuario, en base a las comparaciones realizadas.

Este conjunto de módulos, tienen como finalidad dos tareas principales:

- Tarea de Inscripción: Se añade un nuevo usuario a la base de datos, registrando su plantilla y su identidad.
- Tarea de Reconocimiento: Se toma decisiones respecto a la identidad del usuario, comparando la plantilla de entrada con las plantillas existentes en la base de datos; como criterio fundamental se tendrá el grado de semejanza entre las mismas.

Para encontrar información más detallada, ver Anexo A, sección 2.1.3.2.

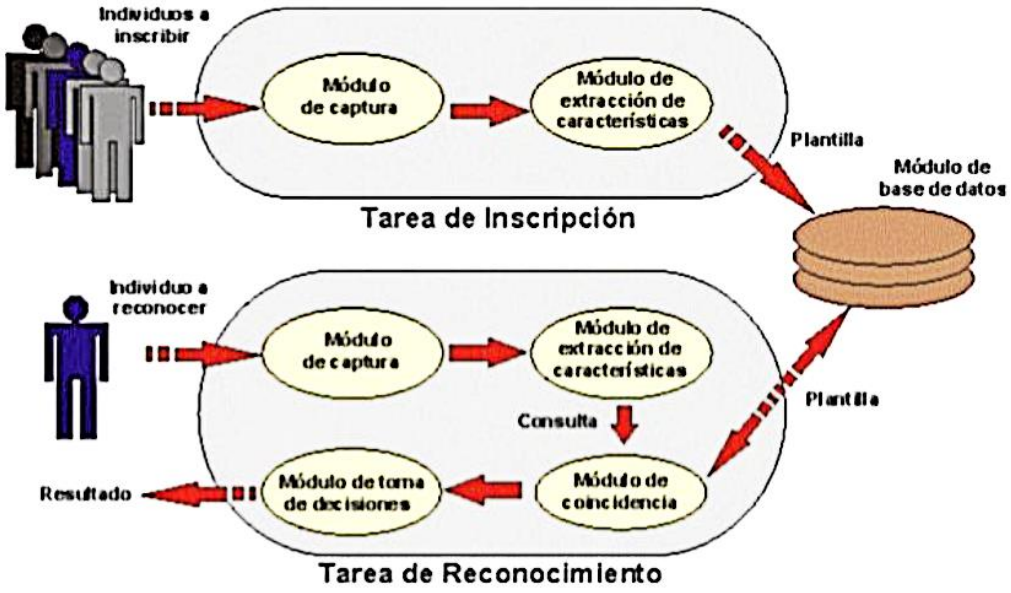


Figura 1.3. Módulos del sistema biométrico [9]

1.3.3 KITS DE DESARROLLO DE SOFTWARE (SDK)

1.3.3.1 ZKFinger Windows

Este SDK permite el desarrollo de software en distintos lenguajes de programación (Java, C++, C#) para la obtención de la imagen de una huella dactilar y su plantilla a través de los escáneres de la serie ZK de la marca ZKTeco. Dicho kit es compatible para versiones

de Windows desde XP en adelante [10]. Este kit tiene como requerimiento de software .NET Framework 3.5 o superior.

1.3.3.1.1 Funcionalidades

- Inicializar recursos de memoria y conexión hacia el escáner biométrico.
- Capturar imagen y plantilla de huella dactilar.
- Realizar comparaciones de distintas muestras de una huella.

Las funciones a utilizarse de este kit se encuentran de forma detallada en el Anexo A, sección 2.1.4.1, Tabla 2.5.

1.3.3.2 SDK Standalone

El SDK Standalone permite el intercambio de información con dispositivos de huellas dactilares, dispositivos de control de acceso y dispositivos de tarjeta RFID. Permite desarrollar software en distintos lenguajes de programación (Java, C++, C#) para administrar la información y las huellas digitales del usuario, descargar registros de asistencia, entre otros. Este kit es compatible con Windows [11].

1.3.3.2.1 Funcionalidades

- Carga y descarga de información de usuarios, huellas, tarjetas.
- Configurar reglas de acceso en los dispositivos.
- Configurar hora, direcciones IP, entre otros.
- Accionar eventos en tiempo real, como la verificación de huella.

Las funciones a utilizarse de este kit se encuentran de forma detallada en el Anexo A, sección 2.1.4.2, Tabla 2.6.

1.3.4 ARQUITECTURA CLIENTE-SERVIDOR

La arquitectura cliente/servidor, tiene importancia significativa en la arquitectura de software moderna. Aquí el usuario opera el programa de aplicación (cliente) en su computadora y este programa accede a los recursos del servidor. Los recursos se administran, comparten y se ponen a disposición centralmente. Como se puede ver en la Figura 1.4, el modelo cliente / servidor se basa en un esquema simple de solicitud-respuesta. Esto significa que los archivos ya no tienen que transferirse como un todo; las solicitudes se pueden colocar específicamente. Por ejemplo, el servidor a menudo accede a una base de datos (relacional) y ejecuta una consulta especificada en la solicitud del cliente [12].

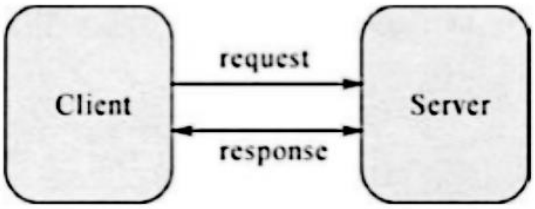


Figura 1.4. Representación arquitectura cliente/servidor [12]

El servidor gestiona los recursos del sistema, cuyas tareas principales son:

- Garantizar la seguridad de la base de datos.
- Controlar el acceso concurrente a la base de datos.
- Garantizar la coherencia de los datos.

Las tareas principales del programa de aplicación del cliente son:

- Proporcionar una interfaz gráfica de usuario.
- Enviar las solicitudes del usuario al servidor y recibir respuestas de éste.
- Ejecutar el procesamiento de los datos recuperados de la base de datos.

La transmisión de datos se completa mediante un mecanismo de comunicación de red para reducir la carga en el servidor. El flujo de datos de esta arquitectura se presenta en la Figura 1.5.

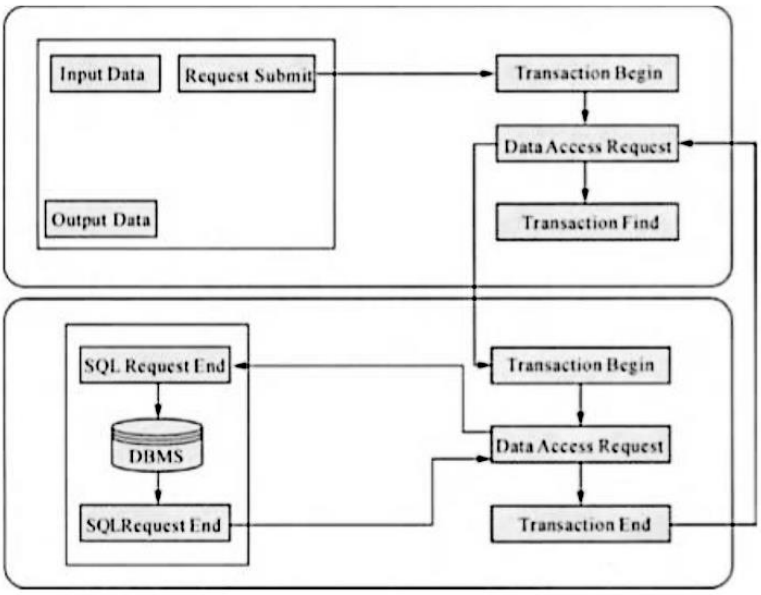


Figura 1.5. Manejo de peticiones en la arquitectura cliente/servidor [13]

Las ventajas de esta arquitectura son dominantes. La más significativa es la adaptabilidad y la flexibilidad aportadas por el aislamiento definido de los componentes. De hecho, la arquitectura Cliente/Servidor posee una poderosa capacidad de operación de datos y procesamiento de transacciones [13].

1.3.5 FIREBASE

Firebase es un *Backend-as-a-Service* (BaaS), es decir es un servicio creado por Google, y construido sobre su infraestructura, para vincular aplicaciones creadas por desarrolladores al *backend* en la nube. Firebase proporciona una variedad de herramientas para dichas aplicaciones.

Entre las herramientas principales se tienen:

- Autenticación: Posibilita la autenticación mediante contraseñas, números de teléfono, redes sociales, Google, entre otras. Adicionalmente permite integrar uno o más métodos de autenticación en una aplicación.
- Bases de Datos en tiempo real: Es una base de datos No-SQL que se encuentra en la nube, ésta almacena documentos en formato JSON.
    - Estos documentos son un conjunto de pares clave-valor, y el conjunto de documentos forma una colección. Los datos se sincronizan en todos los usuarios en tiempo real y se encuentran disponibles en forma permanente.
- *Hosting*: Proporciona alojamiento de forma sencilla para una aplicación web, el contenido se almacena en caché en las redes de entrega de contenido a nivel mundial.
- *Cloud Storage*: Proporciona el servicio de almacenamiento de información de todo tipo de contenido.
- *Cloud Functions*: Este servicio escala automáticamente los recursos informáticos para que coincidan con los patrones de uso de la aplicación. Es por esto que los desarrolladores al usar Firebase, no se preocupan por la seguridad y configuración del servidor, aprovisionamiento de nuevos servidores o el desmantelamiento de los antiguos [14] [15].

Una información más detallada de los tipos de bases de datos utilizados en este proyecto se encuentra en el Anexo A, secciones 4 y 5.

1.3.6 VISUAL STUDIO CODE

Visual Studio Code integra la simplicidad de un editor de código fuente con herramientas poderosas de desarrollo de software, como es la finalización y depuración de código IntelliSense, el cual es una ayuda para completar el código que incluye una lista de características: Lista de miembros, información de parámetros, información rápida y palabra completa [16], [17].

Estas características ayudan al desarrollador a obtener más información sobre el código que está utilizando, realizar un seguimiento de los parámetros que está escribiendo y agregar llamadas a propiedades y métodos con solo unas pocas teclas.

Es compatible con Windows, macOS, y Linux, e incluye soporte para JavaScript y Node.js. Adicionalmente proporciona un gran número de extensiones para otros lenguajes como (C++, C#, Java, PHP) [16].

Visual Studio Code similar a otros editores de código presenta una interfaz de usuario simple e intuitiva. La interfaz de usuario se divide en cinco áreas [18] como se puede observar en la Figura 1.6:

- Editor de Código (C): En esta sección se realiza la edición de archivos, se puede abrir un sinnúmero de archivos y visualizar su contenido.
- Barra Lateral (B): La cual presenta diferentes vistas, una de ellas el explorador, que muestra todas las carpetas y archivos a los que se tiene acceso.
- Barra de Estado (E): Contiene información sobre el proyecto abierto y los archivos que se están editando.
- Barra de Actividad (A): La cual permite cambiar de vistas.
- Paneles (D): Muestra diferentes paneles debajo del editor que contienen información de salida o depuración, así como errores y advertencias.

Cada vez que se inicia Visual Studio Code, éste conserva el estado que tenía la última vez.

Más información acerca del resto de herramientas de programación utilizadas en este proyecto, se detalla en el Anexo A, sección 6.

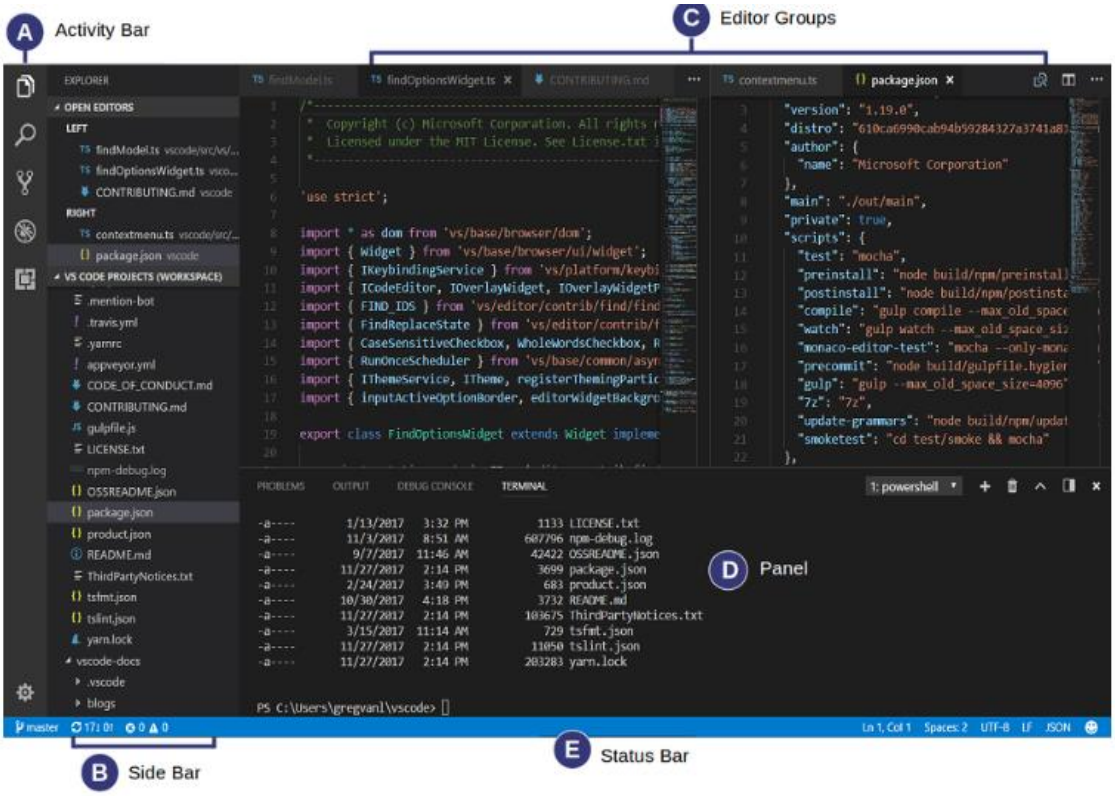


Figura 1.6. Interfaz gráfica de Visual Studio Code [18]

1.3.7 REACT NATIVE

React Native es un *framework* de Javascript utilizado para la creación de aplicaciones móviles en Android y iOS, en el cual se utilizan tecnologías web estándar para la construcción de las mismas, tales como: HTML, JavaScript, etc. [19].

Integra lo mejor del desarrollo nativo brindado por React, con lo mejor proporcionado por Javascript para la creación de interfaces de usuario [20].

React Native permite crear aplicaciones móviles que se ven, se sienten y funcionan mucho más como aplicaciones nativas que las aplicaciones web típicas, porque la tecnología central detrás de esto es realmente nativa. Posibilita a los desarrolladores hacer esto mientras que continúan utilizando la mayoría de las mismas herramientas de desarrollo web que han usado a lo largo de los años y al mismo tiempo permite que ese desarrollo sea multiplataforma [19].

Información acerca de Javascript, que es el lenguaje sobre el que se desarrolla este *framework*, se encuentra en el Anexo A, sección 8.

1.3.7.1 Redux

Redux es un contenedor de estado predecible para aplicaciones JavaScript, lo que significa que permite escribir aplicaciones que se comporten de manera consistente en

diferentes entornos: Cliente, servidor o nativo. También hace que las aplicaciones sean fáciles de probar y depurar [21] . Redux tiene 3 conceptos clave: *Actions, reducers* y *store* (Figura 1.7).

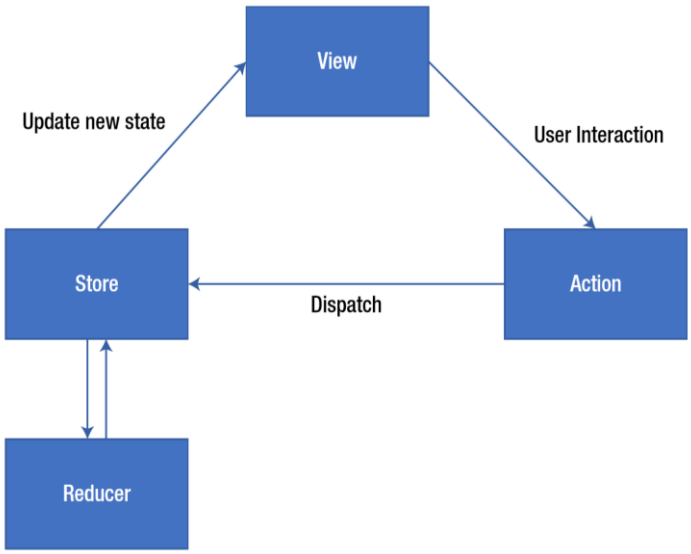


Figura 1.7. Flujo de datos de Redux [21]

- *Actions*: Son eventos que envían datos desde la aplicación (interacciones del usuario, llamadas a una API, etc.) a la *Store*, la cual siempre obtiene la información de las *Actions*. Las *Actions* son objetos JavaScript simples que tienen una propiedad de tipo, que describe el tipo de acción y la carga de información que se envía a la *Store*. Para enviar a la *Store* se usa el *Dispatcher* que es el que maneja todo el flujo de datos.
- *Reducers*: Especifican cómo cambia el estado de la aplicación en respuesta a las *Actions* enviadas a la *Store*, debido a que estas últimas solo describen lo que sucedió.
- *Store*: Es un objeto que contiene todo el estado de la aplicación y además provee eventos que ayudan a acceder al estado y despachar *Actions.*

1.3.8 SERVICIO SOAP

SOAP (*Simple Object Access Protocol*) es un protocolo basado en XML para el intercambio de información entre computadoras. SOAP se puede usar en una variedad de sistemas de mensajería y se puede entregar a través de una variedad de protocolos de transporte; es independiente de la plataforma y, por lo tanto, permite que diversas aplicaciones se comuniquen.

El enfoque inicial de SOAP son las llamadas a procedimientos remotos transportadas a través de HTTP. SOAP, por lo tanto, permite que las aplicaciones cliente se conecten fácilmente a servicios remotos e invocar métodos remotos. Por ejemplo, una aplicación cliente puede agregar inmediatamente la traducción de idiomas a su conjunto de características localizando el servicio SOAP correcto e invocando el método correcto.

Otros entornos de trabajo, incluidos CORBA, DCOM y Java RMI, proporcionan funcionalidades similares para SOAP. Los mensajes SOAP están escritos completamente en XML y, por lo tanto, son independientes de la plataforma y el lenguaje [22].

SOAP, por consiguiente, representa una piedra angular de la arquitectura de servicios web, lo que permite diversas aplicaciones para intercambiar fácilmente servicios y datos.

SOAP ha recibido un amplio apoyo de la industria. Existe un número importante de implementaciones SOAP, incluidas las implementaciones para Java, COM, Perl, C #, y Python, de la misma forma, cientos de servicios SOAP han florecido en toda la Web.

1.3.9 METODOLOGÍA SCRUM

Un proyecto Scrum se traduce en un esfuerzo de colaboración para crear un nuevo producto, servicio u otro, dependiendo de lo definido en la declaración de la visión del Proyecto. Los proyectos se ven afectados por limitaciones temporales, económicas, organizacionales, de alcance, calidad, recursos u otras que hacen que sea complicada la planeación, ejecución, y administración del proyecto para finalmente tener éxito. Sin embargo, la implementación exitosa de un proyecto proporciona beneficios comerciales significativos a una organización. Por lo tanto, es importante que las organizaciones seleccionen y practiquen un enfoque apropiado de gestión de proyectos.

Scrum es una de las metodologías ágiles más populares. Es un marco adaptativo, iterativo, rápido, flexible y efectivo diseñado para ofrecer un valor significativo a un proyecto. Scrum garantiza la transparencia en la comunicación y crea un entorno de responsabilidad colectiva y progreso continuo.

Scrum está estructurado de tal manera que apoya el desarrollo de productos y servicios en todo tipo de industrias y en cualquier tipo de proyecto, independientemente de su complejidad. Una fortaleza clave de Scrum radica en su uso de equipos multifuncionales, auto-organizados y empoderados que dividen el trabajo en ciclos de trabajo cortos y concentrados llamados *Sprints*. La Figura 1.8 proporciona una visión general del flujo de un proyecto Scrum.

El ciclo Scrum comienza con una reunión de partes interesadas, durante la cual se crea la visión del proyecto. Luego, el propietario del producto junto con el *Scrum Master* desarrollan el *Product Backlog* que es una lista priorizada de requisitos en forma de Historias de Usuarios. Cada *sprint* comienza con una reunión de planificación de *sprint* durante la cual se consideran las historias de usuarios de alta prioridad para su inclusión en el *sprint*.

Un *sprint* generalmente dura entre una y seis semanas e implica que el equipo Scrum trabaje para crear entregables o incrementos de producto potencialmente entregables [23]. Durante el *sprint*, se llevan a cabo Reuniones Diarias cortas y altamente enfocadas donde los miembros del equipo discuten el progreso diario.

Hacia el final del *sprint*, se lleva a cabo una reunión de revisión, donde el equipo Scrum se pone al tanto de los avances. Posteriormente se realiza la actualización del *Product Backlog*.

El propietario del producto y las partes interesadas aceptan los entregables solo si cumplen con los criterios de aceptación predefinidos [23].

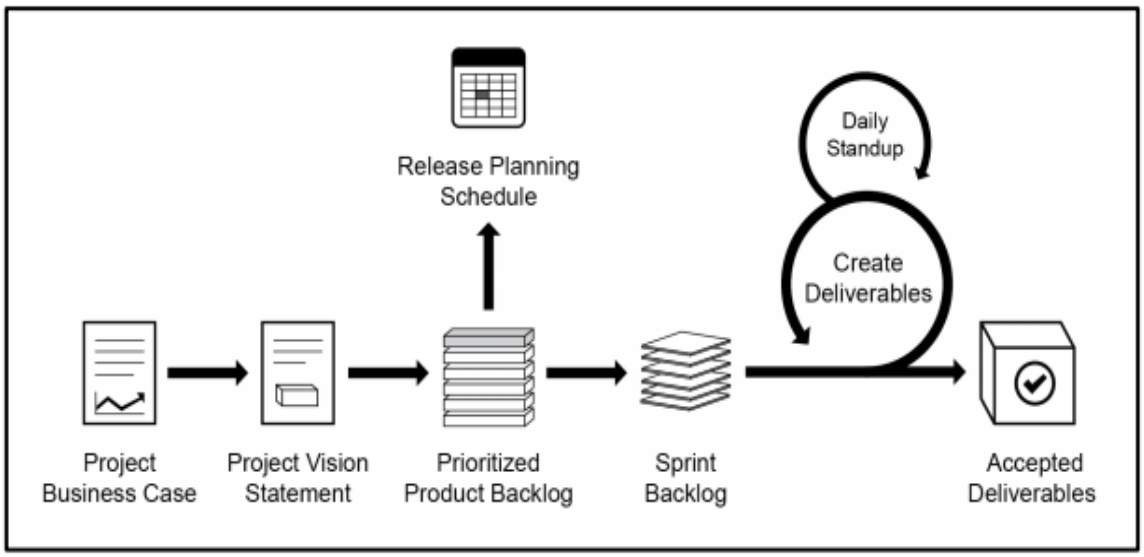


Figura 1.8. Flujo Scrum para un sprint [23]

En la Tabla 1.2 se enlistan los principales procesos Scrum que se pueden aplicar a cualquier tipo de proyecto, los mismos que se encuentran agrupados en 4 fases para una mayor comprensión.

Tabla 1.2. Fases Scrum

Fase	Proceso SCRUM fundamental
Inicial	• Crear la visión del proyecto, que servirá de inspiración y proveerá una guía clara durante todo el proyecto • Identificar al *Scrum Master* (persona que lidera y asesora a los equipos) y a las partes interesadas, mediante un criterio específico. • Formar el equipo Scrum. El propietario del producto y el Scrum Master se encargan de la selección de los miembros del equipo. • Crear las historias de usuario, diseñadas por el propietario del producto con la ayuda del equipo Scrum para asegurarse que los requerimientos del cliente estén claramente plasmados y completamente comprendidos por las partes interesadas; posteriormente estas historias se incorporan al *Product Backlog*.
Planeación y Estimación	• Identificar Tareas. En este proceso se transforma la historia de usuario en una lista de tareas específicas, y se evalúa el esfuerzo estimado que éstas conllevarán. El equipo Scrum revisa estas historias con el propósito de generar un calendario de planificación de entrega, que es básicamente un calendario de implementación que puede ser compartido con los interesados. • Crear el *Product Backlog*, que contiene los requerimientos del proyecto definidos y priorizados. • Crear *sprint Backlog*. En este proceso el equipo Scrum crea el *sprint backlog* que contiene todas las tarea a ser completadas en un *sprint* como parte de la reunión de planificación
Implementación	• Crear los entregables. En este proceso el equipo Scrum trabaja en las tareas del *sprint backlog* para crear los entregables del *Sprint*. Se utiliza usualmente un *Scrum Board* (tablero de organización) para llevar un seguimiento del trabajo y actividades realizadas, así como también los inconvenientes que se hayan presentado. • Realizar la reunión de revisión. En esta reunión el equipo Scrum comparte los progresos realizados y los inconvenientes presentados que deben ser solucionados. Posteriormente se realiza la actualización del *Product Backlog* con cualquier cambio o modificación debatido en la reunión.
Revisión y Retrospectiva	• Demostrar y Validar el *sprint*. En este proceso el equipo Scrum presenta el entregable del *sprint* al propietario del producto y a las partes interesadas, con el fin de asegurar la aprobación por parte de los mismos.

2. METODOLOGÍA

El presente proyecto de titulación estará basado en la metodología SCRUM, la cual consta de cuatro fases [23].

La fase de inicialización permitirá crear la visión del proyecto, identificar los roles del equipo SCRUM y crear las historias de usuario en base a las encuestas realizadas, con las cuales se generarán los requerimientos del sistema.

A continuación, se tiene la fase de planeación, donde se establecerá la arquitectura del sistema. Además, se creará el *product backlog* y el *sprint backlog* los cuales ayudarán a tener un mayor orden en el proyecto y a priorizar tareas.

Seguidamente viene la fase de implementación. En donde en primera instancia se realizará el diseño de las bases de datos, y posteriormente se hará el diseño de las interfaces de usuario tanto de la aplicación de administración como de la aplicación móvil. Una vez terminado esto, se levantará el servidor en conjunto al servicio SOAP el cual será consumido por la aplicación de la administración.

Consecuentemente se levantará la aplicación de administración junto con el desarrollo de la lógica de todos sus módulos, que incluye el manejo de usuarios junto con sus huellas dactilares, la administración de los dispositivos biométricos con sus aulas, horarios, registros de asistencia; y finalmente se desarrollará la aplicación móvil.

En cuanto a Hardware se construye una maqueta en la cual se emula la instalación que tendría en una puerta, esto es con el lector de huellas biométricas, la cerradura electromagnética, el botón de salida y sus respectivas conexiones eléctricas y de red. Por último, se hará la instalación real en el aula 701 del edificio de Química / Eléctrica (Q/E).

En la fase de retrospectiva, el equipo Scrum en conjunto con el *Product Owner*, revisan los *sprints* y aprueban el avance. Además, se realizarán las pruebas pertinentes para demostrar la funcionalidad del proyecto implementado.

2.1 FASE DE INICIALIZACIÓN

2.1.1 VISIÓN DEL PROYECTO

Desarrollo de un prototipo para control de acceso a aulas y registro de asistencia del personal docente de la FIEE.

2.1.2 ROLES DE SCRUM

Los roles asignados a los participantes están organizados como se muestran en la Tabla 2.1.

Tabla 2.1. Roles de SCRUM

Rol	Nombre
Scrum Master	Pablo Hidalgo
Product Owner	Pablo Hidalgo
Development Team	Bryan Carrión, Doménica Gómez

2.1.3 ENCUESTA

Para poder generar historias de usuario, se creó una encuesta de 10 preguntas que se hicieron a 10 autoridades de la FIEE, cuyo modelo se encuentra en el ANEXO A. Esta encuesta se dividió en 2 partes que tratan el control de asistencia a clases, y el control de acceso a aulas.

2.1.3.1 Parte 1: Control de asistencia a clases

La pregunta 1 tiene como objetivo determinar si en la facultad se requiere un mayor orden en el registro de asistencia para el personal docente. Los resultados de esta pregunta se muestran en la Figura 2.1, en la cual se puede observar un absoluto acuerdo de los encuestados.

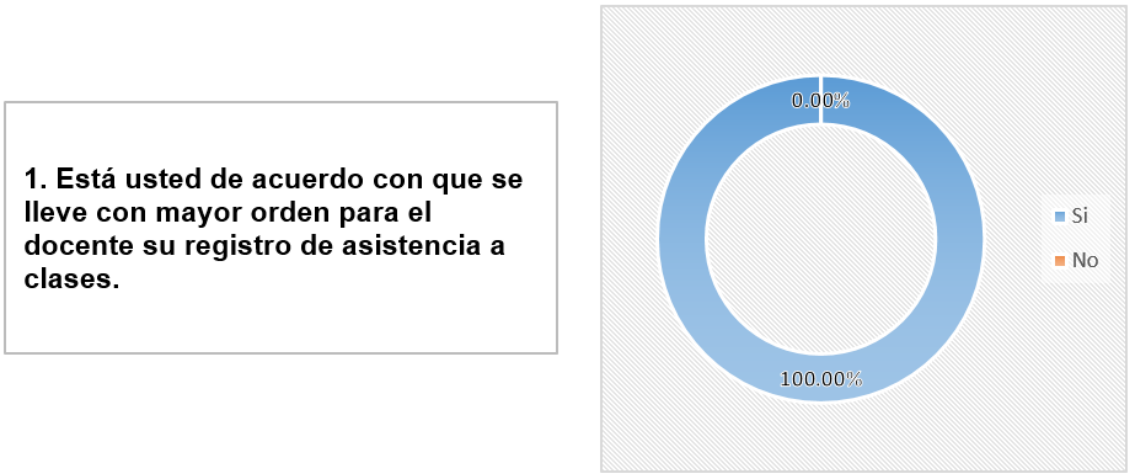


Figura 2.1. Resultados de la pregunta 1 de la parte 1 de la encuesta

La pregunta 2 permite definir si se necesita que el control se haga a través de un dispositivo electrónico, lo cual ayudaría a automatizar el proceso. Los resultados de esta pregunta se muestran en la Figura 2.2, en la cual se observa un acuerdo mayoritario al respecto.

2. Le gustaría que la contabilización de horas de asistencia a clases se haga de manera automática mediante un dispositivo electrónico.

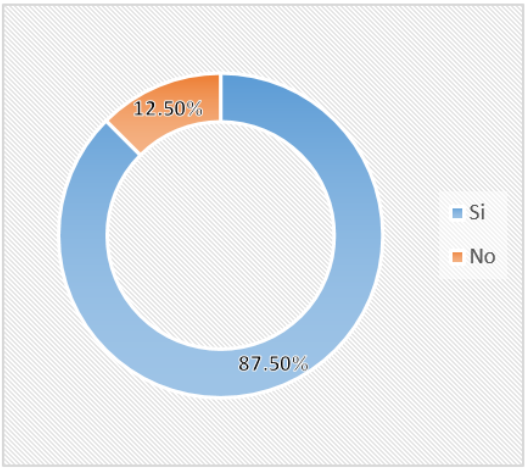


Figura 2.2. Resultados de la pregunta 2 de la parte 1 de la encuesta

La pregunta 3 permite determinar si la tecnología a usarse en el sistema, debe ser biométrica. Los resultados de esta pregunta se muestran en la Figura 2.3, en la cual se expone una ligera discrepancia, debido a que se sugirió el uso de otras tecnologías como RFID (*Radio Frequency Identification*).

3. Está usted de acuerdo con un sistema biométrico automatizado que lleve el registro de asistencia a clases.

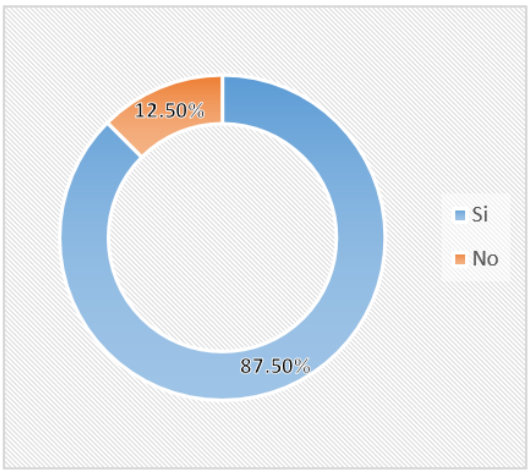


Figura 2.3. Resultados de la pregunta 3 de la parte 1 de la encuesta

La pregunta 4, tiene como finalidad determinar si se requiere que el sistema a desarrollar permita solicitar la recuperación de las horas de clase, o de hacer uso de algún aula por parte del docente. En los resultados a esta pregunta, mostrados en la Figura 2.4, se puede observar un acuerdo mayoritario entre los encuestados.

4. Está usted de acuerdo con que mediante este sistema el docente pueda solicitar permisos y planificar su recuperación.

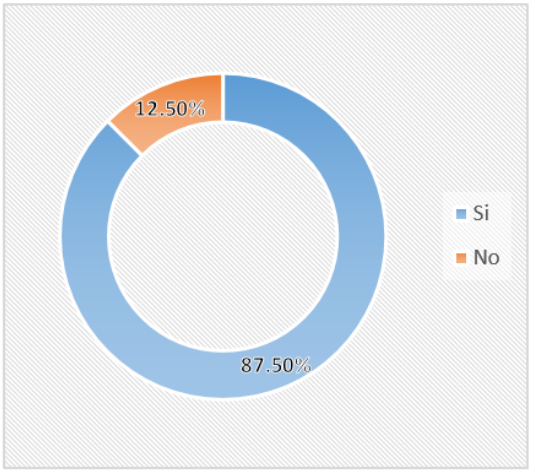


Figura 2.4. Resultados de la pregunta 4 de la parte 1 de la encuesta

La pregunta 5 tiene como propósito determinar si el usuario estaría de acuerdo con revisar su asistencia a través de la aplicación móvil a desarrollar. Lo resultados obtenidos que se muestran en la Figura 2.5, muestra en acuerdo total entre los encuestados.

5. Le gustaría poder revisar su registro de asistencia a clases mediante una aplicación móvil.

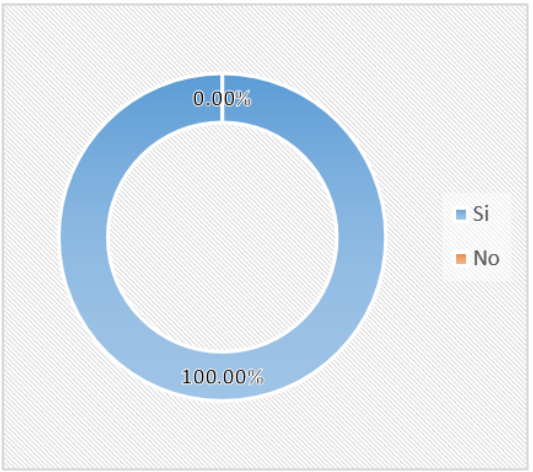


Figura 2.5. Resultados de la pregunta 5 de la parte 1 de la encuesta

2.1.3.2 Parte 2: Control de acceso a aulas

La pregunta 1 tiene como intención, averiguar si se debería tener un control de acceso a las aulas de la facultad. En los resultados mostrados en la Figura 2.6 se observa que un 87.50% de los encuestados está de acuerdo en que las aulas de la Facultad dispongan de un sistema de control de acceso.

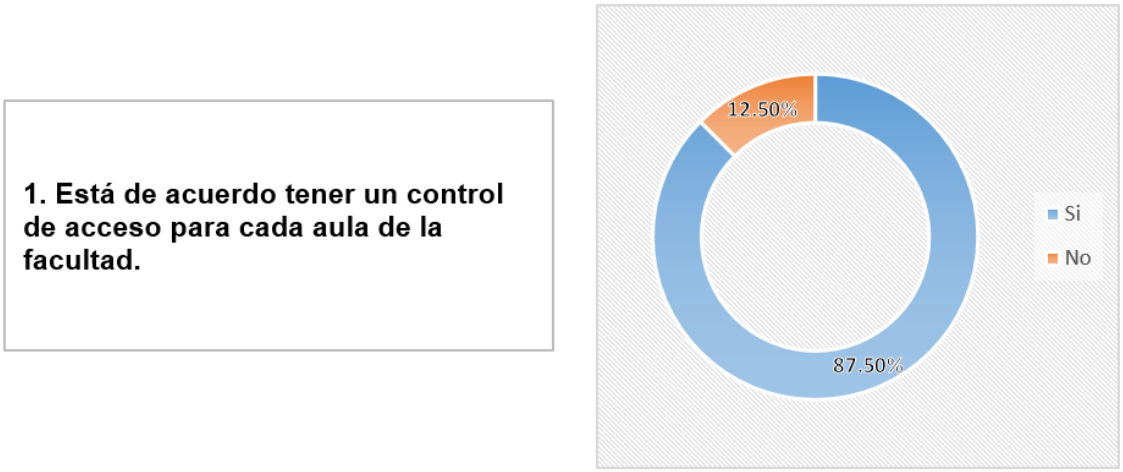


Figura 2.6. Resultados de la pregunta 1 de la parte 2 de la encuesta

La pregunta 2 tiene por objeto, determinar si el dispositivo de control de acceso, debería usar un lector biométrico de huellas dactilares. Los resultados de esta pregunta, mostrados en la Figura 2.7, señalan consonancia entre los encuestados.

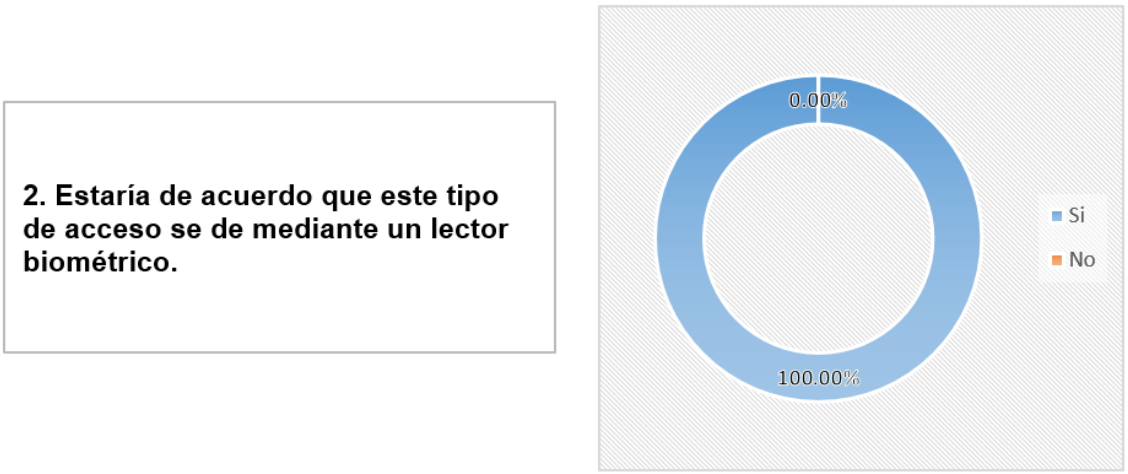


Figura 2.7. Resultados de la pregunta 2 de la parte 2 de la encuesta

La pregunta 3, pretende observar si se estaría de acuerdo con que el docente solo pueda acceder a un aula mientras se encuentre en su hora de clase. Esta pregunta generó discrepancia entre los encuestados, como se muestra en los resultados presentados en la Figura 2.8, ya que los docentes señalan que podrían requerir un aula, para tomar una prueba, o recuperar una clase no dada.

3. Le parece adecuado que el profesor pueda acceder al aula solo en la hora en la que le corresponde dar clase.

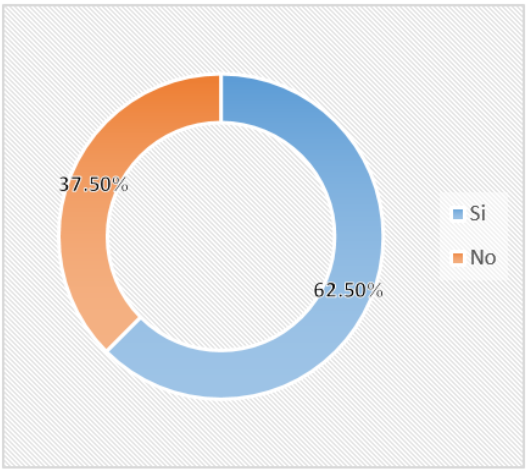


Figura 2.8. Resultados de la pregunta 3 de la parte 2 de la encuesta

La pregunta 4 tiene como objetivo evidenciar si el docente requeriría información sobre un aula en específico, como su capacidad u horarios. Los resultados mostrados en la Figura 2.9, indican que este requerimiento es una necesidad.

4. Está usted de acuerdo con que usted pueda obtener toda la información de un aula en específico como horas de disponibilidad, cupos, etc.

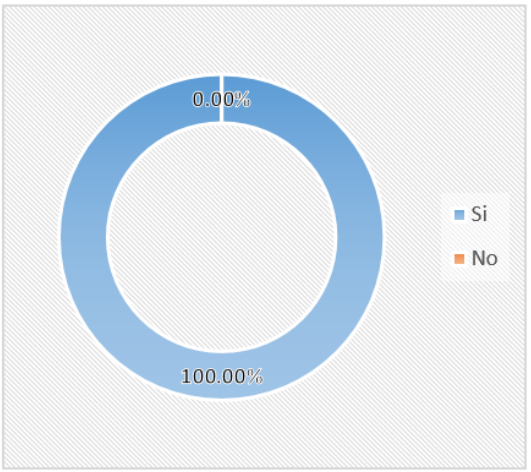


Figura 2.9. Resultados de la pregunta 4 de la parte 2 de la encuesta

La pregunta 5 tiene como propósito conocer si el docente le gustaría que se pueda reservar un aula de manera dinámica, es decir que, mediante los requerimientos del docente, el sistema le permita o no realizar dicha acción. En los resultados de esta pregunta mostrados en la Figura 2.10 se observa una respuesta mayoritariamente positiva.

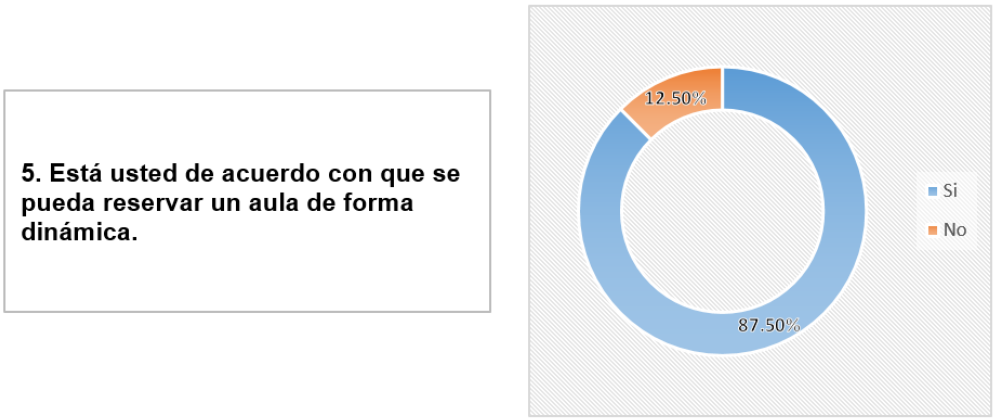


Figura 2.10. Resultados de la pregunta 5 de la parte 2 de la encuesta

2.1.4 HISTORIAS DE USUARIO

Se generan las historias de usuario en base a la información recabada en las encuestas del punto anterior, con el propósito de estructurar claramente los requerimientos de los usuarios. Las historias de usuario se muestran en la Tabla 2.2.

Tabla 2.2. Historias de usuario

ID	Preguntas de referencia	Titulo	Descripción
HU_01	Pregunta 1 de la primera parte. Pregunta 1 de la segunda parte.	Pre-registro de datos de docentes.	El usuario necesitará que sus datos se encuentren en el sistema, por ello, al tratarse de los docentes de la FIEE, se lo puede obtener de la base de datos local, previo a la autorización de las autoridades.
HU_02	Pregunta 2 y 3 de la primera y parte. Pregunta 2 de la segunda parte.	Registro de huellas dactilares.	Para que el usuario pueda hacer uso del registro de asistencia y el control de acceso a través de un lector biométrico, es necesario registrar las huellas dactilares de cada uno de los docentes que harán uso del sistema.
HU_03	Pregunta 4 de la primera parte. Pregunta 3, 5 de la segunda parte.	Validación de acceso.	El usuario requiere que no se limite el acceso a aulas solo en horas de clase.
HU_04	Pregunta 4 de la primera parte. Pregunta 5 de la segunda parte.	Reservar Aulas.	El usuario desea que se puedan realizar reservas de aulas, solicitando así el permiso de algún aula en específico.
HU_05	Pregunta 5 de la primera parte. Pregunta 4 de la segunda parte.	Muestra de información al usuario.	El usuario deberá poder ver la información completa sobre las aulas y sus horarios, además, deberá poder ver también sus registros de asistencia.

2.1.5 REQUERIMIENTOS

Los requerimientos del proyecto se han identificado basándose en la visión del mismo, y en las historias de usuario generadas en el punto anterior.

2.1.5.1 Requerimientos funcionales

- Habilitar acceso en base a huella del usuario y al horario del aula.
- Permitir la actualización de la base de datos semestralmente.
- Registrar y actualizar datos del usuario.
- Permitir la realización de cambios de horarios de aulas.
- Permitir la reserva de aulas en un horario específico.
- Agregar y controlar terminales biométricos.
- Generar reportes de asistencia.
- Generar reportes de utilización de aulas.
- Mostrar información de las aulas al usuario.
- Mostrar registros de asistencia al usuario.
- Permitir al usuario solicitar la reserva de un aula.

2.1.5.2 Requerimientos no funcionales

- Utilizar la arquitectura Cliente-Servidor.
- Usar terminales biométricos para el control de acceso y registro de asistencia.
- Utilizar lenguaje de programación C# en el servidor privado para el control de los dispositivos biométricos.
- Utilizar el servicio SOAP para la administración del sistema.
- Emplear el lenguaje de programación C# para crear la aplicación de escritorio que consuma el servicio.
- Utilizar el framework React Native para crear la aplicación móvil Android.
- Alojar los datos en *SQL Server* y Firebase según corresponda.

2.2 FASE DE PLANEACIÓN

Tomando en cuenta la arquitectura seleccionada (Cliente-Servidor) a continuación, se describe el sistema propuesto, en el que la estimación de tiempo para cada uno de los *sprints* se presenta en la Tabla 2.4.

El prototipo cuenta con: Una aplicación Android en el lado del cliente. En el lado del servidor están el hardware del servidor en sí, más las correspondientes conexiones de red entre éste y los terminales biométricos, los que a su vez se encuentran conectados eléctricamente a una cerradura electromagnética y un botón de salida.

Además, en el servidor se instala el servicio SOAP a consumirse por la aplicación de administración. La aplicación móvil se comunica con el servidor a través del BaaS (*Backend as a Service*) de Firebase, con el que el usuario se autentica y obtiene información con la API usando Internet.

La comunicación entre los terminales biométricos y el servidor se da de la siguiente manera: Primero, el usuario coloca su dedo en el lector de huellas dactilares del terminal biométrico; seguidamente, si la huella es reconocida por el dispositivo, se envía la señal al servidor; éste comprueba que corresponda al profesor en la hora indicada y habilita el acceso al aula en la que se encuentre el terminal.

Es necesario señalar que, los registros se generan una vez que ha sido validado el acceso, y se guardan tanto en la base de datos local del servidor en *SQL server*, como en la *Realtime Database* de Firebase.

2.2.1 CASOS DE USO

En esta sección se presentarán los distintos casos de uso, mismos que son útiles para representar la interacción de los actores con el sistema.

Los actores del prototipo son:

- Usuario no registrado.
- Usuario Pre-registrado.
- Usuario registrado.
- Usuario Administrador.

En base a los requerimientos se han generado diagramas de casos de uso, los cuales muestran la interacción que los actores tienen con el sistema.

El usuario registrado interactúa con la aplicación móvil, en la cual puede ingresar a la aplicación iniciando sesión, revisar sus solicitudes de reserva, los horarios de aulas, registros de asistencia, y realizar solicitudes de reserva. Esto se ilustra en el diagrama de la Figura 2.11.

El usuario administrador interactúa con la aplicación de escritorio que consume el servicio web, ésta le permitirá: Agregar o actualizar usuarios, agregar y manejar terminales biométricos revisar horarios y modificarlos de manera permanente, realizar reservas de aulas, atender solicitudes, generar reportes de asistencia o inasistencia de profesores, generar reportes de utilización de aulas y actualizar la base de datos.

Para el caso de los usuarios no registrados y pre-registrados, éstos actuarán en los casos de “Agregar usuario” y “Actualizar usuario” del Administrador respectivamente, ya que el

registro de huella se hace de manera presencial a través del lector biométrico. Esto se ilustra en la Figura 2.12.

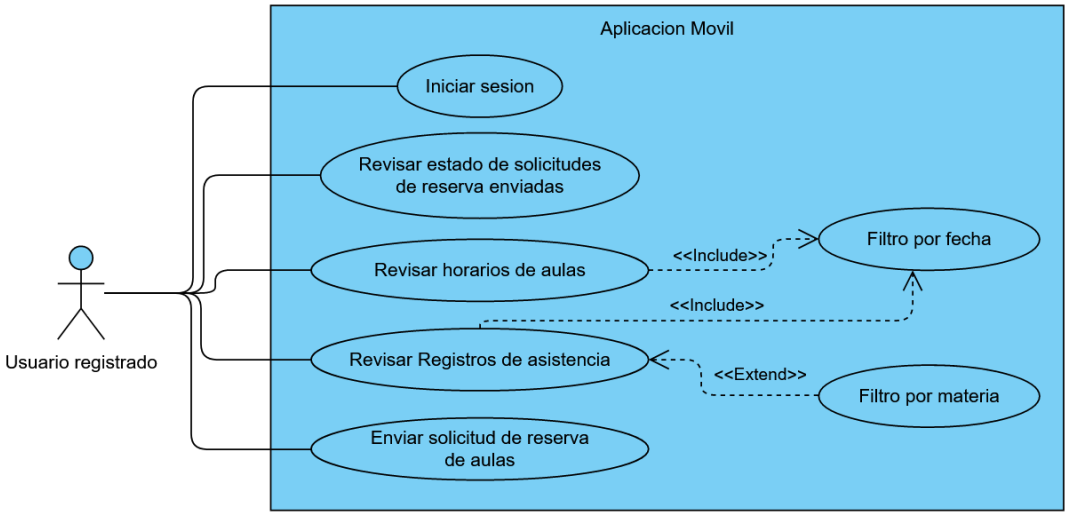


Figura 2.11. Casos de Uso: Usuario registrado

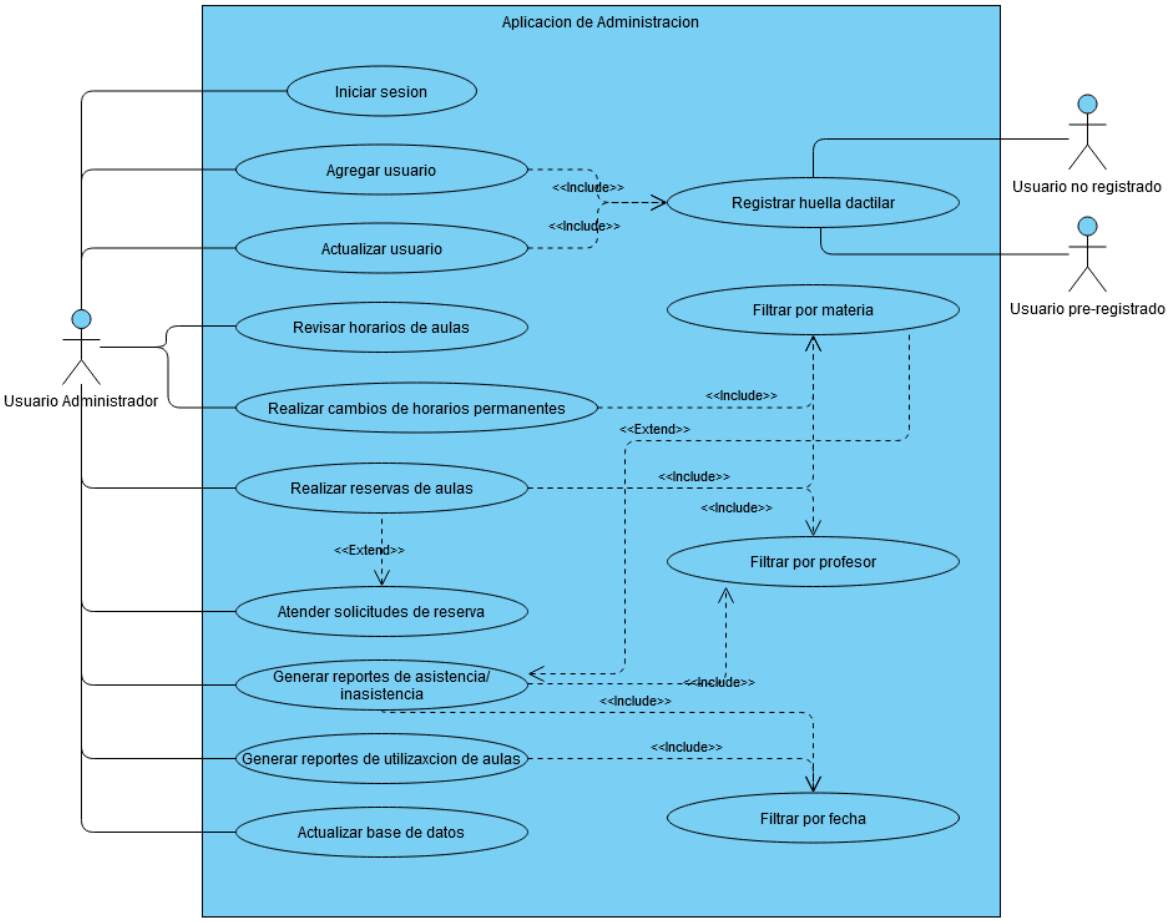


Figura 2.12. Casos de uso: Usuarios Administrador, no registrado y pre-registrado

2.2.2 PRODUCT BACKLOG

El *Product backlog* (Tabla 2.3.) permite identificar las tareas a realizar para cumplir con los requerimientos, lo que permite tener un enfoque general de todo lo que se espera llevar a cabo.

Tabla 2.3. Product Backlog (parte 1 de 3)

Requerimientos	Tareas
1.- Adquirir terminales biométricos para el control de acceso y registro de asistencia.	Análisis y selección del fabricante de los dispositivos a utilizar, en conjunto a sus respectivos SDK.
2.- Crear las bases de datos.	Modelar las base de datos.
	Codificar las base de datos.
	Alojar las bases de datos en Firebase y *SQL Server*
3.- Montar el servidor.	Definir las clases y librerías a usar.
	Levantar el servicio.
	Codificar algoritmos internos de control y comunicación con los terminales biométricos.
	Habilitar en el *Firewall* los puertos necesarios.
4.- Diseñar interfaces gráficas	Diseñar interfaces para la aplicación de administración
	Diseñar interfaces para la aplicación móvil
	Codificar la interfaz de aplicación de administración
5.- Permitir al usuario administrador ingresar a la aplicación.	Validar las credenciales de ingreso a la aplicación
6.- Permitir al usuario administrador actualizar la base de datos con un archivo de Microsoft Excel.	Definir el formato correcto en el cual debe estar el libro de Microsoft Excel para evitar errores de lectura.
	Leer los datos y recuperarlos correctamente como objetos.
	Pre-registrar usuarios nuevos en la base de datos.
	Almacenar semestralmente los horarios en la base de datos.
	Codificar la lógica y las validaciones del submódulo correspondiente.

Tabla 2.3. Product Backlog (parte 2 de 3)

Requerimientos	Tareas
7.- Permitir al usuario administrador agregar y actualizar usuarios.	Registrar datos personales como nombres, apellidos, cédula, correo electrónico, contraseña.
	Registrar huella dactilar.
	Guardar la foto de perfil de usuario.
	Crear usuario en Firebase para autenticación en la aplicación móvil.
	Codificar la lógica y las validaciones del submódulo correspondiente.
8.- Permitir al usuario administrador agregar terminales biométricos.	Conectar el sistema al terminal a través de TCP/IP.
	Asignar un aula al dispositivo.
	Codificar la lógica y las validaciones del submódulo correspondiente.
9.- Permitir al usuario administrador el control de terminales biométricos.	Definir las funciones a controlar de manera remota.
	Codificar la lógica y las validaciones del submódulo correspondiente.
10.- Permitir al usuario administrador modificar horarios de manera permanente.	Recuperar horarios en base a materias y grupos.
	Codificar la lógica y las validaciones del submódulo correspondiente.
11.- Permitir al usuario administrador reservar aulas.	Recuperar horarios en base al profesor, materias y grupos.
	Codificar la lógica y las validaciones del submódulo correspondiente.
12.- Permitir al usuario administrador atender solicitudes de reserva.	Recuperar solicitudes de la *Realtime Database* de Firebase.
	Codificar la lógica y las validaciones del submódulo correspondiente.
13.- Permitir al usuario administrador generar reportes de asistencia e inasistencia.	Recuperar registros obtenidos de los terminales biométricos para generar los reportes.
	Calcular datos estadísticos respectivos.
	Generar reportes en archivos tipo PDF y XLSX.
14.- Permitir al usuario administrador generar reportes de utilización de aulas.	Recuperar registros obtenidos de los terminales biométricos para generar los reportes.
	Calcular datos estadísticos respectivos.
	Generar reportes en archivos tipo PDF y XLSX.

Tabla 2.3. Product Backlog (parte 3 de 3)

Requerimientos	Tareas
15.- Permitir al usuario acceder a la aplicación móvil.	Crear usuario en *Firebase Authentication* para que pueda acceder con su correo electrónico y contraseña.
16.- Permitir al usuario revisar el estado de sus solicitudes de reserva.	Mostrar una lista en pantalla de sus solicitudes y del estado de las mismas.
17.- Permitir al usuario revisar los horarios de aulas.	Mostrar los horarios del aula seleccionada por el usuario.
18.- Permitir al usuario revisar sus registros de asistencia.	Mostrar los registros de asistencia del usuario en base a varios filtros.
19.- Permitir al usuario hacer solicitudes de reserva de aulas.	Mostrar un formulario en el que el usuario pueda enviar una solicitud de reserva de un aula al sistema.
20.- Implementación física en escenario simulado	Construcción de maqueta compuesta por una puerta y pared de madera
	Instalación de equipos y cableado en maqueta
21.- Implementación física en escenario real	Instalación de equipos y cableado en escenario real

2.2.3 SPRINT BACKLOG

El *Sprint Backlog* permite una mejor planeación y manejo de las tareas a realizar para cumplir los requerimientos definidos en el *Product Backlog*. Los *sprints* se muestran en la Tabla 2.4.

Tabla 2.4. Sprint Backlog

Sprint	Requerimiento	Tiempo Estimado	Sprint	Requerimiento	Tiempo Estimado
SPRINT 1	1	1 semana	SPRINT 9	11	1 semana
SPRINT 2	2,3	1 semana	SPRINT 10	13	2 semanas
SPRINT 3	4	1 semana	SPRINT 11	14	1 semana
SPRINT 4	5	1 semana	SPRINT 12	4	2 semanas
SPRINT 5	7	1 semana	SPRINT 13	15	1 semana
SPRINT 6	6	1 semana	SPRINT 14	16,17,18	1 semana
SPRINT 7	8,9	2 semanas	SPRINT 15	19	1 semana
SPRINT 8	10, 17	2 semanas	SPRINT 16	20,21	1 semana

2.2.3.1 Tablero SCRUM

Para monitorear el avance de las tareas, se usa el tablero de control SCRUM de *QuickScrum.* En esta plataforma se puede añadir todo lo relacionado al *Product Backlog*, crear los *sprints* y realizar el seguimiento a las tareas dentro de éstos. Este tablero permite llevar un mayor orden en el desarrollo del proyecto, aumentando así la productividad.

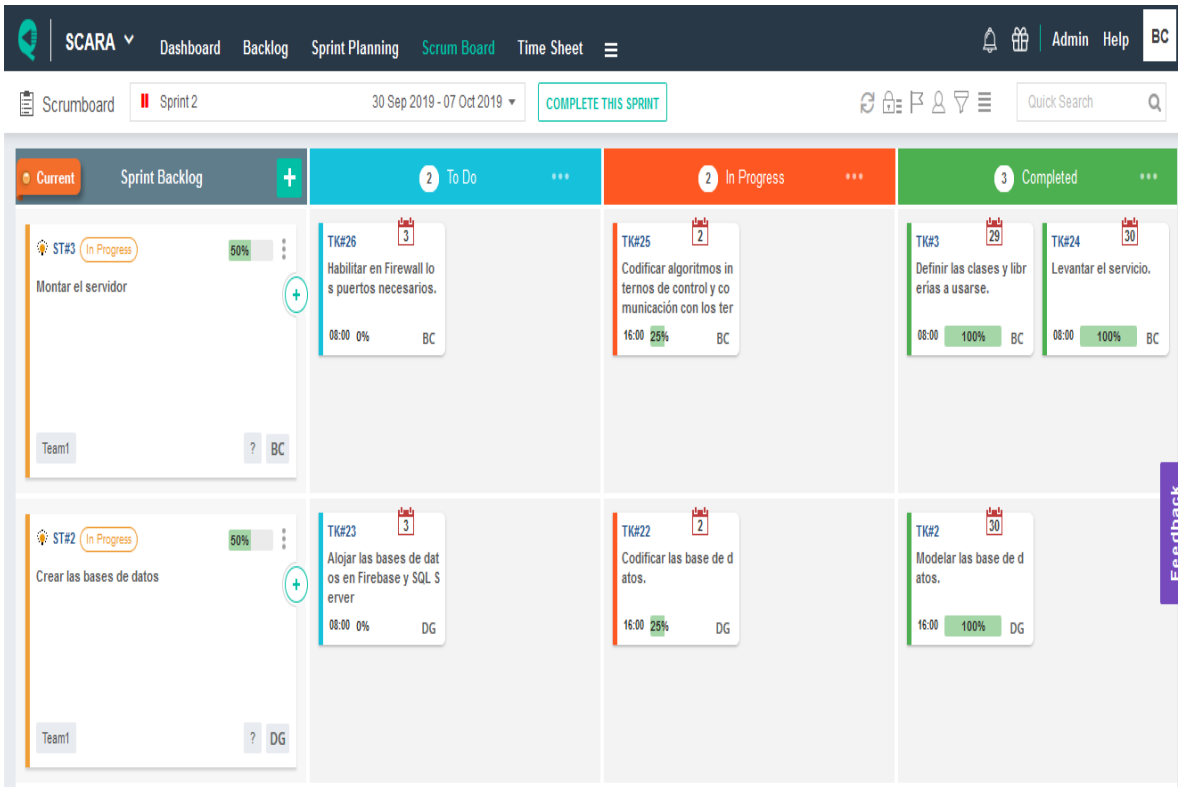


Figura 2.13. Tablero Scrum de QuickScrum

2.2.4 CONVENCIONES DE CÓDIGO

2.2.4.1 C Sharp

2.2.4.1.1 Convenciones de nombres

La consistencia es la clave para el código escalable, lo cual se traduce en seguir un estándar al nombrar: Proyectos, archivos de origen e identificadores; se incluyen campos, variables, propiedades, métodos, parámetros, clases, interfaces y espacios de nombres.

Un resumen de las especificaciones de las convenciones de nombres para C# se muestra en la Tabla 2.5, en la cual los símbolos tienen el siguiente significado: “P” = `PascalCase`, “c” = `camelCase`, “X” = no aplica, y “_” = prefijo con guión bajo.

Tabla 2.5. Convenciones de nombres [24]

Identificador	Pública	Protegida	interna	Privada
Archivo de proyecto	P	X	X	X
Archivo de recurso	P	X	X	X
Otros archivos	P	X	X	X
Espacio de nombres	P	X	X	X
Clase o estructura	P	P	P	P
Interfaz	P	P	P	P
Método	P	P	P	P
Propiedad	P	P	P	P
Campo	P	P	P	_c
Constante	P	P	P	_c
Campo estático	P	P	P	_c
Delegado	P	P	P	P
Evento	P	P	P	P
Variable interna	X	X	X	C
Parámetro	X	X	X	C

2.2.4.1.2 Convenciones de código

La convenciones en el formato del código, van de la mano con las de nombre para crear un código consistente y sostenible, aunque esto suele diferir de programador en programador; hay ciertos puntos que se respetan, como los mostrados en la Tabla 2.6.

Tabla 2.6. Convenciones de código

Código	Estilo
Archivos	Un espacio de nombres y una clase por archivo.
Llaves	En una nueva línea, siempre usarlos cuando sea opcional.
Tabulación	Usar tabulación de tamaño 4.
Comentarios	Usar // o /// pero no /*…*/.
Variables	Una variable por declaración.

2.2.4.1.3 Método de acceso a datos

Para poder generar un servicio para que el cliente pueda acceder a la información almacenada en la base de datos, se hará uso de ADO.NET, que es un conjunto de clases disponibles para el acceso a datos de múltiples plataformas.

Además, para la sección NOSQL, se hará uso de la librería `Firesharp`, que es una librería no oficial que permite la generación de peticiones HTTP a Firebase; adicionalmente esta librería permite la transformación de los objetos en Formato JSON, y viceversa.

2.2.4.2 React Native

2.2.4.2.1 Estructura de directorios

Una estructura consistente en un proyecto es necesaria para que el mismo sea entendible para cualquier persona, lo cual además le da una mayor escalabilidad. Por ello la estructura que se definió para la aplicación móvil se puede observar en la Figura 2.14.

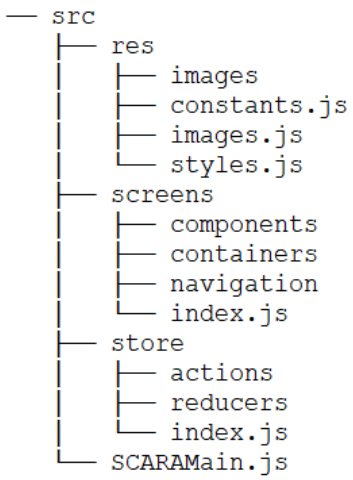


Figura 2.14. Estructura de directorios

La estructura tiene como directorio raíz la carpeta `src`, en la cual se tiene el archivo principal de la aplicación, además de 3 carpetas. En la carpeta `res` se colocarán los recursos a usarse, como imágenes, estilos y valores constantes, como mensajes de error, rótulos de tablas, etc.

La carpeta `screens` contiene todo lo relacionado a la parte gráfica de la aplicación. La capeta `components`, abarca todos los componentes que se han creado, como tablas, botones, etc. Dentro de la carpeta `containers`, irán las pantallas a mostrarse, las cuales son la combinación de distintos componentes. En la carpeta `navigation` se colocan los archivos correspondientes a la navegación de la aplicación.

Finalmente en la carpeta `store`, se ubican los archivos correspondientes a todo lo relacionado a Redux, como los accionadores, reductores y el archivo de index que contiene a la *store* de React y los *middleware* necesarios.

2.2.4.3 Respaldo y manejo de versiones

Con el propósito de respaldar la información y realizar el control de cambios, se levantaron repositorios privados para cada proyecto, esto es, para la Aplicación Móvil, para la Aplicación de Administración, y para el servidor en GitHub. En la Figura 2.15 se muestra un ejemplo de uno de los repositorios que se generaron, en este caso, del cliente de administración del sistema.

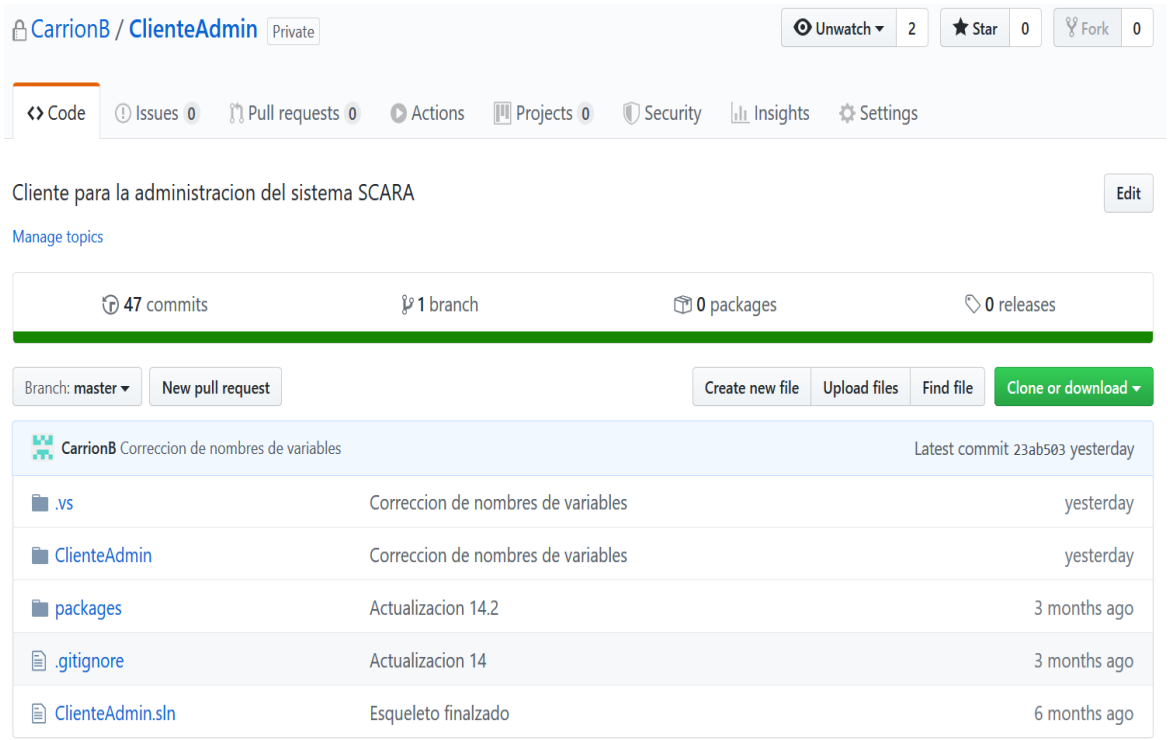


Figura 2.15. Repositorio levantado para la Aplicación de Administración

Para levantar un repositorio en GitHub, es necesario en primer lugar, a través de la ventana de comandos, situarse en la carpeta en la que se encuentra el proyecto con el comando `cd`, seguidamente se digitan los comandos mostrados en la Figura 2.16, que fueron los utilizados para generar el repositorio en la Figura 2.15.

```
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/CarrionB/ClienteAdmin.git
git push -u origin master
```

Figura 2.16. Lista de comandos para levantar el repositorio ClienteAdmin

2.3 FASE DE IMPLEMENTACIÓN

2.3.1 SPRINT 1

Hoy en día existe un sinnúmero de opciones en lo que respecta a lectores biométricos de huella dactilar en el área comercial, es por esto que se requiere un estudio de mercado, mismo que se realizará en el presente *sprint*.

A continuación, se presentarán algunas opciones viables para este proyecto, sus principales características técnicas y costos. Posteriormente se realizará un análisis comparativo para escoger la mejor opción.

2.3.1.1 ESCÁNER BIOMÉTRICO

Mediante este dispositivo, se recuperarán las huellas dactilares del personal docente, las cuales serán almacenadas en la base de datos para las futuras comparaciones.

1. **Marca: SecuGen – Modelo: Hamster Plus**

Tabla 2.7. Características escáner biométrico SecuGen Hamster Plus [25]

Resolución de imagen	500 DPI
Área de detección	13.2 mm x 15.2 mm
Dimensiones	53 x 73 x 84 mm (con soporte). 27 x 40 x 73 mm (sin soporte)
Peso	165 g (con soporte). 100 g (sin soporte)
Temperatura de funcionamiento	-20°C hasta 65 °C
Humedad de funcionamiento	0 - 90% no condensada
Interfaz	USB 1.1 de alta velocidad. USB 2.0 de alta velocidad
Tensión de alimentación	5 V (a través de USB)
Corriente de alimentación	140 mA (máx.)
Imagen escala de grises	256 niveles (8 bits)

Figura 2.17. Escáner biométrico SecuGen Hamster Plus [25]

2. **Marca: Digital Persona – Modelo: U.are.u 4500**

Tabla 2.8. Características escáner biométrico digital persona U.are.u 4500 [26]

Resolución de imagen	512 dpi (promedio x,y sobre el área de escaneo)
Área de detección	14.6 x 18.1 mm
Dimensiones	65 x 36 15.56 mm
Peso	105 g
Temperatura de funcionamiento	0 - 40°C
Humedad de funcionamiento	20 - 80% no condensada
Interfaz	USB 2.0 Alta velocidad
Tensión de alimentación	5.0 V +5%
Corriente de alimentación	< 110 mA (típico)
Imagen escala de Grises	256 niveles

Figura 2.18. Escáner biométrico digital persona U.are.u 4500 [26]

3. **Marca: ZKTeco – Modelo: ZK9500**

Tabla 2.9. Características escáner biométrico ZKTeco ZK9500 [27]

Resolución de imagen	500 dpi
Área de Detección	16.5 x 23 mm
Dimensiones	75.5 x 53.2 x 19 mm
Temperatura de funcionamiento	0 - 40°C
Humedad de funcionamiento	0 - 90% no condensada
Interfaz	USB 2.0 / USB 1.1
Tensión de alimentación	5 V, 200 mA escaneo; 5 V, 60 mA en modo espera
Corriente de alimentación	200 Ma
Imagen escala de Grises	256 niveles

Figura 2.19. Escáner biométrico ZKTeco ZK9500 [27]

Para elegir la mejor opción que responda a todos los requerimientos del presente proyecto, se presenta en la Tabla 2.10 los aspectos más relevantes a tomar en cuenta.

Tabla 2.10. Comparación entre escáneres biométricos [28], [29], [30], [31]

Variables / Dispositivo	Precio (USD)	Capacidad Funcional Necesaria	SDK Disponibles	Identificación (FAR y FRR)
SecuGen – Hamster Plus	$101,99	Si	Java	FAR: 0,001% FRR: 0,1%
Digital Persona - U.are.U 4500	$103,99	Si	Visual Basic Visual C# PHP	FAR: 0,001% FRR: 0,1%
ZKTeco- Zk 9500	$65,00	Si	Java Android Visual C#	FAR: <=0,0001% FRR: <=0.01

En base al análisis realizado, se ha determinado que la mejor opción para este caso es el dispositivo Marca:ZKTeco Modelo: ZK9500, puesto que además de ofrecer una excelente relación calidad-precio, presenta mejores características de hardware, herramientas de software útiles para el desarrollo de aplicaciones, tasa de error muy baja, y un alto nivel de exactitud.

Por tanto, este dispositivo podrá responder adecuadamente a los requerimientos del presente proyecto.

2.3.1.2 TERMINAL BIOMÉTRICO

Mediante este dispositivo, se permitirá o denegará el acceso al aula bajo la restricción que la huella dactilar ingresada pertenezca al docente al que le corresponde dictar clases en esa aula al momento del registro.

1. **Marca: Hikvision Modelo: Ds-k1t8003ef**

Tabla 2.11. Características terminal biométrico Hikvision Ds-k1t8003ef [32]

Pantalla LCD	2.4" (320x240 LCD-TFD pantalla display)
Capacidad de huellas	100
Capacidad de tarjetas	1.000
Capacidad de eventos	100.000
Modo de comunicación	TCP/IP 10/100Mbps, auto adaptativo, Ehome
Fuente de poder	12 Vdc / 1 A
Temperatura de funcionamiento	-10°C a 55°C
Humedad de funcionamiento	10% a 90% (no condensada)
Dimensiones	140mm x 155mm x 30mm

Figura 2.20. Terminal biométrico Hikvision Ds-k1t8003ef [32]

2. **Marca: ZKTeco - Modelo: K50**

Tabla 2.12. Características terminal biométrico ZKTeco K50 [33]

Pantalla	TFT 2.8"
Capacidad de huellas	800
Capacidad de tarjetas	100
Capacidad de eventos	50.000
Comunicación	TCP/IP USB-Host
Fuente de poder	5 Vdc / 1 A
Temperatura de funcionamiento	0°C a 45°C
Humedad de funcionamiento	20% a 80% (no condensada)
Dimensiones	185 x 140 x 30 mm

Figura 2.21. Terminal biométrico ZKTeco K50 [33]

3. **Marca: ZKTeco – Modelo: K20**

Figura 2.22. Terminal biométrico ZKTeco K20 [34]

Tabla 2.13. Características terminal biométrico ZKTeco 20 [34]

Pantalla	TFT de 2.8"
Capacidad de huellas	500
Capacidad de tarjetas	500
Capacidad de eventos	50
Comunicación	TCP/IP USB-Host
Fuente de poder	5 Vdc / 1 A
Temperatura de funcionamiento	0°C a 45°C
Humedad de funcionamiento	20% a 80%
Dimensiones	185 x 140 x 30 mm

De igual forma para la selección de la mejor opción a utilizar en el presente proyecto, en la Tabla 2.14. se resumen los aspectos más relevantes a tomar en cuenta.

Tabla 2.14. Comparación entre terminales biométricos [32], [35]

Variables / **Dispositivo**	**Precio (USD)**	**Capacidad Funcional Necesaria**	**SDK Disponibles**	**Identificación (FAR y FRR)**
Hikvision Ds-k1t8003ef	$76,00	Si	C++	FAR: 0,001% FRR: 0,1%
ZKTeco- K50	$100,00	Si	Visual Basic, Visual C#, Java	FAR: <=0,0001% FRR: <=1%
ZKTeco- K20	$100,00	Si	Visual Basic, Visual C#, Java	FAR: <=0,0001% FRR: <=1%

Acorde al estudio efectuado, se ha determinado que la opción más adecuada, en el caso del terminal es el lector biométrico de huella dactilar Marca: ZKTeco Modelo K50, puesto que ofrece las herramientas de software necesarias para llevar a cabo el desarrollo de la aplicación de administración, mayor capacidad de almacenamiento, alto nivel de exactitud, baja tasa de error, y un costo coherente teniendo en cuenta sus múltiples prestaciones.

2.3.2 SPRINT 2

En este *sprint* se generarán las bases de datos para proceder a levantar el servidor y el servicio.

2.3.2.1 Diseño

Para poder levantar el servidor y el servicio es necesario en primer lugar, definir las clases base para el proyecto. Se crea una clase `Usuario`, la cual tiene los siguientes atributos:

- Cédula: Funcionará como identificativo único del usuario.
- Apellidos y Nombres.
- Contraseña: Servirá para ingresar tanto a la Aplicación de Administración en caso de habilitársele el permiso al mismo, y a la Aplicación Móvil.
- E-mail: Servirá como nombre de usuario para el ingreso a la Aplicación Móvil.
- Imagen de perfil.
- Huella: Se necesita registrar la huella del usuario para poder ingresarla en los terminales biométricos.
- Parámetros que permitan identificar si el usuario es administrador, y/o profesor; adicionalmente, otro para habilitarlo o deshabilitarlo.

También se necesita definir clases para el manejo de los horarios de clase. Primero se definió una clase denominada `Ranura de tiempo`, la cual consiste en un periodo de tiempo entre la hora inicio y la hora fin de clase, de un horario perteneciente a un grupo en un día en específico.

Para definir un horario de clase se necesita también la asignatura que se impartirá, por lo que se añadió la clase `Materia` con ese objetivo. Con estas clases ya determinadas, se forma la clase `Grupo`, la cual cuenta con los siguientes atributos:

- Un usuario que debe ser profesor.
- La materia correspondiente.
- El número de cupos.
- Un parámetro que permita identificar si es, o no, un grupo de recuperación.
- Los datos del semestre al que pertenece, como el nombre del mismo y la fecha de inicio y fin.

Para completar el horario de clase se necesitan varios grupos, los cuales se encontrarán en la clase `Aula`, junto a la capacidad de la misma.

A la vez, se añade la clase que contendrá a la clase `Aula`, la cual será `Dispositivo`, que tiene además los atributos del terminal biométrico como dirección IP, dirección MAC y demás datos dados por el fabricante. En base a esto se define que, al hablar de un Dispositivo a lo largo de este documento, se entenderá que se trata de un terminal biométrico.

Es necesario señalar que el cliente de administración hará uso del servicio y del servidor para comunicarse con los terminales biométricos, por lo que en este nivel se crea la clase `DispositivoServidor`, la cual heredará de la clase `Dispositivo` y además contiene

un objeto de tipo `CZKEMClass` el mismo que contiene las funciones de control, de los terminales.

En cuanto a los registros de asistencia, también se crea una clase llamada `Registro`, la cual contendrá los datos del profesor al que le corresponde el registro, del aula en la que se registró su ingreso, y el grupo y la materia correspondientes. Igualmente contará con la fecha y hora en la que se dio el registro.

Las relaciones entre estas clases se muestran en el diagrama de clases que se encuentra en la Figura 2.23.

En base al diagrama de clases, se construye el diagrama entidad-relación que ayuda a construir la base de datos en SQL. En este diagrama, además de las entidades basadas en las clases ya enunciadas en el diagrama anterior, se añadieron las entidades `AulaGrupo` y `Horarios`.

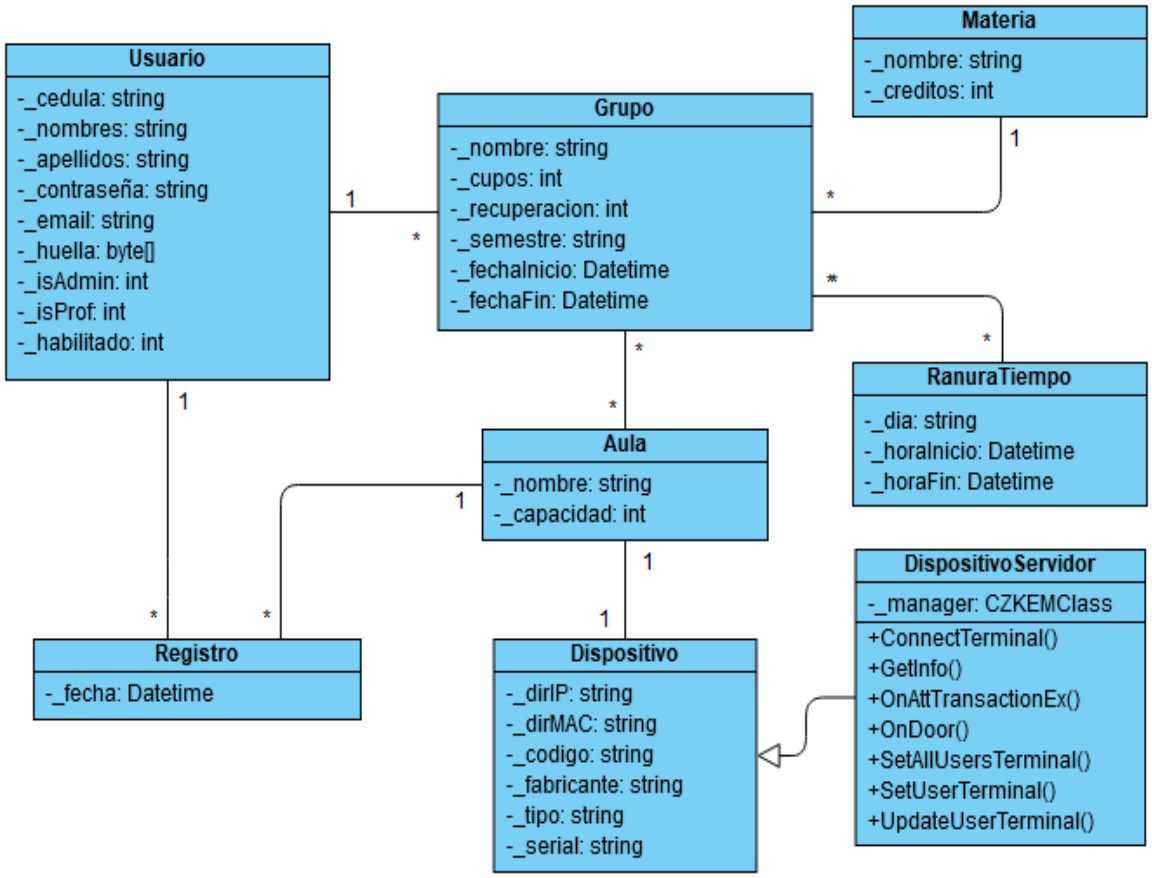


Figura 2.23. Diagrama de clases

La entidad `AulaGrupo`, representa la relación de muchos a muchos entre las entidades `Aulas` y `Grupos`; y la entidad `Horarios`, representa la relación de muchos a muchos entre la entidad `AulaGrupo` y Ranuras de Tiempo (`RTs`). El diagrama se muestra en la Figura 2.24.

Finalmente se diseñaron los esquemas para las bases de datos de Firebase. Para el caso de la Aulas, grupos, materias y profesores, se mantiene cierta similitud con su contraparte relacional como se puede observar en la Figura 2.25.

Cabe señalar que las estructuras de registros se asemejan a la de los anteriores diagramas, con la diferencia que se agrupan por la cédula del ejecutor del mismo, y además no se colocan identificadores, sino solo los nombres de los atributos (Figura 2.26).

Para el caso de las solicitudes, se le asigna una llave en base a la cédula del solicitante, la fecha en la que se requiere, y la materia; con lo cual se facilita la búsqueda y recuperación de datos.

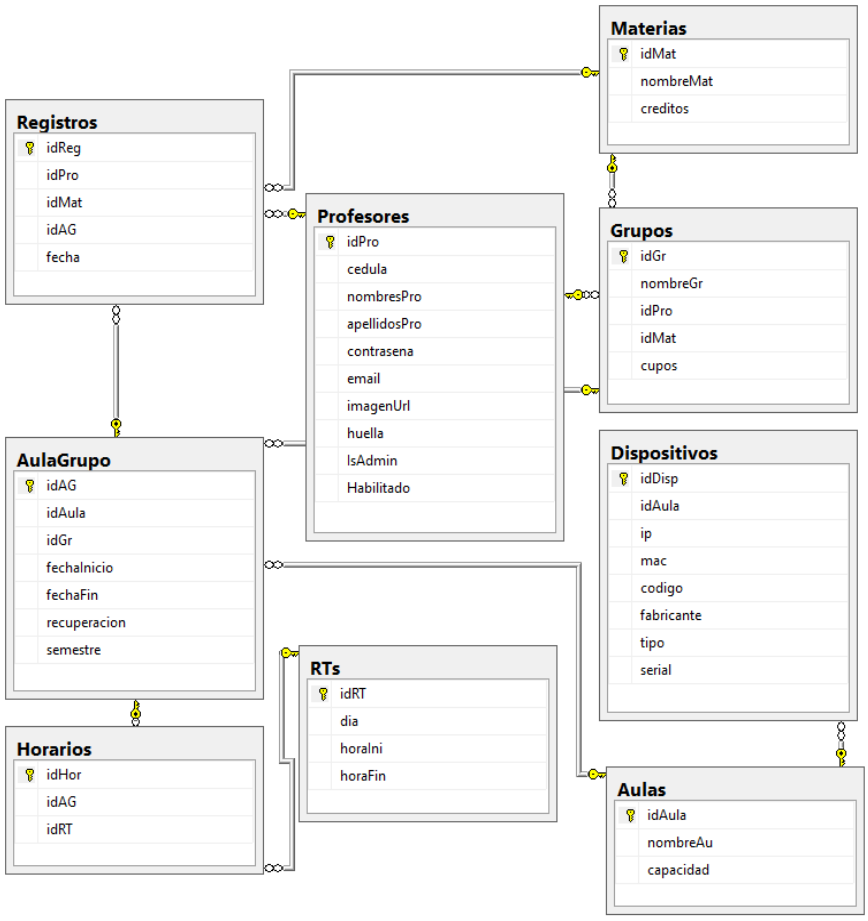


Figura 2.24. Diagrama Entidad-Relación

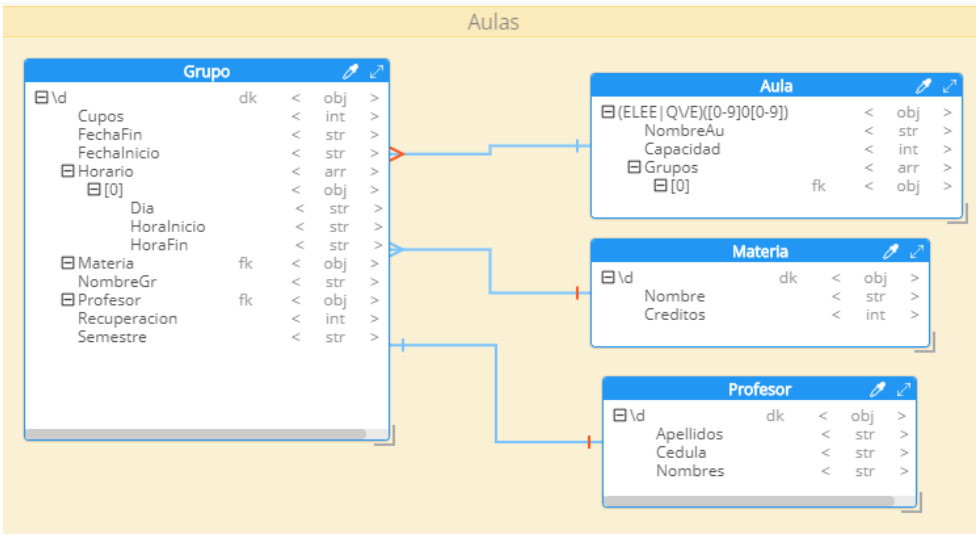


Figura 2.25. Esquema de base de datos scara-db/Aulas

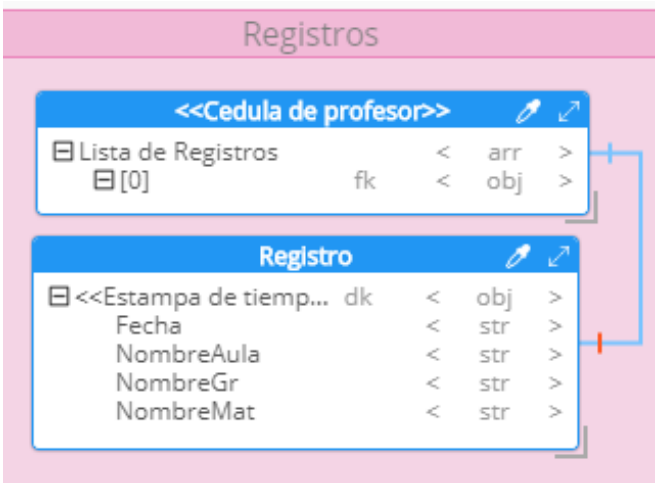


Figura 2.26. Esquema de base de datos scara-db/Registros

Estas solicitudes contarán con los siguientes atributos:

- `Aprobacion`: Es un parámetro que tendrá tres estados: Pendiente, aprobado y rechazado.
- `Cedula`: Permite la identificación del solicitante.
- `Error`: En caso de la solicitud ser rechazada, este parámetro informará al docente la razón del rechazo.
- `FechaReserva`: Fecha en la que se desea hacer la reserva.
- `HoraFin y HoraInicio`: Periodo de tiempo solicitado para la reserva.
- `NombreAu, NombreGr, NombreMat`: Parámetros que permitirán identificar al aula, grupo y materia que se requiere hacer la reserva.

El esquema de Solicitudes se puede apreciar en la Figura 2.27.

Figura 2.27. Esquema de base de datos scara-db/Solicitudes

2.3.2.2 Implementación

Una vez definidas las estructuras de las clases a usar, y de las bases de datos, se procede con la implementación de la base en *SQL Server*, por medio de un *script*; un fragmento de éste se muestra en el Código 2.1.

También se llenó la tabla de ranuras de tiempo (`RTs`) con horarios de 7 a 20 horas como se muestra en el Código 2.2.

Se empezó por codificar el proyecto `ServicioWeb`, el cual tendrá una clase llamada `Componente` con la interfaz correspondiente. En esta interfaz se define el contrato de servicio, los métodos y clases que serán usados por el mismo.

```
create database scaradb
go
use scaradb
go

----Creacion de las tablas

--Tabla Usuarios
create table Usuarios(
idPro int identity(1,1),
cedula varchar (10),
nombresPro varchar (50),
apellidosPro varchar (50),
contrasena varchar (8),
email varchar (255),
imagenUrl varchar (255),
huella varbinary (2048),
IsAdmin int,
Habilitado int,
IsProf int,
primary key (idPro)
)
go
```

Código 2.1. Fragmento Script SQL – Definición de entidades

```
--Insercion de Ranuras de Tiempo

insert into RTs values ('Lunes','7:00','8:00')
go
insert into RTs values ('Lunes','8:00','9:00')
go
insert into RTs values ('Lunes','9:00','10:00')
go
```

Código 2.2. Fragmento Script SQL – Inserción de información

Para la realización del contrato y la serialización de los datos, se hace uso de las librerías señaladas en el Código 2.3, además en la interfaz se debe denotar el contrato del servicio, y en los métodos a implementar, el contrato de operación.

```
using System.Collections.Generic;
using System.ServiceModel;
using System.Threading.Tasks;
using System.Runtime.Serialization;

namespace ServicioWeb
{
    [ServiceContract]
    1 referencia | CarrionB, Hace 85 días | 1 autor, 11 cambios
    public interface IComponente
    {
        [OperationContract]
        1 referencia | CarrionB, Hace 238 días | 1 autor, 1 cambio
        int ValidarIngreso(string cedula, string pass, string parametro);
```

Código 2.3. Fragmento del Componente.cs – Implementación de librerías de servicio web

Además se deben definir los contratos de datos para las clases, y los miembros de datos, los cuales son aplicados en los métodos de acceso a los atributos de cada clase, e implica que se puede serializar. Para muestra de ello, se ilustra el Código 2.4.

```
[DataContract]
22 referencias | CarrionB, Hace 279 días | 1 autor, 1 cambio
public class Materia
{
    string nombre;
    int creditos;

    [DataMember]
    35 referencias | CarrionB, Hace 279 días | 1 autor, 1 cambio
    public string Nombre
    {
        get
        {
            return nombre;
        }

        set
        {
            nombre = value;
        }
    }
```

Código 2.4. Fragmento del Componente.cs – Clase Materia

Al proyecto ServidorDispositivos se añaden las referencias de ambos proyectos, para poder usarlos. Para llevar a cabo un servicio auto montado, primero se configura en el archivo del `App.config`, como se muestra en el Código 2.5, en el cual se define la URL (*Uniform Resource Locator*) a través de la cual el servicio podrá consumirse, así como el tipo de acuerdo, que en este caso es `basicHttpBinding`, el cual define el transporte y codificación de mensajes en XML.

Además, se define el comportamiento de éste, que permita ser depurado, es decir, en caso de presentarse algún error, el servicio enviará dónde y cuál ha sido el error.

Para que el servicio pueda ser consumido, éste se auto montará al iniciar el servidor. Para ello, se usa la clase `ServiceHost` de la librería `System.ServiceModel`, en cuyo constructor se define el tipo de servicio, en este caso, de clase `Componente`, y la misma URL ya definida en el archivo `App.config`. Una vez instanciado el objeto, se ejecuta el método `Open`, el cual permitirá que el servicio abra comunicaciones, y así poder enviar y recibir mensajes (Código 2.6) .

```
10  <services>
11    <service name="ServicioWeb.Componente" behaviorConfiguration="NewBehavior0">
12      <endpoint
13        address="http://X.X.X.X:8080/ServicioWeb/Componente"
14        binding="basicHttpBinding"
15        contract="ServicioWeb.IComponente" />
16    </service>
17  </services>
18  <behaviors>
19    <serviceBehaviors>
20      <behavior name="NewBehavior0">
21        <serviceMetadata httpGetEnabled="true" />
22        <serviceDebug includeExceptionDetailInFaults="True" />
23      </behavior>
24    </serviceBehaviors>
25  </behaviors>
```

Código 2.5. Fragmento de App.config – Etiquetas services y behaviors

```
59  using (ServiceHost anfitrion = new ServiceHost(typeof(Componente),
60      new Uri("http://X.X.X.X:8080/ServicioWeb/Componente")))
61  {
62      // Abre el objeto de comunicacion
63      anfitrion.Open();
```

Código 2.6. Fragmento de Program.cs – Inicio de servicio

Para la implementación en Firebase, se debe ingresar a la consola de la plataforma (Figura 2.28 (a)) a través de cualquier navegador, después de haber ingresado con una cuenta de Google. Una vez dentro, se elige la opción “Añadir proyecto” que aparece en pantalla (Figura 2.28 (b)), seguidamente se le asigna un nombre al mismo. Los siguientes dos

pasos dentro de la plataforma son de Google Analytics, los cuales son totalmente opcionales.

(a) (b)

Figura 2.28. (a) Pantalla principal de la consola de Firebase. (b) Asignación de nombre al proyecto a crear en Firebase

Una vez creado el proyecto, se elige en el menú a la izquierda la opción “Real time database” y se da clic en el botón “Crear base de datos” que aparece en pantalla como se muestra en la Figura 2.29. Seguido, en la plataforma se preguntará si se desea permitir o no la escritura de terceros, ya que esta se encuentra bloqueada por defecto.

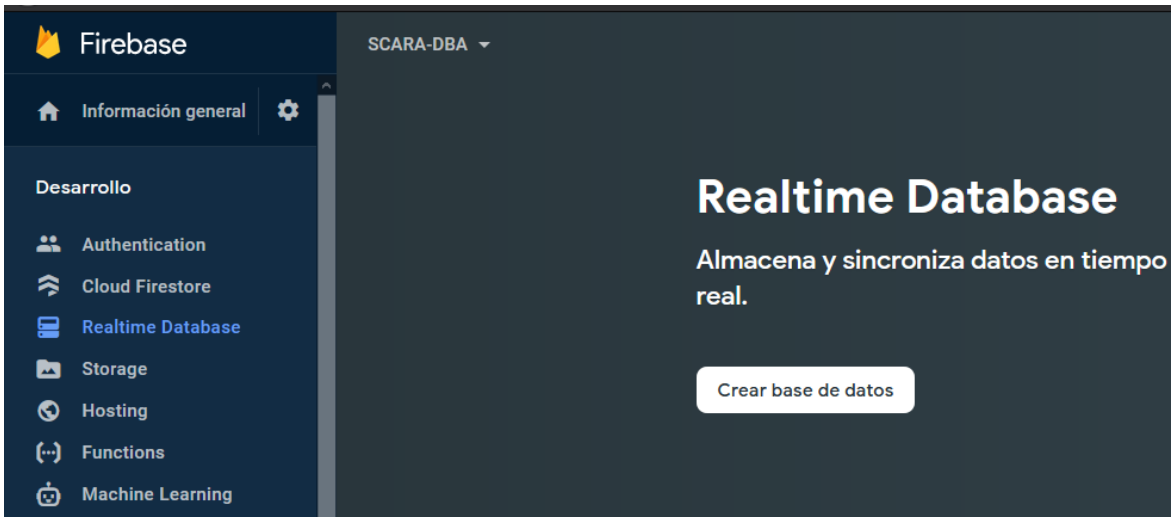


Figura 2.29. Realtime Database de Firebase

Posterior a la creación de la base de datos, se editarán finalmente las reglas de escritura y lectura de la misma (Figura 2.30) siguiendo la estructura ya definida; dichas reglas estarán habilitadas para cualquier usuario autenticado en el proyecto con Firebase.

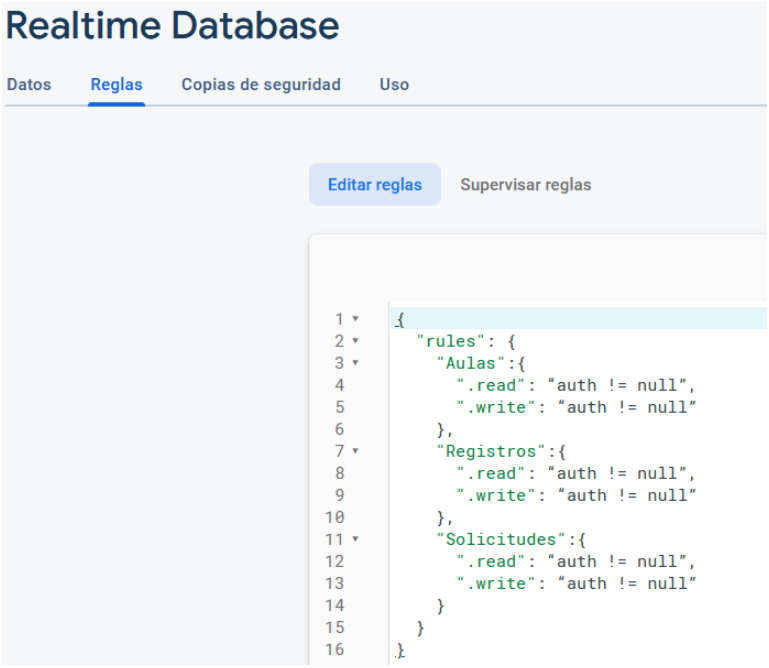
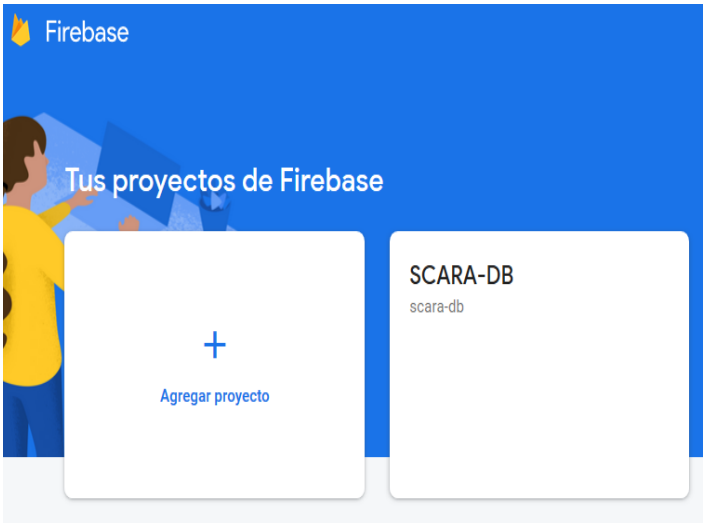
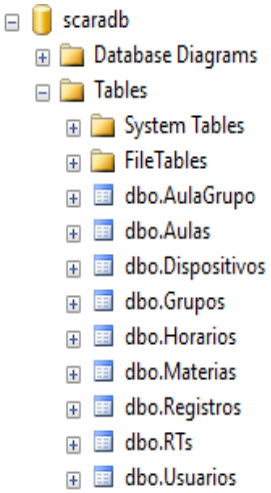


Figura 2.30. Configuración de reglas para la Realtime Database

2.3.2.3 Entregable

El entregable para este *Sprint*, es la creación de las bases de datos, tanto en Firebase como en *SQL Server*, las mismas que se muestran en la Figura 2.31.





(a) (b)

Figura 2.31. Bases de datos creadas en (a) Firebase y (b) SQL

También, se presenta el servicio web corriendo, al colocar la URL en un navegador web, y mostrarse éste en el mismo, como se aprecia en la Figura 2.32.

Figura 2.32. Servicio en navegador web

2.3.3 SPRINT 3

En este *sprint* se diseñan las Interfaces gráficas de usuario para la aplicación de Administración y la aplicación Móvil.

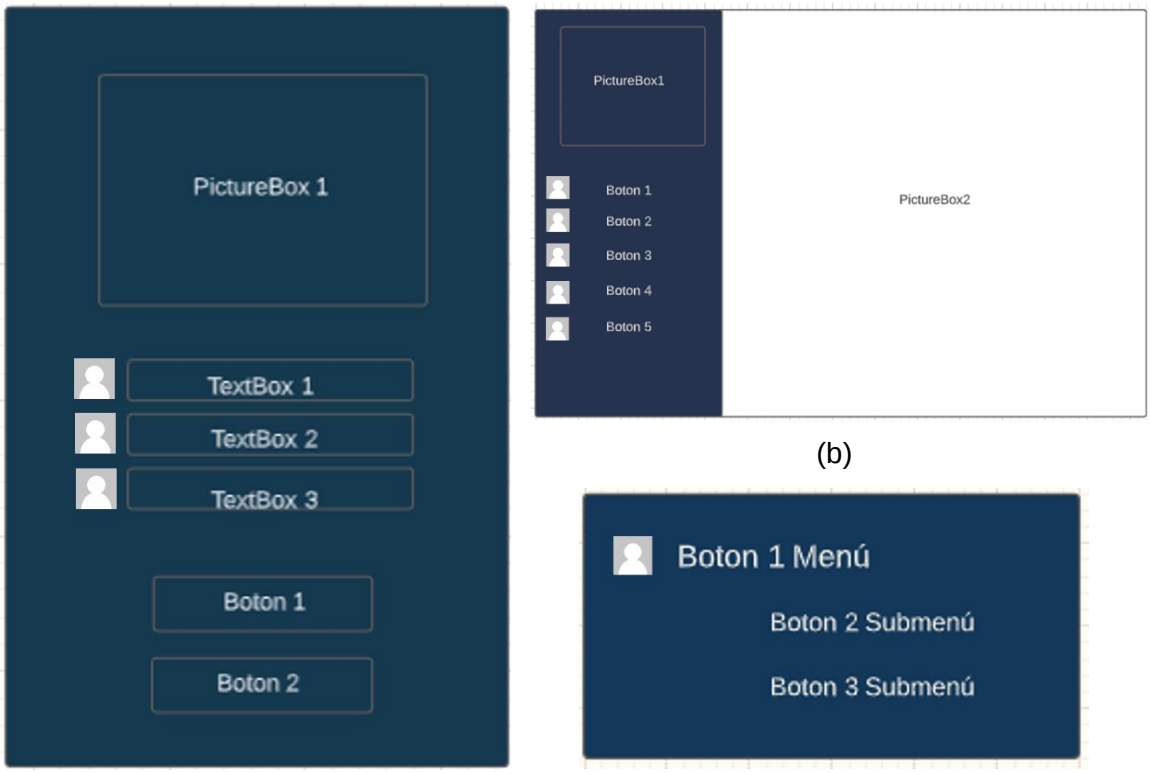
2.3.3.1 Diseño

Los bocetos de la interfaz gráfica se diseñaron en Lucidchart. Se crea una pantalla de ingreso, en la cual el usuario deberá colocar su número de cédula, contraseña y el semestre al que desea ingresar en el sistema, además se coloca el logo de la aplicación en la parte superior. La pantalla principal que se mostrará si el ingreso es exitoso, será una especie de *dashboard* que tendrá un menú en la parte izquierda con sub-menús desplegables y en la parte derecha se deja un espacio en el cual se abrirán el resto de formularios o submódulos. Los bosquejos de estas pantallas se presentan en la Figura 2.33.

Este menú tendrá varias opciones que se detallan brevemente a continuación.

- Usuarios: Despliega el submenú con opciones para agregar o editar usuarios.
- Aulas: Despliega el submenú con opciones para ver horarios, hacer cambios en horarios, reservar aulas, y atender solicitudes de reserva.
- Dispositivos: Despliega el submenú con opciones para agregar dispositivos y controlarlos de manera remota.
- Reportes: Despliega el submenú con opciones para generar reportes de asistencia, inasistencia y utilización de aulas.

- Actualización BD: Mostrará la pantalla correspondiente para la actualización de la base de datos de manera semestral.



(a) (b) (c)

Figura 2.33. Bosquejo de pantallas de (a) Ingreso, (b) Pantalla principal, (c) Submenú desplegado

Cada una de las opciones de cada submenú desplegará su respectivo formulario. A continuación, se explica rápidamente el diseño de las interfaces gráficas de cada submenú.

Las pantallas para agregar usuarios o editarlos tienen un diseño similar, con ciertas diferencias, como el botón que permitirá la actualización de huella, o el *combo box* para seleccionar el usuario a editar. Estos bosquejos se presentan en la Figura 2.34.

La pantalla para mostrar horarios de aulas, solo contará con un *combo box* para elegir el aula y un *Data Grid View*, que es una tabla para mostrar datos, en la que se colocarán los datos del horario correspondiente (Figura 2.35 (a)).

(a) (b)

Figura 2.34. Bosquejo de submódulos para (a) Agregar usuario (b) Actualizar usuario

Para la pantalla de cambios de horarios, se cuenta con dos *Data Grid View*, uno que mostrará los horarios dependiendo de lo seleccionado en el filtro, y otro para pre-visualizar los cambios que se realicen en base a las opciones presentes en la parte inferior del submódulo (Figura 2.35 (b)).

En el caso de la pantalla de reservas de aulas, se tiene un filtro basado en el profesor, y sus materias y grupos; se tendrá también un *Data Grid View* para mostrar el horario del aula a elegir, además de las opciones para definir el periodo de tiempo en que se requerirá el uso de dicha aula (Figura 2.35 (c)).

Para las solicitudes de reserva, en su pantalla se tiene un *Data Grid View*, el cual contará con los datos de las solicitudes enviadas, además de dos botones en cada fila que permitirán atender la solicitud o rechazarla (Figura 2.35 (d)). Los bosquejos de estas cuatro pantallas se muestran en la Figura 2.35.

Para las pantallas correspondientes a los dispositivos, la destinada a su agregación, contiene un campo para el ingreso de su dirección IP, además de un recuadro en el que se mostrarán los datos del mismo, y el botón para agregarlo (Figura 2.36 (a)).

Para el control de dispositivos, se presenta un submódulo con dos recuadros, uno con la lista de dispositivos y el otro para mostrar datos respecto a éste; además se tiene una serie de botones en la parte derecha que permitirá el manejo remoto de sus funciones. Estos bosquejos se presentan en la (Figura 2.36(b)).

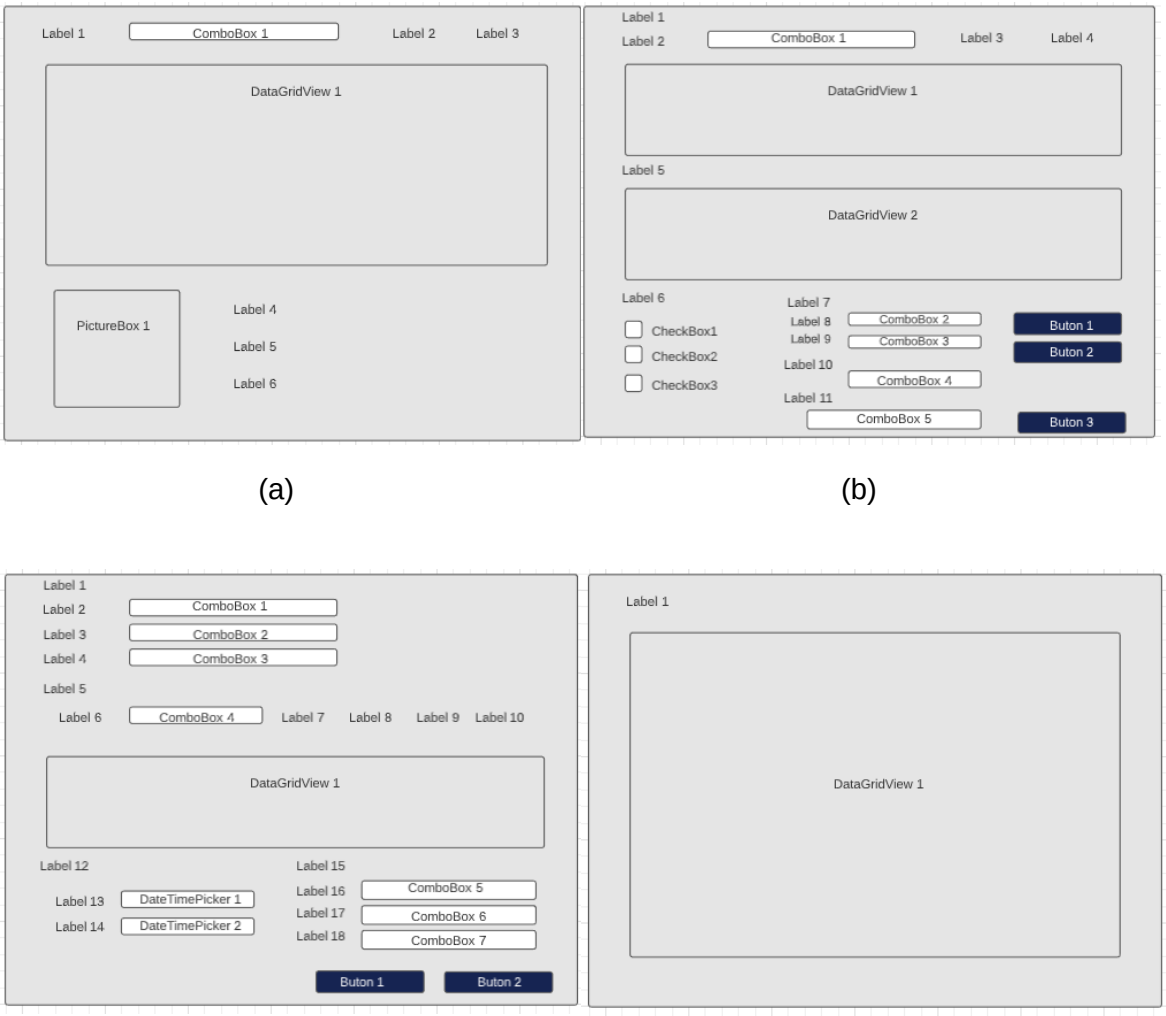
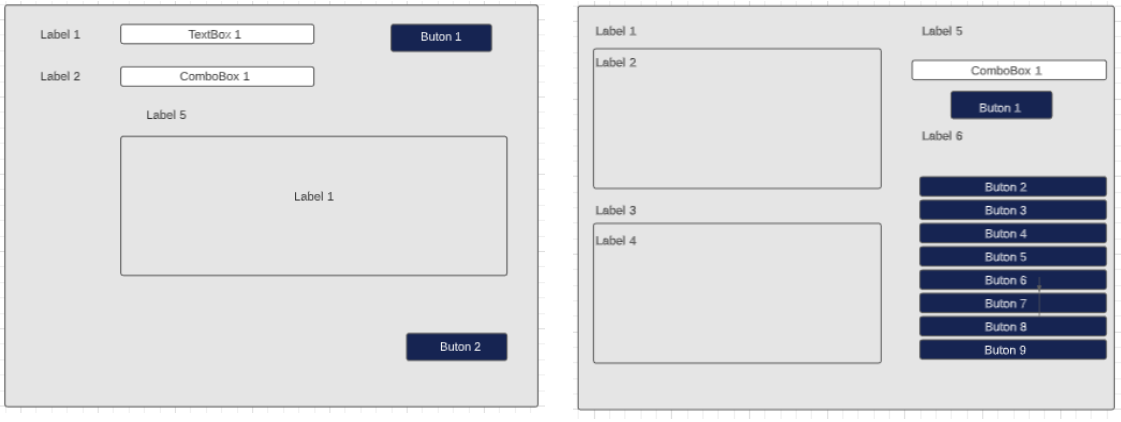


Figura 2.35. Bosquejo de submódulos para (a) Mostrar aulas, (b) Cambios de horarios, (c) Reservas de aulas y (d) Solicitudes de reserva

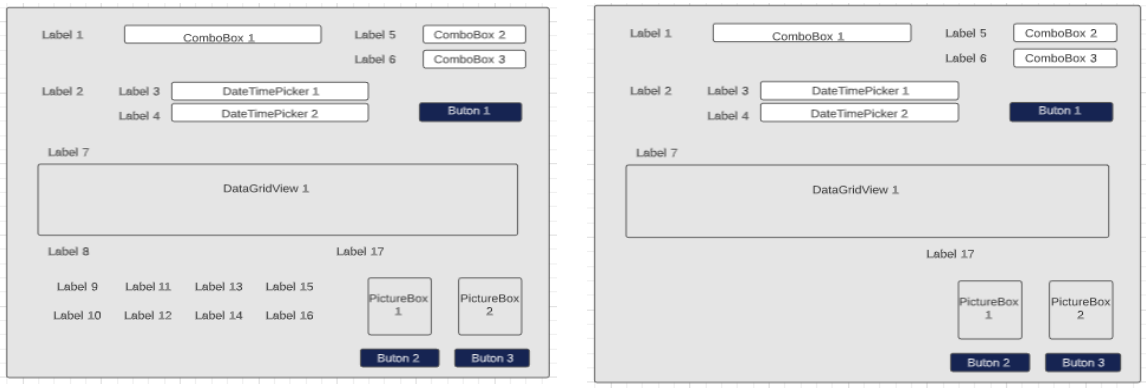
En el caso de los submódulos de reportes, los correspondientes a asistencias e inasistencias, tienen un diseño similar, con la diferencia en ciertos datos estadísticos a presentarse una vez se genera el reporte; mientras que el submódulo relacionado a reportes de utilización de aulas, difiere en el filtro a usar. Los bosquejos correspondientes se muestran en la Figura 2.37.

Para mitigar visualmente los tiempos de carga al usuario, se decidió también crear pantallas de espera, una para la carga de los submódulos y otra para la carga de cualquier operación interna en el submódulo. Estas pantallas se presentan en la Figura 2.38.



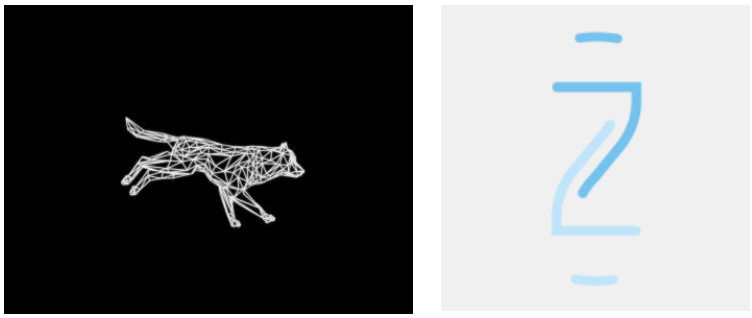
(a) (b)

Figura 2.36. Bosquejo de submódulos para (a) Agregar Dispositivo (b) Manejar Dispositivos



(a) (b)

Figura 2.37. Bosquejo de submódulos para Reportes de (a) Asistencia y (b) Inasistencia



(a) (b)

Figura 2.38. Diseños de pantallas de carga de (a) Submódulo y (b) de Operaciones

Además, se añade una pantalla de presentación, que se mostrará al iniciar el programa. Esta se presenta en la Figura 2.39.

Figura 2.39. Diseño de pantalla presentación

De igual manera que con la aplicación de escritorio, la aplicación móvil tendrá una pantalla de presentación y de ingreso a la misma (Figura 2.40).

Una vez se ingrese a la aplicación, se presenta una pantalla que tiene una barra de menú como encabezado, la cual tiene dos botones y el logo de la aplicación como se observa en la Figura 2.41 (a); el botón de la derecha corresponde a cerrar sesión y el de la izquierda desplegará el menú mostrado en la Figura 2.41 (b)

Una vez dentro de la aplicación, se podrá navegar a través de cuatro pantallas:

- Pantalla de inicio

La pantalla presente en la Figura 2.42 (a), muestra un saludo al usuario, y un espacio en el cual se mostrarán las solicitudes conjuntamente con su estado, además de 3 botones los cuales re-direccionarán hacia las otras pantallas existentes.

- Pantalla de horarios de clase

La pantalla correspondiente a la Figura 2.42 (b), contará con un selector en el cual se podrá elegir el aula a consultar y la fecha, junto con un botón para ejecutar la consulta.

- Pantalla de registros de asistencia

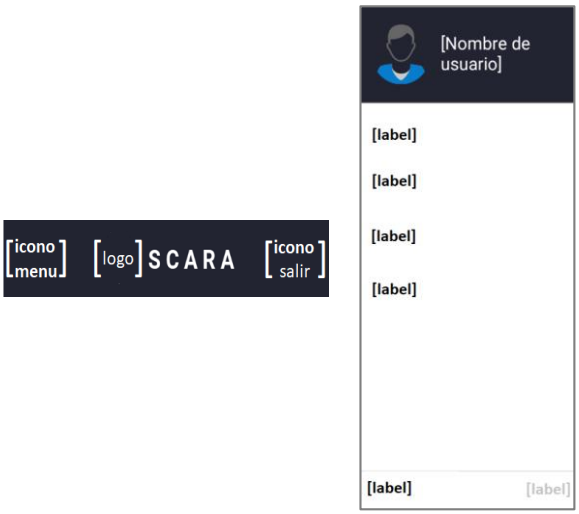
La pantalla de la Figura 2.42 (c), contará con un control que permitirá activar o no, un filtro por materia y grupo, además de los selectores para el mismo y de dos selectores de fecha. Finalmente, se tiene el botón que permitirá mostrar los resultados obtenidos.

- Pantalla de solicitud de reserva

La pantalla presente en la Figura 2.42 (d), contiene selectores para materias, aulas y grupos, además de selectores de fecha y horas para seleccionar el período de tiempo a reservar. Adicionalmente, cuenta con un botón para ejecutar la operación.

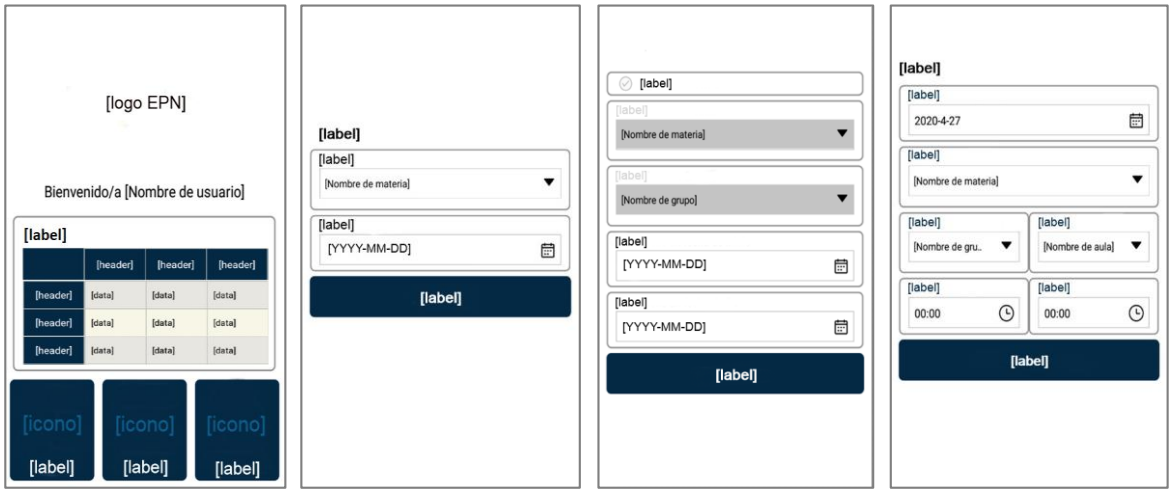
(a) (b)

Figura 2.40. Diseño de pantalla Presentación y bosquejo de pantalla de Ingreso de la Aplicación Móvil



(a) (b)

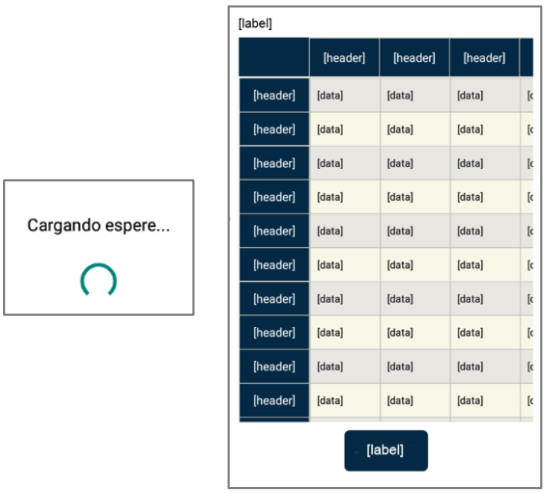
Figura 2.41. Bosquejo de (a) Barra de menú (b) Menú desplegable



(a) (b) (c) (d)

Figura 2.42. Bosquejo de pantalla (a) Principal, (b) Horarios de aulas, (c) Registros de asistencia y (d) Solicitudes de reserva de la aplicación móvil

Aparte se tendrán, dos pantallas modales, las pantallas de carga y de resultados, mostradas en la Figura 2.43.



(a) (b)

Figura 2.43. Bosquejo pantalla (a) de Carga (b) de Resultados para la aplicación móvil

2.3.3.2 Implementación

En la implementación de los submódulos, se recibe ayuda directa del editor de Visual Studio, debido a que, con sus herramientas se arrastran los controles, y el código se genera automáticamente en `NombreForm.Designer.cs`. Se muestra un fragmento de uno de los archivos en el Código 2.7.

```
3   partial class frmMenuPrincipal
4   {
5       /// <summary> Required designer variable.
8       private System.ComponentModel.IContainer components = null;
9
10      /// <summary> Clean up any resources being used.
            33 referencias | CarrionB, Hace 287 días | 1 autor, 1 cambio
14      protected override void Dispose(bool disposing)...
22
23      #region Windows Form Designer generated code
24
25      /// <summary> Required method for Designer support - do not modify the contents of this m
            1 referencia | CarrionB, Hace 113 días | 2 autores, 17 cambios
29      private void InitializeComponent()
30      {
31          this.components = new System.ComponentModel.Container();
32          System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.
33          this.panelContenedor = new System.Windows.Forms.Panel();
34          this.pnlFormularios = new System.Windows.Forms.Panel();
35          this.pnlEstado = new System.Windows.Forms.Panel();
36          this.pbLogoPeq = new System.Windows.Forms.PictureBox();
```

Código 2.7. Fragmento de frmMenuPrincipal_Designer.cs autogenerado

Se programó también la animación del menú de la pantalla principal, para ello, se definió un *booleano* por cada opción en el menú. Estos *booleanos* permitirán determinar si el submenú correspondiente está o no colapsado.

Se creó la función `EncogerPaneles` a la cual se le pasa un objeto de tipo `System.Windows.Forms.Panel`, el cual servirá para identificar al panel que debe desplegarse. Los paneles tienen definidos los parámetros `MinimumSize` y `MaximumSize`, con este último, se genera un algoritmo el cual busca el alto mínimo en este parámetro de los paneles presentes en el menú. Este algoritmo se muestra en el Código 2.8.

```
256 /// Parametro arbitrario para obtener el alto minimo en el parametro MaximumSize
257 int altoMax = 1000;
258 /// Alto minimo de los paneles del menu
259 int altoMin = 50;
260
261 /// Lazo que recorre los paneles del menu
262 foreach (Panel c in pnlMenu.Controls)
263 {
264     /// Si la propiedad Height del MaximumSize del panel de la iteracion y este
265     /// no es el panel del logo (pnlLobo)
266     if (c.MaximumSize.Height < altoMax && c.Name !="pnlLobo")
267     {
268         /// Se guarda su valor en la variable correspondiente
269         altoMax = c.MaximumSize.Height;
270     }
271 }
```

Código 2.8. Fragmento de función EncogerPaneles

Consecuentemente se recorre los paneles de nuevo, comprobando que no sea el panel ingresado como parámetro de entrada, se encoge el panel un tamaño proporcional a 10 pixeles, que es el valor con el que se ensancha un panel; y se comprueba si el panel llegó a su `MinimumSize`, para asignar el *booleano* correspondiente en *true.* Lo explicado se muestra en el Código 2.9.

También se agregó un `Timer` para cada botón en el menú, éste permitirá encoger o expandir progresivamente el panel seleccionado. Cada uno de estos `timers` al iniciarse, ejecuta la función `AccionarPanelSeleccionado`, mostrada en el Código 2.10, la cual permite realizar la acción enunciada.

Finalmente, se tiene la función `CargarForm`, la cual muestra la pantalla de carga a la vez que se carga el submódulo elegido. Además, esta función recibe un entero, el cual permitirá elegir el submódulo a cargar mediante una sentencia `case`, y se accionará al presionar cualquiera de los botones del submenú.

```
273  foreach (Panel c in pnlMenu.Controls)
274  {
275      if (c.Name != panel.Name)
276      {
277          c.Height -= (c.MaximumSize.Height * 10 / (altoMax - altoMin));
278
279          if (c.Size == c.MinimumSize)
280          {
281              if (c.Name == "pnlUsuario")
282              {
283                  isCollapsedUser = true;
284              }
285              if (c.Name == "pnlDisp")
286              {
287                  isCollapsedDisp = true;
288              }
289
290              if (c.Name == "pnlAulas")
291              {
292                  isCollapsedAulas = true;
293              }
294              if (c.Name == "pnlReport")
295              {
296                  isCollapsedReport = true;
```

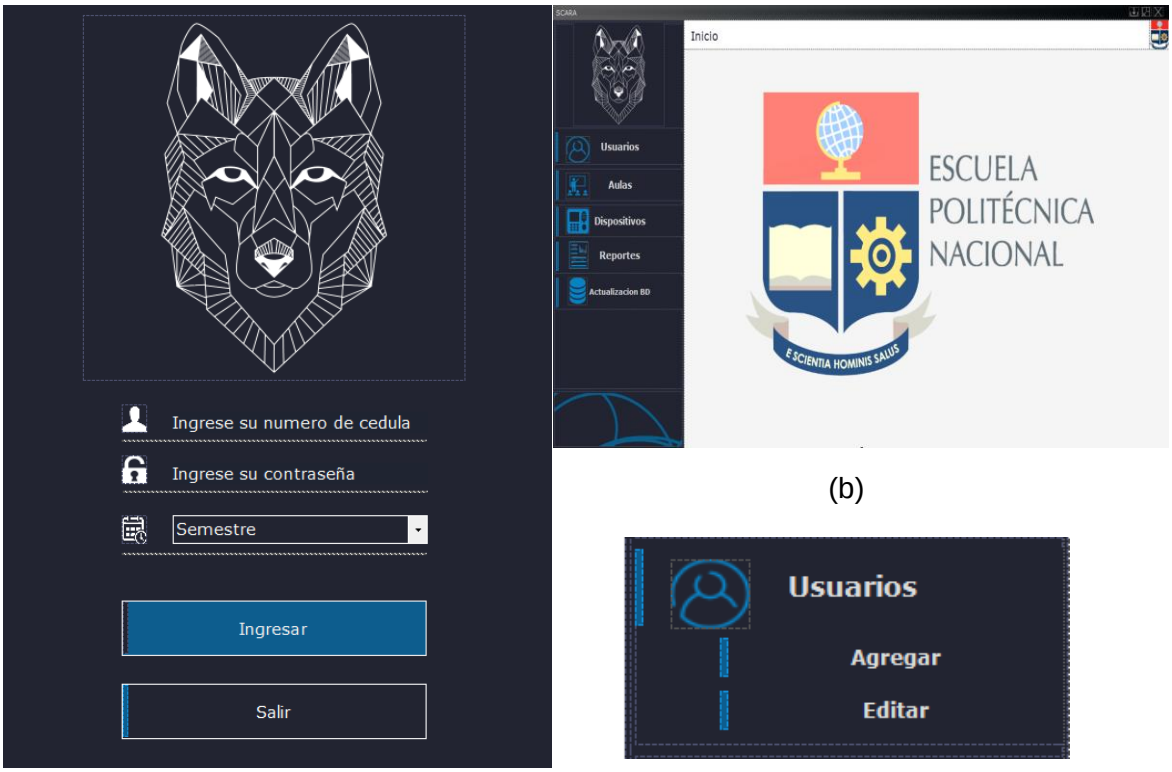
Código 2.9. Algoritmo para encoger paneles

```
451  private void AccionarPanelSeleccionado(ref bool IsCollapsed, Panel panel, System.Windows.Forms.Timer timer)
452  {
453      /// si el menu no esta desplegado
454      if (IsCollapsed)
455      {
456          /// se encoge el resto de paneles
457          EncogerPaneles(panel);
458          /// el menu se expande en el transcurso del tiempo del timer en 10 pixeles
459          panel.Height += 10;
460          /// si el tamanio del submenu ha llegado a su tamanio maximo
461          if (panel.Size == panel.MaximumSize)
462          {
463              /// se detiene el timer por lo que deja de expandirse
464              timer.Stop();
465              /// se cambia el estado de despliegue del submenu
466              IsCollapsed = false;
467          }
468      }
469      /// si el menu esta desplegado
470      else
471      {
472          /// menu empieza a encogerse
473          panel.Height -= 10;
474          /// si llego a su tamanio minimo
475          if (panel.Size == panel.MinimumSize)
476          {
477              /// el timer se detiene
478              timer.Stop();
479              /// cambia el estado de despliegue del submenu
480              IsCollapsed = true;
481          }
```

Código 2.10. Función para encoger/desplegar paneles

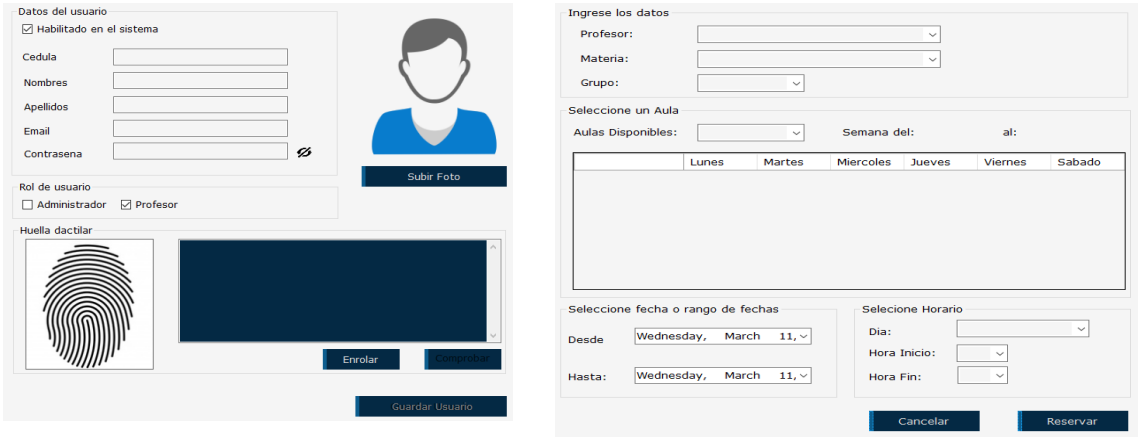
2.3.3.3 Entregable

El entregable del presente *sprint* consta de los diseños de las interfaces gráficas de la aplicación de administración, conjuntamente con la funcionalidad básica de la interfaz de la pantalla principal de la aplicación, mismas que se presentan en las siguientes Figuras Figura 2.44, Figura 2.45, Figura 2.46 y Figura 2.47.



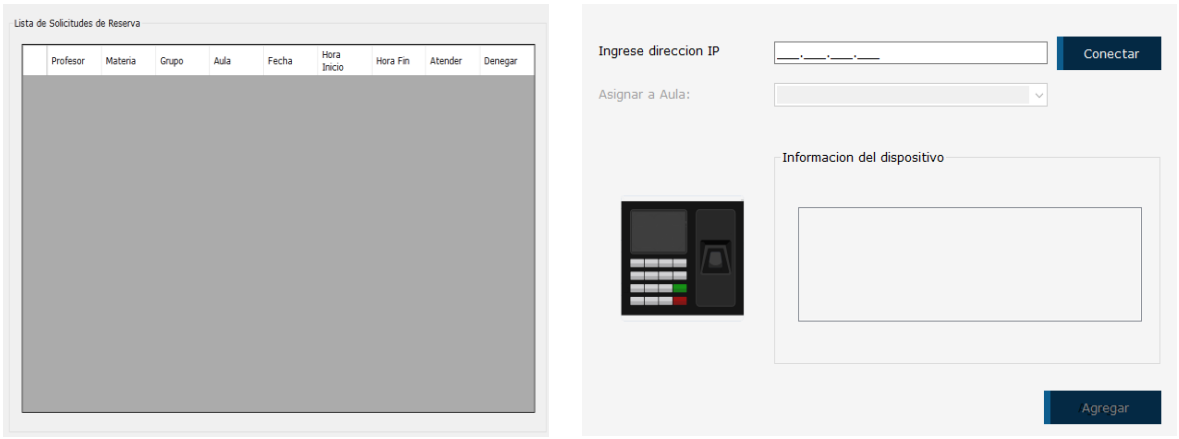
(a) (b) (c)

Figura 2.44. Interfaz gráfica de pantallas (a) Ingreso, (b) Pantalla principal, (c) Submenú desplegado



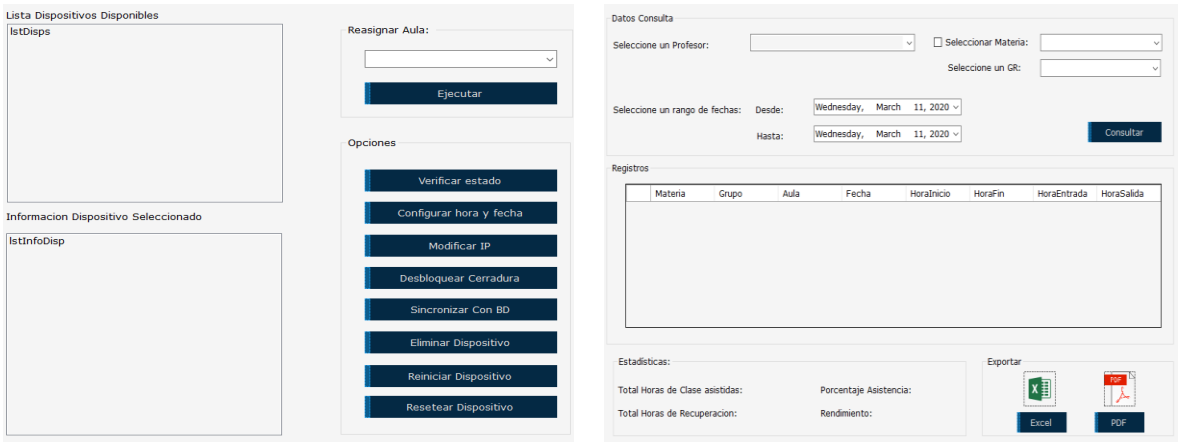
(a) (b)

Figura 2.45. Interfaz gráfica de submódulos (a) Agregar Usuario (b) Reserva de Aulas



(a) (b)

Figura 2.46. Interfaz gráfica de submódulos (a) Solicitudes de Reserva (b) Agregar Dispositivos



(a) (b)

Figura 2.47. Interfaz gráfica de submódulos (a) Manejar Dispositivos (b) Reportes de Asistencia

2.3.4 SPRINT 4

En este *sprint* se programará el ingreso a la Aplicación de Administración.

2.3.4.1 Diseño

El proceso para validar el ingreso a la aplicación de administración se muestra en el diagrama de secuencia de la Figura 2.48.

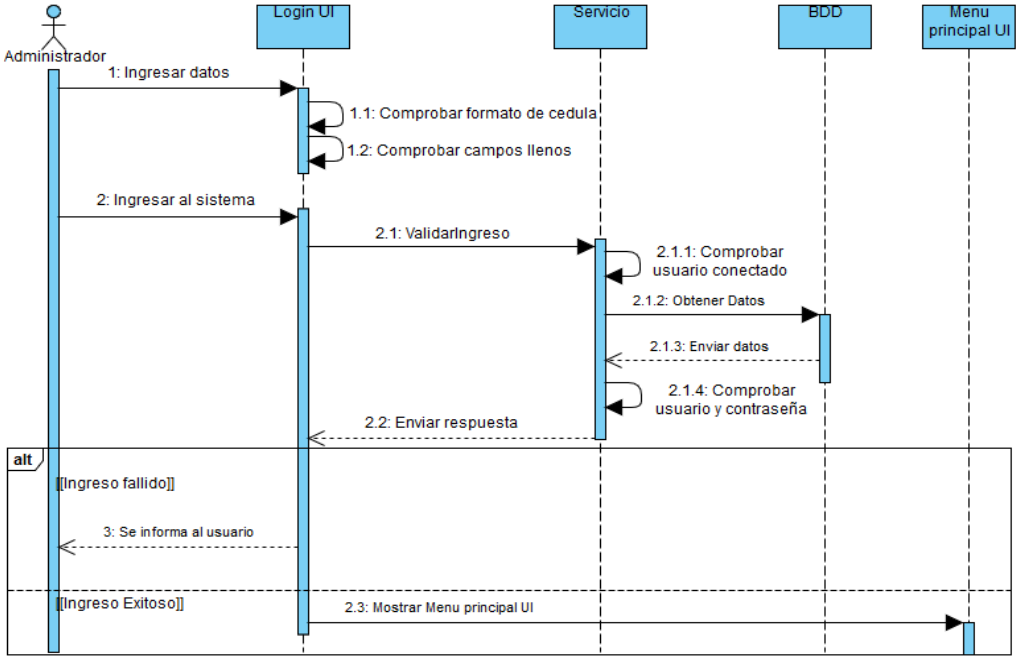


Figura 2.48. Diagrama de secuencia: Ingreso a Aplicación de Administración

2.3.4.2 Implementación

En el lado del servidor, primero se instancia en el sistema una lista la cual guarda los usuarios conectados, además, del método `ComprobarConexionClienteAdmin`, el cual permite validar si el usuario ya se encuentra conectado al sistema, y por tanto validar la ejecución o no de cualquier otro método.

En la función `ValidarIngreso`, misma que permite al usuario ingresar a la aplicación, se hace uso de la función de comprobación antes mencionada, para reforzar la seguridad del sistema. Seguidamente, se debe establecer la conexión a la base de datos en SQL Server; para ello, se creó la función `ObtenerConexion` la cual devuelve un objeto de tipo `SqlConnection` que se instancia en base al nombre del servidor de base de datos, la librería de red que permite transmitir los datos a través de esta conexión, el nombre de la base de datos, y la habilitación de seguridad como se muestra en el Código 2.11.

Esta función se usará cada que se necesite interactuar con la base existente en SQL Server.

Para validar el ingreso, se enviarán los tres parámetros requeridos en el Login; con la cédula se comprueba la existencia del usuario y se compara la contraseña ingresada, y en base a esto se permite o no el ingreso. Además, con el parámetro de semestre, se comprobará si es el semestre actual, con lo cual se habilitarán o no la mayoría de funciones (Código 2.12).

En el lado del cliente, se implementa la función `TestCedula` la cual permite comprobar que el número de cédula ingresado sea correcto. Para ello se debe tomar en cuenta lo siguiente:

```
131  public static SqlConnection ObtenerConexion()
132  {
133      /// SqlConnection representa una conexion a una base de datos SQL Server
134      /// Se crea y se setea una conexion conec del tipo SqlConnection, que recibe como
135      /// parametros de entrada la cadena de conexion de la base de datos.
136      SqlConnection conec = new SqlConnection(@"Data Source=" +
137          Environment.GetEnvironmentVariable("COMPUTERNAME") +
138          ",1433; Network Library=DBMSSOCN; Initial Catalog=dbAlfa; Integrated Security=True;");
139
140      /// Abre la conexion de la base de datos
141      conec.Open();
142
143      /// Retorna la conexion
144      return conec;
145  }
```

Código 2.11. Función ObtenerConexion

```
172  using (SqlDataReader reader = comando.ExecuteReader())
173  {
174      /// Lazo que se cumple mientras el reader permanezca leyendo datos.
175      while (reader.Read())
176      {
177          /// Una ver obtenido los dato del profesor correspondiente, se compara la conttrasenia
178          if ((reader["Contrasena"].ToString()) == pass)
179          {
180              /// En caso de ser correcta, se añade a la lista de clientes conectados y se envia
181              /// exito en uno
182              clientesConectados.Add(cedula);
183              exito = 1;
184              /// si el semestre es igual al parametros enviado como parametros de entrada
185              if (semestre == parametro)
186              {
187                  /// se envia a exito en tres, lo que significa que el cliente ingreso en el periodo actual
188                  exito = 3;
189              }
190          }
191      }
192  }
```

Código 2.12. Fragmento función ValidarIngreso

- Se multiplican los números de cada posición impar por 2, y si el resultado de cada operación es mayor o igual a 10, se resta nueve.
- Se suman cada uno de estos resultados con los números en las posiciones pares a excepción del décimo número, que es el verificador.
- Finalmente se resta el resultado de la sumatoria de la decena superior, el resultado de esta que deberá ser igual al décimo digito.

En base a esto se obtiene la función ilustrada en el Código 2.13.

Para poder consumir el servicio en el cliente, se debe agregar una referencia de servicio la cual mediante la directiva `using`, puede ser utilizada en el código. Adicionalmente en el mismo se instancia un objeto proxy con el cual se llamará a todas las funciones a

implementarse dentro del servicio. Para este caso puntual se hará uso del método `ValidarIngreso`, y en base a su respuesta, se informará al usuario, siempre y cuando se haya ingresado todo lo requerido en el Login.

Como se había señalado, de la selección del semestre dependerá las funciones que se habiliten, lo cual se logrará con el atributo `estado` dentro del menú principal, el mismo que es evaluado para mostrar únicamente los componentes necesarios como se indica en el método `Load` del formulario en el Código 2.14.

```
455  private bool TestCedula(string cedula)
456  {
457      /// Esta variable guarda la suma de los numeros en las posiciones impares
458      int sum1 = 0;
459
460      /// Esta variable guarda la suma de los numeros en las posiciones pares, hasta la octava, ya que
461      /// el decimo es como un checksum
462      int sum2 = 0;
463
464      /// chk guardara el resultado del las operaciones, mientras que aux sera de utilidad para varias
465      /// operaciones
466      int chk, aux = 0;
467
468      /// Recorre el string cedula por los lugares impares, que al ser vector, empieza en 0, y recorre de 2 en 2
469      for (int i = 0; i < 9; i = i + 2)
470      {
471          Int32.TryParse(cedula.Substring(i, 1), out aux);
472
473          /// Se extrae el numero de la posicion indicada, multiplica por 2, y se compara con 10
474          if (aux * 2 >= 10)
475          {
476
477              /// Si es mayor a 10, se le resta 9, y se aniade a la suma
478              sum1 = sum1 + aux * 2 - 9;
479          }
480          else
481          {
482              /// Caso contrario, solo se aniade a la suma
483              sum1 = sum1 + aux * 2;
484          }
485      }
486
487      /// En el caso de las posiciones pares, solo se suman
488      for (int i = 1; i < 9; i = i + 2)
489      {
490          Int32.TryParse(cedula.Substring(i, 1), out aux);
491          sum2 = sum2 + aux;
492      }
493
494      /// Se aniaden ambas sumatorias, se obtiene el residuo de esta suma y se resta de 10
495      chk = 10 - (sum1 + sum2) % 10;
496
497      /// Ese resultado se compara con el ultimo digito, si son iguales, es valida
498      Int32.TryParse(cedula.Substring(9, 1), out aux);
499      if (chk == aux)
500      {
501          return true;
502      }
503      return false;
504  }
```

Código 2.13. Función TestCedula

```
59  private void MenuPrincipal_Load(object sender, EventArgs e)
60  {
61      /// para que no exista seleccion por defecto
62      this.Select();
63
64      /// se inicializa en verdadero porque en la ventada aparece inicialmente colapsado
65      isCollapsedUser = true;
66      isCollapsedDisp = true;
67      isCollapsedAulas = true;
68      isCollapsedReport = true;
69
70      /// si el estado esta en uno, significa que no se encuentra en el semestre actual, por tanto
71      /// se limitan las funciones
72      if (estado == 1)
73      {
74
75          /// no son visibles los paneles mencionados a continuacion
76          pnlUsuario.Visible = false;
77          pnlDisp.Visible = false;
78          pnlActualizacionBd.Visible = false;
79          pnlAulas.MaximumSize = new Size(259,80);
80      }
81  }
```

Código 2.14. Función MenuPrincipal_Load

2.3.4.3 Entregable

El entregable consta del funcionamiento del Login para el ingreso a la aplicación, además de las validaciones y mensajes necesarios para el usuario.

2.3.5 SPRINT 5

En este *sprint* se presentará el manejo de usuarios conjuntamente con la captura y procesamiento de su huella dactilar, es decir, los submódulos Agregar Usuario y Actualizar Usuarios que se encuentran dentro del módulo Usuarios.

2.3.5.1 Diseño

El proceso tanto para agregar y actualizar usuarios a través de la aplicación de administración se muestra en el diagrama de secuencia de la Figura 2.49.

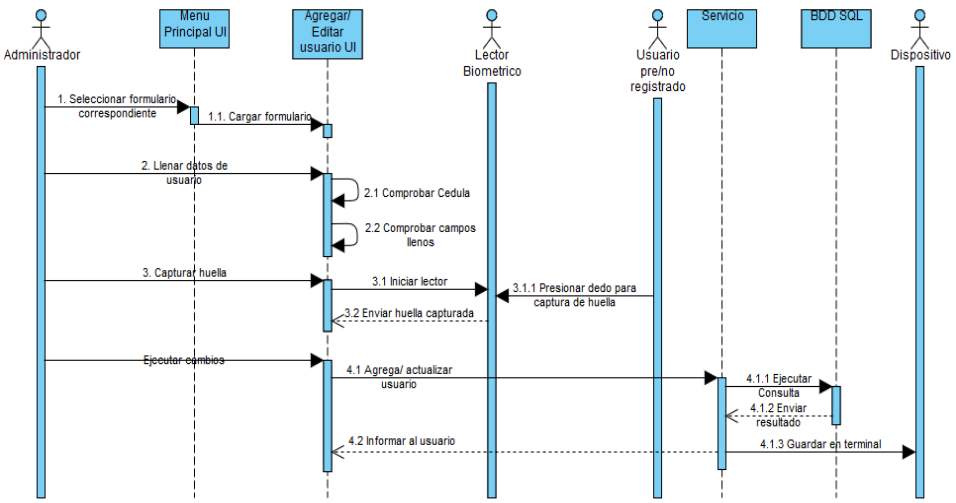


Figura 2.49. Diagrama de secuencia de los submódulos Agregar y Actualizar Usuarios.

2.3.5.2 Implementación

Para poder recuperar la huella con el lector biométrico ZK9500, se referencia a la librería `libzkfpcsharp`, de la cual se hace uso a través de la directiva `using` en la clase `FuncEscaner`, la misma que se encuentra en la Aplicación de Administración.

En esta clase se instancian varios atributos como manipuladores, arreglos de bytes donde se guardarán los escaneos de las huellas dactilares, y demás atributos relacionados con la imagen a obtener de dicho escaneo, atributos tales como alto, ancho o la cantidad de pixeles por pulgada.

Se instancia la función `LoadScanner`, la cual permite inicializar el lector (línea 82), establecer conexión con éste (línea 86), y separar un espacio en la cache del PC (línea 99), ya que el lector no cuenta con memoria interna. Este proceso está ilustrado en el Código 2.15.

Seguidamente, se inicializan los arreglos para guardar las huellas, además de las propiedades de imagen en base a las del lector, como se muestra en el Código 2.16. Es necesario señalar que esta función será llamada para cualquier operación de captura o comparación de huella para este lector.

```
78  public static bool LoadScanner(TextBox txtRes)
79  {
80      bool exito = true;
81      /// Inicializa el dispositivo
82      if ((ret = zkfp2.Init()) == zkfperrdef.ZKFP_ERR_OK)
83      {
84          /// MessageBox.Show("Dipositivo Inicializado");
85          /// Valida conexion con dispositivo
86          if (IntPtr.Zero == (mDevHandle = zkfp2.OpenDevice(0)))
87          {
88              string message = "Conexion fallida";
89              string caption = "Error";
90              MessageBoxButtons buttons = MessageBoxButtons.OK;
91              MessageBoxIcon icon = MessageBoxIcon.Error;
92              MessageBox.Show(message, caption, buttons, icon);
93              return exito = false;
94          } else
95          {
96
97              /// Permite crear un algoritmo de cache del ordenador, que optimizara la gestion de la informacion de la misma,
98              /// ya que el dispositivo no posee de memoria interna y usa la cache para guardar las muestras
99              if (System.IntPtr.Zero == (mDBHandle = zkfp2.DBInit()))
100             {
101                 string message = "Conexion a cache fallida";
102                 string caption = "Error";
103                 MessageBoxButtons buttons = MessageBoxButtons.OK;
104                 MessageBoxIcon icon = MessageBoxIcon.Error;
105                 MessageBox.Show(message, caption, buttons, icon);
106                 zkfp2.CloseDevice(mDevHandle);
107                 mDevHandle = System.IntPtr.Zero;
108                 return exito = false;
109             }
```

Código 2.15. Fragmento función LoadScanner

```
113 /// Se define un arreglo de bytes, en el que se registraran los 3 intentos de enrolamineto
114 for (int i = 0; i < 3; i++)
115 {
116     RegTmps[i] = new byte[2048];
117 }
118
119 /// Variable temporarl que se usara para recuperarun paamtro del dipositivo, dependiendo
120 /// del codigo usado posteriormente (1,2,3)
121 byte[] paramValue = new byte[4];
122 int size = 4;
123
124 /// Para obtener los parametros de la imagen obtenida por el biometrico, se usa la
125 /// funcion GetParameters, con el codigo correspondiente (1 = Ancho, 2 = alto, 3 = DPI)
126
127 /// Obtiene el ancho de la imagen
128 zkfp2.GetParameters(mDevHandle, 1, paramValue, ref size);
129 zkfp2.ByteArray2Int(paramValue, ref mfpWidth);
```

Código 2.16. Inicialización dentro de la función LoadScanner

Para poder mostrar la huella en pantalla, se crea la función `ArregloBytesaImagen`. En esta función se instancia una imagen de mapa de bits en base a los parámetros de dimensión del lector, y se usa el puntero del primer pixel de esta para guardar todos los bytes del arreglo. Además, se define una paleta de colores para poder visualizar la imagen de la huella, todo esto se ilustra en el Código 2.17.

```
303 public static Image ArregloBytesaImagen(ref byte[] arr, int width, int height)
304 {
305     /// genera la imagen en mapa de bytes, se especifica que el formato sera ocho bits por pixel, indexado
306     /// lo que le permitira tener 256 colores
307     var output = new Bitmap(width, height, PixelFormat.Format8bppIndexed);
308
309     /// genera un rectangulo que especifica localizacion y tama;o que definira la region donde se bloquearan los bits
310     var rect = new Rectangle(0, 0, width, height);
311
312     /// bloqueo de bits para un mejor manejo de la calidad de la imagen mediante el uso de lockbits (porque si no se usa
313     /// lo que disminuye la calidad d ela imagen y para esta aplicacion esto es un aspecto relevante)
314     var bmpData = output.LockBits(rect, ImageLockMode.ReadWrite, output.PixelFormat);
315
316     /// obtiene la direccion del primer pixel del bitmap
317     var ptr = bmpData.Scan0;
318
319     /// Lazo en el que se copiarar bit a bit en el bufffer de la imagen
320     for (var i = 0; i < height; i++)
321     {
322         /// copia del array que contine los bytes de la huella al puntero con el ancho dado para crear la imagen
323         Marshal.Copy(arr, i * width, ptr, width);
324         /// avanza el mismo ancho al siguiente puntero para seguir guardando los datos
325         ptr = new IntPtr(ptr.ToInt64() + width);
326     }
327
328     /// Se instancia una paleta de colores por defecto
329     ColorPalette tempPalette;
330     {
331         /// genera la imagen en mapa de bytes, se especifica que el formato sera ocho bits por pixel, indexado
332         /// lo que le permitira tener 256 colores
333         using (Bitmap tempBmp = new Bitmap(1, 1, PixelFormat.Format8bppIndexed))
334         {
335             /// Se define la paleta de colores en base a la propiedad de la imagen de un pixel creada anteriormente
336             tempPalette = tempBmp.Palette;
337         }
338         /// tenemos un array de estructura de colores
339         for (int i = 0; i < 256; i++)
340         {
341             tempPalette.Entries[i] = Color.FromArgb(i, i, i);
342         }
```

Código 2.17. Fragmento función ArregloBytesaImagen

Para la captura de la huella dactilar, se usa la función `Enroll`, la cual tienen un lazo que espera a que se hayan hecho 3 escaneos de la huella en el lector. Dentro de este lazo, se trata de adquirir una primera lectura (línea 183), una vez obtenida, en las siguientes se comprueba que haya similitud con la primera (línea 191); sino, se informará paulatinamente de las incidencias de la operación al usuario (Código 2.18).

Una vez se tengan las tres muestras, se ejecuta una operación de combinación de estas (línea 211 - Código 2.19).

También se creó la función `Match`, la cual permitirá la comparación de la huella ya adquirida, con una muestra tomada del lector, para comprobar que fue correctamente obtenida, tomando un 80% como mínimo de similitud.

```
182  /// Lectura de la huella
183  ret = zkfp2.AcquireFingerprint(mDevHandle, FPBuffer, CapTmp, ref cbCapTmp);
184  /// tiempo de espera para procesamiento
185  Thread.Sleep(500);
186  /// validacion de la lectura de la huella
187  if (ret == 0)
188  {
189      /// Comprueba que se esta tomando la muestra del mismo dedo
190
191      if (RegisterCount > 0 && zkfp2.DBMatch(mDBHandle, CapTmp, RegTmps[RegisterCount - 1]) <= 0)
192      {
193          txtRes.AppendText("Por favor utilice el mismo dedo." + System.Environment.NewLine);
194          break;
195      }
```

Código 2.18. Fragmento función Enroll

```
206  /// Verifica si ya se ha tomado el numero maximo de muestras
207  if (RegisterCount >= REGISTER_FINGER_COUNT)
208  {
209
210      /// Varifica que se hayan combinado correctamente las muestras, y se haya aniadido a la base de datos
211      if (zkfp.ZKFP_ERR_OK == (ret = zkfp2.DBMerge(mDBHandle,
212          RegTmps[0], RegTmps[1], RegTmps[2], RegTmp, ref cbRegTmp))
213          && zkfp.ZKFP_ERR_OK == (ret = zkfp2.DBAdd(mDBHandle, iFid, RegTmp)))
214      {
215          /// Si fue exitoso, aumenta en uno elid de la huella
216          iFid++;
217          txtRes.AppendText("Enrolamiento exitoso" + System.Environment.NewLine);
218          zkfp2.Terminate();
219          return RegTmp;
220      }
221      else
222      {
223          /// Sino muestra que vergas paso
224          txtRes.AppendText("Enrolamiento fallido, error con codigo: " + ret + System.Environment.NewLine);
225          zkfp2.Terminate();
226      }
227      break;
```

Código 2.19. Verificación de obtención de huella en función Enroll

Seguidamente, en el submódulo Agregar Usuarios, se colocan las validaciones correspondientes, como la comprobación de número de cédula, usada anteriormente, solo letras en nombres y apellidos, solo números en la contraseña. Adicionalmente, se

autogeneran las direcciones de correo electrónico con el formato nombre1.apellido1@epn.edu.ec.

Para agregar una foto al perfil del usuario, se comprueba que su cédula haya sido ingresada en el `textBox` correspondiente, ya que servirá como su identificativo al guardarse en *Firebase Storage*. La imagen será seleccionada de la PC del administrador para lo cual se le presentará un cuadro de diálogo.

Una vez seleccionada, es convertida en un *stream* de datos para poder ser transmitida por la red; seguidamente se llama a la función `AgregaImagenAsync` a través del proxy para guardarla y obtener la URL de la misma. Este proceso se puede observar en el Código 2.20.

```
272  /// se obtiene la ruta absoluta del mismo
273  string filename = System.IO.Path.GetFullPath(ofd.FileName);
274  ///  Se obtiene un stream para la transmision del archivo
275  var stream = File.Open(filename, FileMode.Open);
276  /// se crea un arreglo de bytes del tamaño del stream
277  byte[] bytes = new byte[stream.Length];
278  /// Se usa la funcion read para obtener y escribir sus datos en el arreglo de bytes
279  stream.Read(bytes, 0, bytes.Length);
280  /// Se llama a la funcion AgregaImagenAsync para almacenarla y obtener la url
281  /// de la misma
282  imagenUrl = await proxy.AgregaImagenAsync(bytes, txtCedula.Text);
283  /// se cierra el stream para liberar el archivo
284  stream.Close();
```

Código 2.20. Algoritmo para almacenamiento de Imagen en Firebase Storage

Finalmente, para agregar el usuario al sistema, se comprueba que todos los campos estén llenos, que exista la imagen y la huella haya sido capturada. Una vez hecha la comprobación, se llama a la función `AgregarUsuario` a través del proxy. La interfaz gráfica de este submódulo se presenta en la Figura 2.45 (a).

Para el submódulo Actualizar usuario cuya interfaz gráfica se basa en la del submódulo Agregar Usuarios, el proceso es muy similar, con la diferencia que se obtiene la información de todos a través de la función `ObtenerProfesores`. Posteriormente se lo actualiza llamando a la función `ActualizarUsuario`, ambas llamadas a través del proxy.

En el lado del servidor, para la función `AgregarUsuario`, y cualquier otra de agregación de datos, se ejecuta un algoritmo en el cual se mantienen ordenados los índices de los registros en *SQL Server*; este se puede observar en el Código 2.21 y se logra específicamente con la sentencia `DBCC CHEKIDENT`, una vez definido el índice correspondiente, se ejecuta la sentencia `INSERT`.

Una vez determinado el índice, se añaden los datos con un objeto de tipo `SqlCommand`, el cual permitirá ejecutar la consulta de SQL; seguidamente los parámetros se añaden con `AddWithValue` a excepción de la huella, como se observa en el Código 2.22.

En la función `ActualizarUsuario` se usa simplemente la sentencia `UPDATE` con los parámetros necesarios, y se ejecuta de manera similar que la anterior con la ayuda de un objeto de tipo `SqlCommand`, como se aprecia en el Código 2.23.

```
316 using (SqlDataReader reader = (new SqlCommand("select * from [dbAlfa].[dbo].[Usuarios]", cnx)).ExecuteReader())
317 {
318     //Lazo que se cumple mientras el reader permanezca leyendo datos.
319     while (reader.Read())
320     {
321         //se genera una resta para saber si los indices son subsecuentes o no
322         intAux = Convert.ToInt32(reader["idPro"].ToString()) - indAux;
323         //en caso de ser subsecuentes, se iguala el  indice al ultimo con el que se hizo la comparacion para agregar
324         //el nuevo usuario con el indice adecuado
325         if (intAux == 1)
326         {
327             indAux = Convert.ToInt32(reader["idPro"].ToString());
328         }
329         //caso contrario se usa el penultimo indice para ingresar la informacion
330         else
331         {
332             break;
333         }
334     }
335     //se genera la sentencia en la cual se reasigna el ultimo indice al obtenido en el algoritmo anterior
336     sentencia = "DBCC CHECKIDENT ('Usuarios', RESEED, " + indAux +
337                 "); Insert into [dbAlfa].[dbo].[Usuarios](cedula, nombresPro, apellidosPro, contrasena, huella," +
338                 " isAdmin, habilitado, email, imagenUrl,isProf) VALUES(@Cedula, @NombresPro, @ApellidosPro, @Contrasena,"+
339                 " @Huella, @IsAdmin, @Habilitado, @Email, @ImagenUrl, @IsProf)";
340 }
```

Código 2.21. Fragmento función AgregarUsuario

```
347 sqlWrite.Parameters.AddWithValue("@Email", profAux.Email);
348 sqlWrite.Parameters.AddWithValue("@ImagenUrl", profAux.ImagenUrl);
349 sqlWrite.Parameters.Add("@Huella", SqlDbType.VarBinary, profAux.Huella.Length).Value = profAux.Huella;
350 sqlWrite.Parameters.AddWithValue("@IsAdmin", profAux.IsAdmin);
```

Código 2.22. Inserción de parámetros al comando SQL

```
249 //se genera la consulta para actualizar los datos del profesor en base al cedula
250 string consulta = "UPDATE Usuarios SET nombresPro = @NombresPro, apellidosPro = @ApellidosP
251 //se usa la conexion a la base de datos
252 using (SqlConnection cnx = ObtenerConexion())
253 //se ejecuta el comando para insertar los nuevos valores
254 using (SqlCommand sqlWrite = new SqlCommand(consulta, cnx))
255 {
256     //se agregan los nuevos valores en base al profesor enviado como argumento de entrada
257     sqlWrite.Parameters.AddWithValue("@NombresPro", profAux.Nombres);
258     sqlWrite.Parameters.AddWithValue("@ApellidosPro", profAux.Apellidos);
```

Código 2.23. Fragmento función ActualizarUsuario

Es necesario señalar que, en conjunto a las operaciones anteriores, se crean o actualizan los perfiles de autenticación en Firebase para poder usarlos en el ingreso de la aplicación

móvil. Para ello se debe obtener el archivo de credenciales de Google de la siguiente manera:

- Primero en la consola de Firebase dentro de cualquier navegador web, habiendo ya elegido el proyecto, se da clic en el botón de configuración, y luego en Configuración del proyecto.

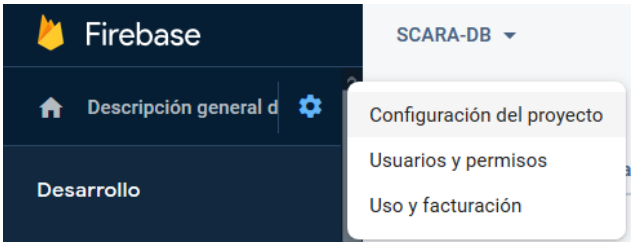


Figura 2.50. Submenú de configuración de Firebase

- Una vez ahí, se da clic en la pestaña Cuentas de servicio, seguido de cuentas de servicio de Google *Cloud Platform.*

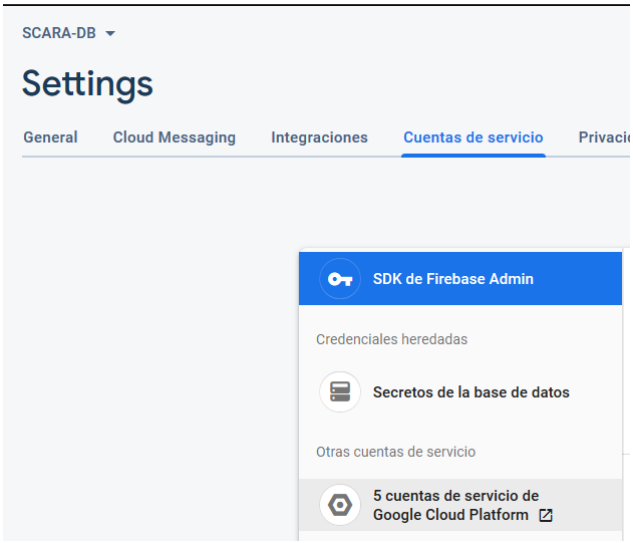


Figura 2.51. Pestaña Cuentas de servicio

- Se abrirá en una nueva pestaña del navegador web el servicio de cuentas de Google *Cloud Platform*, aquí se elige la cuenta correspondiente a *App engine default service account* (Figura 2.52).
- Se elige editar, con lo cual se habilitará la opción de Crear; al dar clic se abrirá una alerta, en la cual se elige el formato del archivo en JSON, y se crea y guarda en los archivos del sistema (Figura 2.53).

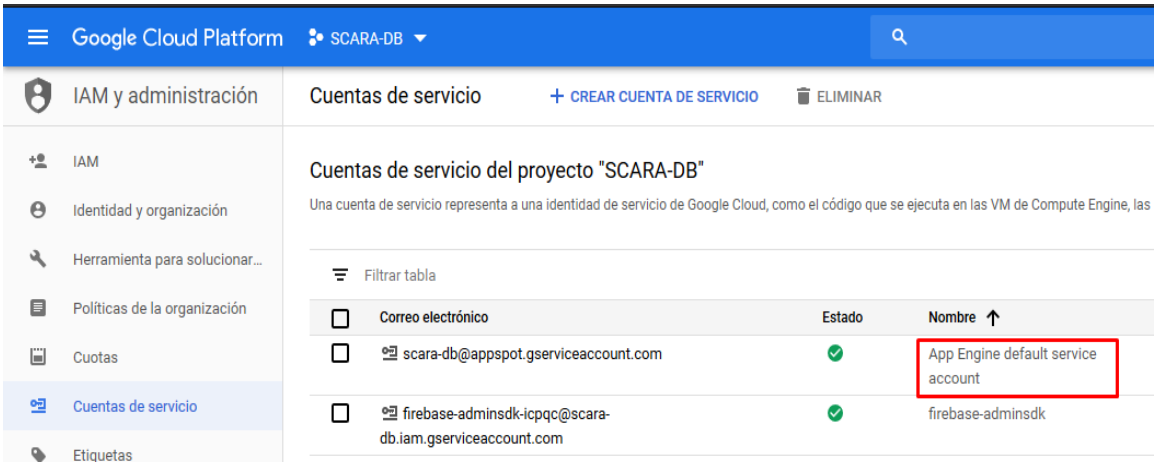


Figura 2.52. Google Cloud Platform

- Este archivo se añade a las variables de entorno con el nombre de `GOOGLE_APPLICATION_CREDENTIALS`, junto a la ruta a éste.

Es necesario también agregar las librerías `FirebaseAdmin` y `FirebaseAdmin.Auth`, las cuales permitirán usar los objetos necesarios para la autenticación del servidor y el manejo de usuarios en Firebase.

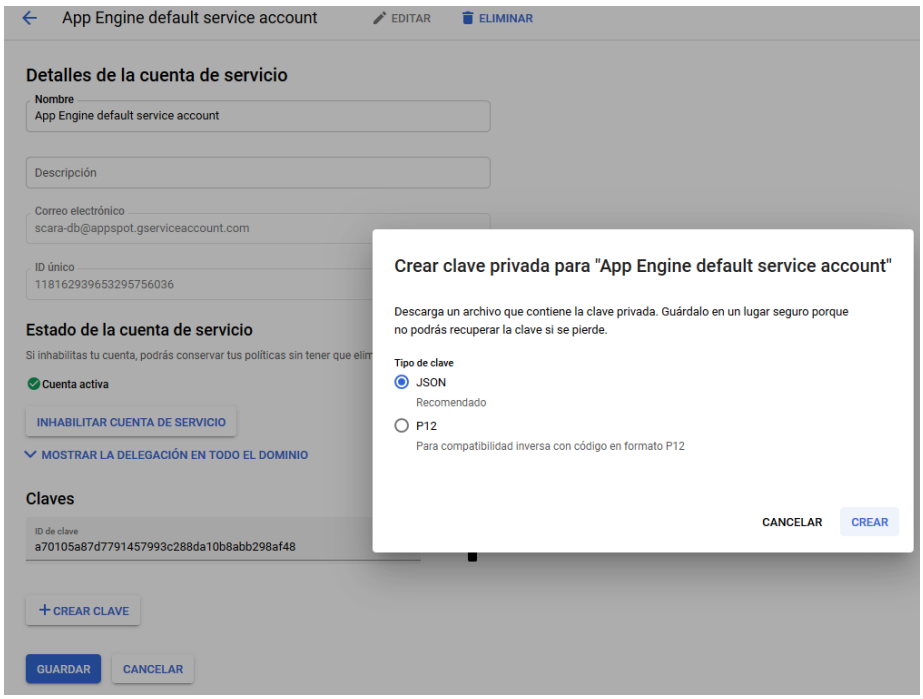


Figura 2.53. Generación de clave

A continuación, en el componente se instancia la función `IniciarFirebase` la cual hace uso del archivo en las variables de entorno para poder conectarse a Firebase. Esta función se llamará al iniciar el servidor.

```
229  public static void IniciarFirebase()
230  {
231      /// se crea una nueva aplicacion de fb en base a las credenciales del proyecto obtenido de google
232      /// las cuales ya se encuentran definidas con una variable de entorno de windows
233      FirebaseApp.Create(new AppOptions()
234      {
235          Credential = GoogleCredential.GetApplicationDefault(),
236      });
237  }
```

Código 2.24. Función IniciarFirebase

Se tiene la función `AgregarUsuarioFB`, la cual agregará un usuario para la autenticación, siempre y cuando el usuario sea un profesor; en caso que el usuario no tenga una imagen, se le asigna una por defecto. Posteriormente, se inicia una tarea asíncrona para iniciar la operación de guardado, y se usa un objeto de tipo `UserRecordArgs`, en el que se añaden los datos del usuario, y usando `DefaultInstance.CreateUserAsync`, se crea el usuario en Firebase (Código 2.25).

La función `ActualizarUsuarioFB` funciona de manera similar a `AgregarUsuarioFB`, con la única diferencia que se usa la función `UpdateUserAsync`, para realizar la actualización en Firebase (Código 2.26).

2.3.5.3 Entregable

El entregable para el presente *sprint* consiste en los submódulos Agregar Usuario y Actualizar Usuario completamente funcionales y con sus respectivas validaciones.

```
904  await Task.Run(async () =>
905  {
906      //se crea un nuevo objeto del tipo UserRecordArgs el cual contendra los datos del usuario
907      UserRecordArgs args = new UserRecordArgs()
908      {
909          Email = p.Email,
910          EmailVerified = true,
911          Password = p.Contrasena,
912          DisplayName = nombreAux,
913          PhotoUrl = p.ImagenUrl,
914          Disabled = false,
915          Uid = p.Cedula
916      };
917
918      try
919      {
920          //se crea el usuario en fb
921          UserRecord userRecord = await FirebaseAuth.DefaultInstance.CreateUserAsync(args);
922          //se imprime el nombre del usuario agregado
923          Console.WriteLine($"Successfully created new user: {userRecord.DisplayName}");
924      }
```

Código 2.25. Fragmento función AgregarUsuarioFB

```
956     UserRecordArgs args = new UserRecordArgs()
957         {
958             Email = p.Email,
959             EmailVerified = true,
960             Password = p.Contrasena,
961             DisplayName = nombreAux,
962             PhotoUrl = p.ImagenUrl,
963             Disabled = false,
964             Uid = p.Cedula
965         };
966 
967         try
968         {
969         //se crea el usuario en fb
970         UserRecord userRecord = await FirebaseAuth.DefaultInstance.UpdateUserAsync(args);
```

Código 2.26. Fragmento función ActualizarUsuarioFB

2.3.6 SPRINT 6

En este *sprint* se implementará la función de actualizar la base de datos mediante un archivo de Excel, es decir el módulo Actualización BD.

2.3.6.1 Diseño

El proceso para realizar la actualización de la base de datos a través de la aplicación de administración se muestra en el diagrama de secuencia de la Figura 2.54.

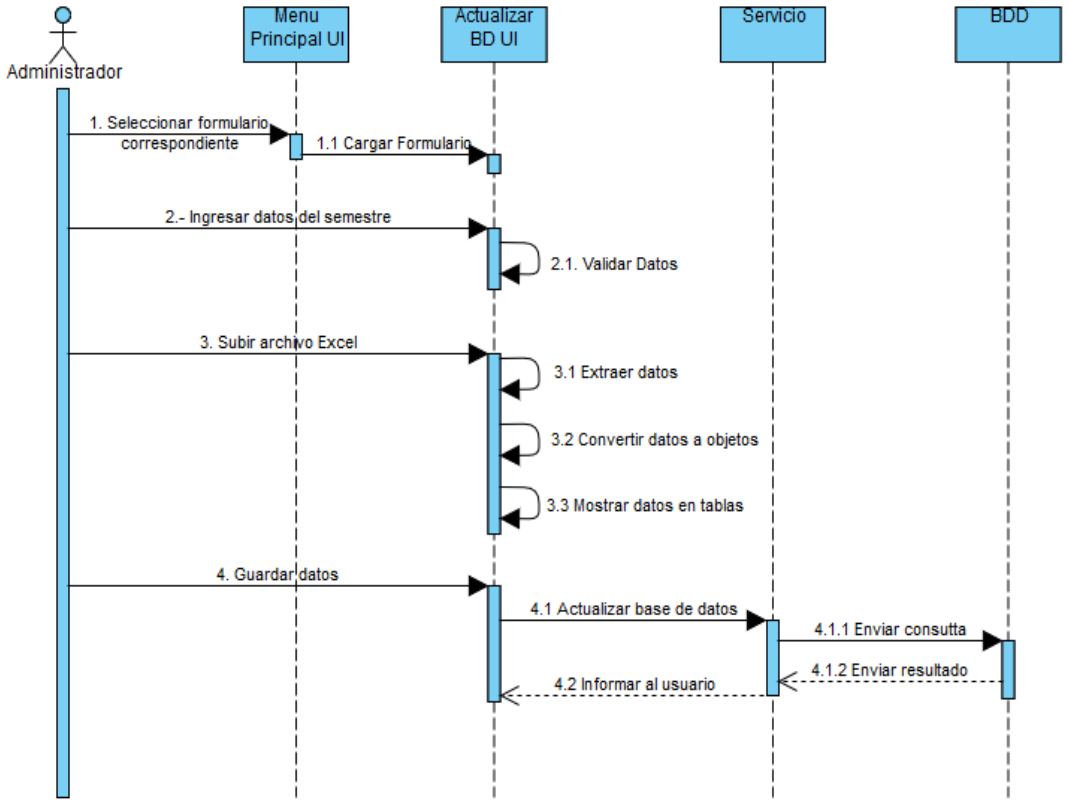


Figura 2.54. Diagrama de secuencia del submódulo ActualizarBD

2.3.6.2 Implementación

El documento en Excel deberá tener un formato específico, para que éste pueda ser leído correctamente por el sistema. Antes de poder subir el documento, se validan los campos correspondientes al semestre al que pertenecen estos horarios; también se verifica que la fecha de inicio del semestre sea menor a la fecha fin. Adicionalmente, esta última debe corresponder a la fecha de cierre semestral del SAEw.

Para recuperar los datos del documento, primero se le solicita al usuario elegir un documento de Excel a través de un cuadro de diálogo, una vez elegido, se recupera la ruta absoluta de éste. Luego se instancia una conexión al archivo en base a la ruta obtenida anteriormente para extraer los datos con la librería `System.Data.OleDb`.

Se instancian cuatro objetos: Uno de tipo `DataTable` que será donde se guardan los datos de manera temporal; uno de tipo `OleDbConnection` para establecer la conexión con el archivo; uno de tipo `OleDbCommand` para ejecutar la consulta; y, uno de tipo `OleDbDataAdapter` para adaptar los datos obtenidos y guardarlos en el `DataTable`.

Con un lazo `for` se recorre de hoja en hoja, primero obteniendo el nombre de la hoja, luego se ejecuta el comando que recupera todas las filas que tengan aulas correspondientes al edificio de Química/Eléctrica, y algunas aulas válidas del edificio de Eléctrica. Seguidamente se llena el objeto `DataTable`; a continuación, con las funciones `ObtenerProfesores`, `ObtenerMaterias` y `ObtenerAulas` se consiguen los objetos respectivos y se los agrega a las listas auxiliares correspondientes.

Como un ejemplo, se muestra en el Código 2.27 un fragmento de la función `ObtenerMaterias`, la cual muestra cómo se realiza la validación para este objeto.

```
public void ObtenerMaterias(DataTable table)
{
    //booleano que permite saber si la materia es valida para ser ingresada a la bd
    bool matValida = true;
    //lazo que recorre las filas del datatable
    foreach (DataRow row in table.Rows)
    {
        //si  columna correspondiente al nombre de la materia es diferente de vacio
        if (row[table.Columns[0]].ToString() != "")
        {
            //lazo que recorre la lista de materias obtenidas del excel
            foreach (Materia m in _matsExcel)
            {
                //si el nombre de la materia coincide con el nombre de la materia de la fila de la iteracion
                if (m.Nombre.ToUpper() == row[table.Columns[0]].ToString().Substring(0, row[table.Columns[0]].ToString().IndexOf("(")).
                {
                    //la materia ya se encuentra en la lista por lo tanto no es valida
                    matValida = false;
                    //sale del lazo
                    break;
                }
            }
```

Código 2.27. Fragmento de función ObtenerMaterias

Cada una de estas funciones realiza la validación de no obtener objetos repetidos, comparando con los existentes en las listas correspondientes, y finalmente se muestran en las tablas de la interfaz gráfica. Esta operación se muestra en el Código 2.28.

Finalmente, para agregar la información a las bases de datos se usan las funciones `AgregarLoteProfesores`, `AgregarLoteMaterias` y `AgregarLoteAulas` a través del proxy y éstas además enviarán una respuesta de cuántos objetos de cada categoría fueron agregados.

Adicionalmente estas funciones comparan lo obtenido del archivo Excel con lo ya existente en el servidor, para de igual manera, evitar información duplicada en las bases de datos. Un ejemplo se muestra en el Código 2.29.

Los datos obtenidos de profesores y aulas se agregan a SQL Server realizando el mismo proceso explicado en el *sprint* anterior. Cabe recalcar que para el caso de aulas, se realiza también la agregación de grupos y horarios, ya que éstos son contenidos dentro de las mismas.

```
245 for (int i = 0; i <= 3; i++)
246 {
247     //se limpia el datatable
248     dt.Clear();
249     //se abre la conexion al archivo
250     connExcel.Open();
251     //genera un esquema para mostrar los datos tal como se muestran en el gui del excel
252     DataTable dtEsquema;
253     dtEsquema = connExcel.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, null);
254     //se recupera el nombre de las hojas de calculo del excel
255     nombreHoja = dtEsquema.Rows[i]["TABLE_NAME"].ToString();
256     //se cierra la conexion con el excel
257     connExcel.Close();
258     //se abre la conexion al archivo
259     connExcel.Open();
260     //se asigna la consulta al comando que se ejecutara para obtener la informacion
261     cmdExcel.CommandText = "SELECT Asignatura,Hor,Paral,Aula,Prof,Cupo,Lun,Mar,Mié,Jue,Vie,Sáb From [" + nombreHoja +
262                            "] where(Aula like 'Q/E[0-9][0]%' or Aula = 'ELEE003' or Aula = 'ELEE004' or Aula = 'ELEE206') "+
263                            "and Asignatura not like 'SEMINARIO%' and Prof <> '' and Prof <> 'SIN PROFESOR'  and Cupo <> '' "+
264                            "and Cupo <> '0'";
265     //cmdExcel.CommandText = "SELECT * From [" + nombreHoja + "] where Cupo <> 0";
266     //se le asigna el comando
267     odaExcel.SelectCommand = cmdExcel;
268     //ejecuta el comando y llena el datatable con la informacion de la hoja especificada
269     odaExcel.Fill(dt);
270     //se cierra la conexion con el excel
271     connExcel.Close();
272     //la informacion obtenida se muestra en la respectiva tabpage del tabcontrol
273     MostrarDatos((DataGridView)(tabDatosCont.TabPages[i].Controls[0]), dt);
274     //se envia lo mostrado en los datatable a las listas para ser enviadas a la bd
275     //DisplayData(dt);
276
277     ObtenerProfesores(dt);
278     ObtenerMaterias(dt);
279     ObtenerAulas(dt);
280 }
```

Código 2.28. Algoritmo para recuperación de datos del archivo Excel

Adicionalmente, las aulas son agregadas a Firebase, para lo cual mediante las librerías `FireSharp.Config`, `FireSharp.Interfaces` y `FireSharp.Response`, se interactúa con la *Realtime Database*.

Para iniciar el proceso de almacenamiento en Firebase, se genera un objeto de tipo `IFirebaseClient` con la función `GenerateClientFB`, este permite la conexión con la base de datos y así poder realizar operaciones en la misma. Para ello esta función requiere el enlace de la base de datos y el *token* de autenticación. Esta función se muestra en el Código 2.30.

Seguidamente, se usa la función `AgregarAulasFB`, la misma que se llama una vez se hayan agregado los datos en SQL *server*. A la vez esta función llama a `GenerateClientFB`, con la cual se inserta cada aula con su información, y además se definen los nombres de éstas como sus llaves. La función se muestra en el Código 2.31.

```
public int AgregarLoteMaterias(string cliente, List<Materia> matExcel)
{
    //lista de materias que contendra las materias obtenidas desde la bd
    List<Materia> matsAux = ObtenerMaterias(cliente);
    //variable integer que nos mostrara si se agrego correctamente
    int exito = 0;

    int totalMateriasAgregadas = 0;
    //se comprueba que exista conexion con el servidor
    if (ComprobarConexionClienteAdmin(cliente))
    {
        // se genera la consulta por default en caso de que la base este vacia
        string sentencia = "Insert into [dbAlfa].[dbo].[Materias](nombreMat, creditos) VALUES(@NombreMat, @Creditos)";
        //booleano que permitira conocer si la materia ya se encuentra en la bd
        bool matValida = true;
        //entero que permitira conocer si se agregaron materias nuevas a la bd
        int indexMat = 0;
        //Se invoca al metodo ObtenerConexion y esta conexion es almacenada en la variable con del tipo SqlConnection.
        using (SqlConnection cnx = ObtenerConexion())
        {
            //lazo que recorre la lista de materias obtenidas del excel
            foreach (Materia m in matExcel)
            {
                //lazo que recorre la lista de materias obtenidas de la bd
                foreach (Materia mAux in matsAux)
                {
                    //si las materias de la iteracion coinciden
                    if (mAux.Nombre == m.Nombre)
                    {
                        //se incrementa el contador
                        indexMat++;
                        //se setea el booleano en false
                        matValida = false;
                        //se sale del lazo
                        break;
                    }
                }
```

Código 2.29. Fragmento de la función AgregarLoteMaterias.

```
public static IFirebaseClient GenerateClientFB()
{
    //se crea una configuracion en base a un token y a la direccion de la base
    IFirebaseConfig config = new FirebaseConfig
    {
        AuthSecret = "[redacted]",
        BasePath = "https://[redacted]",
    };
    //genero el cliente en base a la configuracion definida anteriormente
    IFirebaseClient client = new FireSharp.FirebaseClient(config);
    //retorna el clientete
    return client;
}
```

Código 2.30. Función GenerateClientFB

```
1410  public void AgregarAulasFB(List<Aula> aulas)
1411  {
1412
1413      /// genero el cliente en base a la configuracion definida anteriormente
1414      IFirebaseClient client = GenerateClientFB();
1415      /// se recorre la lista de aulas obtenidas de la base
1416      foreach (Aula a in aulas)
1417      {
1418          /// se guarda el aula en el lugar correspondiente y se asigna como llave el nombre del aula
1419          /// reemplazando el "/" por el "*" para evitar conflictos con el json
1420          SetResponse response = client.Set("Aulas/" + a.NombreAu.Replace("/", "*"), a);
1421      }
1422  }
```

Código 2.31. Función AgregarAulasFB

2.3.6.3 Entregable

El entregable de este *sprint* consiste en el submódulo ActualizarBD completamente funcional y con sus respectivas validaciones.

2.3.7 SPRINT 7

En este *sprint*, se crearán los métodos y algoritmos necesarios para la implementación de las operaciones relacionadas a dispositivos, mismas que serán abarcadas dentro del módulo Dispositivos en los submódulos Agregar Dispositivos y Manejo Dispositivos.

2.3.7.1 Diseño

Los procesos para la agregación de dispositivos al sistema y el manejo de los mismos a través de la aplicación de administración, se presentan en los diagramas de secuencia de las Figura 2.55 yFigura 2.56.

2.3.7.2 Implementación

Para la interacción con los dispositivos, se monta un sistema de comunicación TCP/IP entre el servicio y el servidor. Para ello se crea la clase `ManejadorComandos`, de la cual se instancia un objeto dentro del método `EjecutarComando`, el mismo que se ejecuta en un hilo diferente, y se mantiene a la espera de una trama con un comando en el puerto 8082 (Código 2.32).

La trama de comando tiene tres campos: El primero contiene un número, el cual define el tipo de comando a ejecutar, dichos tipos se pueden observar en la Tabla 2.15; el segundo muestra qué usuario lo envía; y el tercero envía algún dato adicional dependiendo del comando. También se tiene una trama de respuesta la cual consta de dos campos: El primero contiene el resultado de si la operación fue exitosa, y el segundo consiste en datos extra dependiendo del comando ejecutado. De manera simplificada se muestran estas tramas en la Figura 2.57.

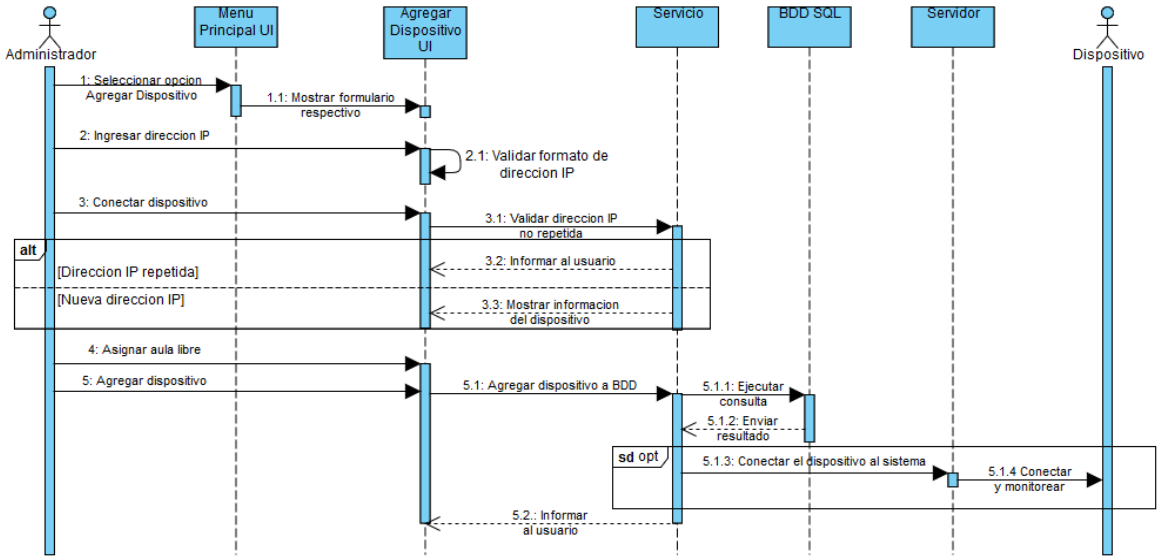


Figura 2.55. Diagrama de secuencia del submódulo Agregar Dispositivo

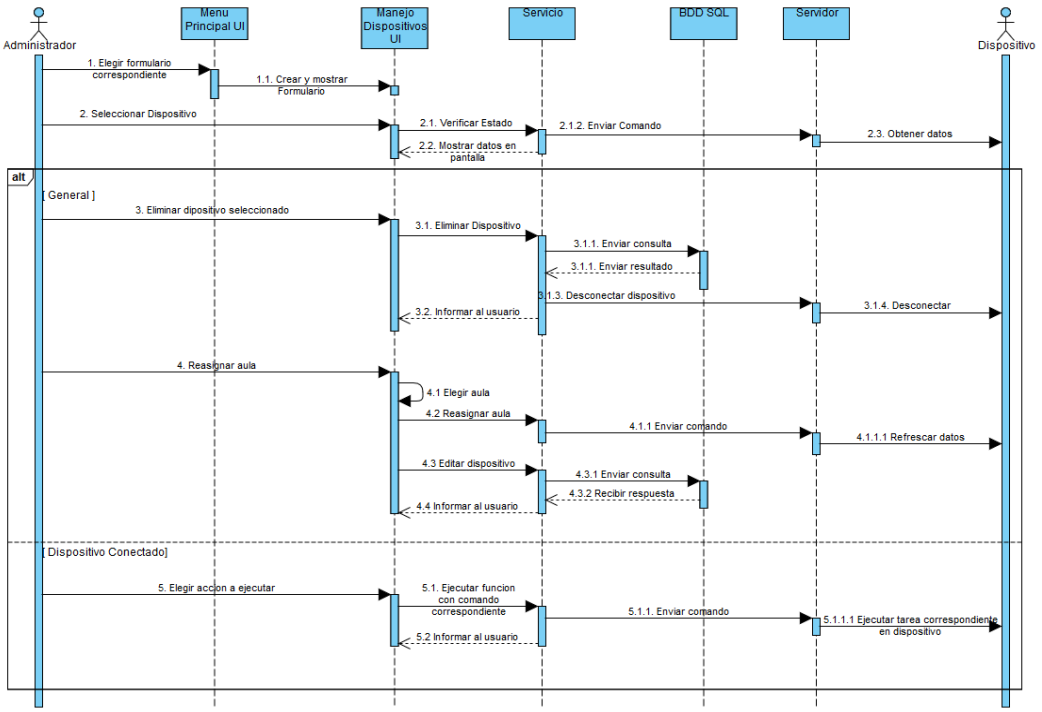


Figura 2.56. Diagrama de secuencia del submódulo Manejo Dispositivos

```
358     public static void EjecutarComando()
359     {
360         /// Creacion de la variable escuchador del tipo TcpListener, TcpListener permite escuchar las conexione
361         /// del cliente TCP
362         TcpListener escuchador = new TcpListener(8082);
363         /// Empieza a escuchar las peticiones de conexiones del cliente
364         escuchador.Start();
365         /// Lazo que permite que el servidorTCP no se cierre y siga escuchando sin importar que el cliente cier
366         /// la conexion
367         while (true)
368         {
369             /// Se crea un variable cliente del tipo TcpClient, se acepta una conexion con el cliente mediante
370             TcpClient cliente = escuchador.AcceptTcpClient();
371             /// Se muestra por consola el mensaje "Cliente se ha conectado"
372             Console.WriteLine("Cliente se ha conectado");
373             /// Se crea una variable gestor del tipo ManejadorComandos
374             ManejadorComandos gestor = new ManejadorComandos();
375             /// Se invoca al metodo ManipuladorCliente mediante gestor (para ejecutar el hilo de los procesos)
376             gestor.ManipuladorCliente(cliente, dispositivos, estConex, estHab);
377         }
378         /// Cierra el escuchador
379         escuchador.Stop();
380     }
```

Código 2.32. Función EjecutarComando

Figura 2.57. Tramas Comando/Respuesta

Una vez se haya recibido la trama, dentro del objeto `ManejadorComandos` se inicia un hilo en base a la función `ManipuladorComunicaciones`. Esta función descompondrá la trama recibida y enviará la trama resultante. Para descomponer la trama, primero recibe el flujo de datos y obtiene sus bytes, para luego decodificarlos en ASCII.

Seguidamente se obtiene cada uno de los campos de la trama enviada en base al carácter "@" y se define la respuesta que está predeterminada como no exitosa. Esto se puede observar en el Código 2.33.

A continuación se tiene una serie de sentencias `if`, las cuales permiten seleccionar en base al campo Tipo, el comando a ejecutarse, como se muestra en el Código 2.34.

Una vez ejecutado el comando, se devuelve la respuesta transformándola de nuevo a bytes y retornándola al servicio para que pueda enviar la respuesta al cliente. Adicionalmente, se cierra el cliente TCP/IP entre el servicio y el servidor. Esta operación se muestra en el Código 2.35.

```
63  /// Creacion de la variable flujo del tipo NetworkStream
64  /// NetworkStream permite enviar o recibir datos por parte del cliente
65  NetworkStream flujo = cliente.GetStream();
66
67  /// Creacion de la variable bufferRx del tipo byte con una tamaño de 1024
68  byte[] bufferRx = new byte[1024];
69
70  /// Lee el flujo de datos recibidos del cliente y los almacena en el bufferRx
71  flujo.Read(bufferRx, 0, bufferRx.Length);
72
73  /// Convierte los datos recibidos del tipo byte a una cadena de string
74  string dato = Encoding.ASCII.GetString(bufferRx);
75
76  /// Corta la cadena recibida al tamaño actual del mensaje recibido, ya que el tamaño del bufferRx es 1024
77  dato = dato.Substring(0, dato.IndexOf("\0"));
78
79  /// Se separa los datos recibido mediante un Split, un split separa cadenas cada que encuentre el caracter '@'
80  /// y estos datos son almacenados en un vector de string.
81  string[] datosCliente = dato.Split('@');
82
83  /// string que devolvera el resultao de la operacion
84  string respuesta = "false";
```

Código 2.33. Obtención y descomposición de trama

```
96  if (datosCliente[0] == "0")
97  {
```

Código 2.34. Ejemplo selección comando

```
680 /// la variable respuesta es convertida de string a byte para poder ser enviada al cliente
681 byte[] bufferTx = Encoding.ASCII.GetBytes(respuesta);
682
683 /// Escribe sobre el flujo de datos del cliente
684 flujo.Write(bufferTx, 0, bufferTx.Length);
685
686 /// Vacia los datos del buffer
687 flujo.Flush();
688
689 /// se cierra la conexion con el cliente
690 cliente.Close();
```

Código 2.35. Proceso envío de respuesta

Una vez definido el modo de ejecución de tareas con los dispositivos, se definen las funciones a utilizar. Al agregar un dispositivo, se debe establecer una conexión TCP/IP con el mismo, para ello se tiene un `RichTextBox`, en el cual, con la ayuda de una máscara o patrón predefinido, se ingresa y valida la dirección IP del dispositivo a agregar.

Seguidamente, se comprueba que esta dirección IP no esté siendo utilizada por algún otro dispositivo. A continuación se intenta establecer conexión con el dispositivo y obtener sus datos con la función `RecuperarDatosDisp`, que hace uso de la función `EnviarComando`, en la cual se envía la trama de comando, en este caso, con valor de tipo “0”, como se muestra en el Código 2.36.

```
3314    //se envia la trama para ejecutar el comando respectivo
3315    string resultado = EnviarComando("0" + "@" + cliente + "@" + dirIP);
```

Código 2.36. Envío de comando para recuperación de datos del dispositivo

En la función `EnviarComando` se define un cliente TCP que permitirá la conexión al servidor desde el servicio, seguido de la dirección IP y el puerto a conectarse, que serán la dirección IP del servidor y el puerto 8082. Una vez establecida la conexión, se establece el flujo por el que se intercambiarán datos, luego se envía la trama convirtiéndola a bytes y se espera un tiempo por la respuesta haciendo el proceso inverso, para finalmente cerrar la conexión (Código 2.37).

Retomando la operación de la función `RecuperarDatosDisp`, una vez enviado el comando, se ejecutan una serie de funciones propias del dispositivo a través de la propiedad `Manager`, que es la clase que las contiene, como se observa en el Código 2.38.

Una vez obtenidos los datos del dispositivo desde el servidor, se guardan en un objeto de tipo Dispositivo auxiliar, y se envía éste al cliente, donde se muestran estos datos en pantalla. Seguidamente se selecciona un aula a asignar al dispositivo, y para añadirlo a la base de datos en SQL se llama a la función `AgregarDispositivo`. La interfaz gráfica de este submódulo corresponde a la Figura 2.46 (b).

Esta función además de almacenar sus datos, llama a la función `ConectarDispAgregado`, la cual ejecuta el comando de tipo “13” que permite mantener la conexión con el dispositivo recientemente agregado para que pueda estar operativo.

En cuanto al submódulo Manejo de Dispositivos, cuya interfaz gráfica se presenta en la Figura 2.47 (a), se tienen varias funciones las cuales se listan a continuación.

- Reasignación de aula

En esta función, se elige un aula del listado de aulas disponibles presente en pantalla. Una vez definida, se usa la función `ReasignarAulaDisp` para ejecutar el comando de tipo “15”, el cual recorre los dispositivos hasta encontrar el indicado en el comando.

Una vez obtenido, se recupera el Aula de la base en SQL con la función `ObtenerAula` y refresca los datos del objeto `DispositivoServidor` correspondiente. La operación se muestra en el Código 2.39.

```
3251  public string EnviarComando(string trama)
3252  {
3253      String resultado = "";
3254      //se instancia un objeto de tipo TcpClient
3255      TcpClient remoto = new TcpClient();
3256
3257      try
3258      {
3259          //Se crea un IPEndPoint con lo datos del servidor y se conecta
3260          IPEndPoint ipRemoto = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 8082);
3261          remoto.Connect(ipRemoto);
3262
3263          //se obtiene el buffer en el que se enviaran o recibiran los datos
3264          NetworkStream flujo = remoto.GetStream();
3265
3266          //Se envia los campos requeridos separados por "@", el primer campo funciona como un comando
3267          //para este caso "0", le indica al servidor que se requiere un obtener los datos de un dispos
3268          byte[] bufferTx = Encoding.ASCII.GetBytes(trama);
3269          //se escriben los bytes en el buffer
3270          flujo.Write(bufferTx, 0, bufferTx.Length);
3271          //se envian al servidor
3272          flujo.Flush();
3273          Thread.Sleep(500);
3274
3275          //se crea un arreglo de btyes para obtener los datos del servidor
3276          byte[] bufferRx = new byte[1024];
3277          //se lee los datos del buffer recibido
3278          flujo.Read(bufferRx, 0, bufferRx.Length);
3279
3280          //Se obtiene la respuesta del servidor,
3281          resultado = Encoding.ASCII.GetString(bufferRx);
3282
3283          // se elimina el excedente del buffer
3284          resultado = resultado.Substring(0, resultado.IndexOf("\0"));
3285
3286          //se cierra la conexion
3287          remoto.Close();
3288      }
```

Código 2.37. Función EnviarComando

```
99   DispositivoServidor dispAux = new DispositivoServidor(datosCliente[2]);
100
101  /// se valida que esta ip no este siendo utilizada por otro dispositivo
102  if (ValidarIP(datosCliente[2]))
103  {
104      /// se llama a la funcion para conectar al nuevo dispositivo
105      if (dispAux.Manager.Connect_Net(datosCliente[2],4370))
106      {
107
108          /// se setea la respuesta de la operacion en el string auxiliar
109          respuesta = "true" + "@" + datosCliente[2];
110          /// string que almacenara y concatenara los datos recuperados del dispositivo
111          string sValue = "";
112          if (dispAux.Manager.GetDeviceMAC(1, ref sValue))
113          {
114              respuesta = respuesta + "@" + sValue;
115              dispAux.DirMAC = sValue;
116          }
117          if (dispAux.Manager.GetProductCode(1, out sValue))
118          {
119              respuesta = respuesta + "@" + sValue;
120              dispAux.Codigo = sValue;
121          }
122          if (dispAux.Manager.GetVendor(ref sValue))
123          {
124              respuesta = respuesta + "@" + sValue;
125              dispAux.Fabricante = sValue;
126          }
127          if (dispAux.Manager.GetPlatform(1, ref sValue))
128          {
129              respuesta = respuesta + "@" + sValue;
130              dispAux.Tipo = sValue;
131          }
132          if (dispAux.Manager.GetSerialNumber(1, out sValue))
133          {
134              respuesta = respuesta + "@" + sValue;
135              dispAux.Serial = sValue;
136          }
137      }
138  }
139  /// se cierra la conexion con el dispositivo
140  dispAux.Manager.Disconnect();
```

Código 2.38. Proceso de recuperación de datos del dispositivo

```
625 /// lazo que recorre la lista de dispositivos obtenida de la bd
626 foreach (DispositivoServidor d in dispositivos)
627 {
628     /// si la serial del dispositivo de la iteracion coincide con el dato recibido en la trama
629     if (d.Serial == datosCliente[2])
630     {
631         /// se modifica el aula del dispositivo
632         d.Aula = Componente.ObtenerAula(datosCliente[3]);
633         /// la informacion en el dispositivo debe ser actualizada
634         d.Manager.RefreshData(1);
635         /// se setea el resultado de la operacion en true
636         respuesta = "true";
637         /// se sale del lazo
638         break;
639     }
640 }
```

Código 2.39. Proceso de reasignación de aula

- Verificación de estado

En esta función, se selecciona un dispositivo del listado de dispositivos presente en pantalla, cuya dirección IP será enviada en el comando. Posteriormente, se usará la función `VerificarEstadoDisp` para ejecutar el comando de tipo “4”, el cual recorrerá los dispositivos hasta encontrar el indicado en el comando.

Si el dispositivo se encuentra conectado, se recuperará los datos del mismo, mediante la función `ComprobarUsuarios` que obtendrá la cantidad de administradores y la cantidad total de usuarios en el dispositivo, para mostrarlos en pantalla.

Caso contrario, se indicará que el dispositivo con dicha IP se encuentra desconectado, y se inhabilitarán ciertas funciones en el módulo puesto que para ejecutarlas se necesita que el dispositivo se encuentre enlazado (Código 2.40).

- Configuración de hora y fecha

En esta función, se selecciona un dispositivo del listado de dispositivos presente en pantalla. Se llama a la función `ConfigurarHoraDisp` para ejecutar el comando tipo “1”, en el cual se recorre los dispositivos comparando las direcciones IP hasta que exista coincidencia entre ellas.

Se recupera la hora actual en variables que posteriormente serán utilizadas como parámetros de entrada en la función `SetDeviceTime`, mediante la cual se realizará la configuración del dispositivo, y se procederá a refrescar la información en el mismo (Código 2.41).

```
/// entero que nos permitira devolver el tipo de error en caso de suscitarse
int idwErrorCode = 0;
/// entero que guarda el indice del dispositivo de la iteracion
int aux = 0;
/// lazo que recorre la lista de dispositivos obtenida de la bd
foreach (DispositivoServidor d in dispositivos)
{
    /// si la ip del dispositivo de la iteracion coincide con la recibida en la trama
    if (d.DirIP == datosCliente[2])
    {
        /// si el estado de conexion del dispositivo de la iteracion esta en true
        if (estConex[aux])
        {
            /// se setea el resultado de la operacion en true
            respuesta = "true";
            /// lazo que recorre los datos que se obtienen al ejecutar la funcion
            /// comprobar usuarios
            foreach (int i in d.ComprobarUsuarios())
            {
                /// se concatena a la respuesta enviar
                respuesta = respuesta + "@" + i;
            }
        }
        /// se sale del lazo
        break;
    }
    /// se incrementa el contador
    aux++;
}
```

Código 2.40. Proceso para comprobar el estado de un dispositivo

```
/// entero que nos permitira devolver el tipo de error en caso de suscitarse
int idwErrorCode = 0;
/// lazo que recorre la lista de dispositivos
foreach (DispositivoServidor d in dispositivos)
{
    /// si la ip del dispositivo de la iteracion coincide con la recibida en la trama
    if (d.DirIP == datosCliente[2])
    {
        /// se setean las variables a configurar en el dispositivo
        int idwYear = DateTime.Now.Year;
        int idwMonth = DateTime.Now.Month;
        int idwDay = DateTime.Now.Day;
        int idwHour = DateTime.Now.Hour;
        int idwMinute = DateTime.Now.Minute;
        int idwSecond = DateTime.Now.Second;
        /// se llama a la funcion para configurar la hora y fecha en el dispositivo
        if (d.Manager.SetDeviceTime2(1, idwYear, idwMonth, idwDay, idwHour, idwMinute, idwSecond))
        {
            /// la informacion en el dispositivo debe ser refrescada
            d.Manager.RefreshData(1);
            /// se setea la respuesta de la operacion en true
            respuesta = "true";
        }
        /// caso contrario
        else
        {
            /// se obtiene el error
            d.Manager.GetLastError(ref idwErrorCode);
        }

        break;
    }
}
```

Código 2.41. Algoritmo para modificar hora y fecha del dispositivo

- Modificar dirección IP

En esta función, se selecciona un dispositivo del listado de dispositivos presente en pantalla; al presionar el botón aparece un cuadro de diálogo en el cual se solicita la nueva dirección IP, y el sistema corrobora que esta dirección IP sea válida.

Posteriormente se llamará a la función `ModificarIpDisp` para ejecutar el comando tipo "2", en el cual se recorre la lista de dispositivos obtenida desde la base de datos hasta encontrar coincidencia con la dirección IP del dispositivo seleccionado, se configurará la nueva dirección IP, se refrescará la información en el dispositivo y éste será reiniciado. Si la operación no fue realizada con éxito, se obtiene el mensaje de error (Código 2.42).

- Desbloquear cerradura electromagnética

En esta función, se selecciona un dispositivo del listado de dispositivos, el cual corresponde a un aula como se mostrará en pantalla; al presionar el botón aparecerá un cuadro de diálogo en el cual se solicita el tiempo en segundos que la cerradura permanecerá desbloqueada.

Posteriormente se llama a la función `DesbloquearCerraduraDisp` para ejecutar el comando tipo "3", en el cual se busca el dispositivo seleccionado, y mediante la función `ACUnlock` se desbloqueará la cerradura enviando como parámetro de entrada el tiempo ingresado (Código 2.43).

- Sincronizar con la base de datos

Esta función es activada después de verificar el estado de un dispositivo en particular y generalmente será utilizada al integrar un nuevo dispositivo al sistema para que éste se encuentre con sus datos sincronizados. Para ello, se selecciona un dispositivo del listado de dispositivos presente en pantalla.

Posteriormente se llama a la función `ConteoUsuarios` que devuelve el número de usuarios que contiene el dispositivo, para que éste sea comparado con el número de usuarios presente en la base de datos y validar la necesidad de la sincronización.

En caso de ser necesaria, se llama a la función `SincronizarDisp` para ejecutar el comando tipo "5", en el cual se busca el dispositivo seleccionado y se procede a eliminar la información almacenada en el mismo, y a insertar todos los usuarios presentes en la base de datos (Código 2.44).

Por otro lado, si no es necesaria esta acción, se informará al usuario que el dispositivo se encuentra sincronizado con la base datos mediante una ventana emergente.

```
185 /// entero que nos permitira devolver el tipo de error en caso de suscitarse
186 int idwErrorCode = 0;
187
188 /// lazo que recorre la lista de dispositivos obtenida de la bd
189 foreach (DispositivoServidor d in dispositivos)
190 {
191     /// si la serial del dispositivo de la iteracion coincide con el dato recibido en la trama
192     if (d.Serial == datosCliente[3])
193     {
194         /// se setea la nueva ip del dispositivo
195         if (d.Manager.SetDeviceIP(1, datosCliente[2]))
196         {
197             /// la informacion en el dispositivo debe ser refrescada
198             d.Manager.RefreshData(1);
199             /// el dispositivo debe ser reiniciado
200             d.Manager.RestartDevice(1);
201             /// se setea el resultado de la operacion en true
202             respuesta = "true";
203             /// se espera un tiempo hasta que se reinicie el dispositivo para ser reconectado al sistema
204             Thread.Sleep(5500);
205             /// se actualizan los datos del dispositivo en la lista
206             d.DirIP = datosCliente[2];
207             /// se conecta el dispositivo al sistema
208             d.ConnectTerminal();
209         }
210         /// caso contrario
211         else
212         {
213             /// se obtiene el error
214             d.Manager.GetLastError(ref idwErrorCode);
215         }
216         /// se sale del lazo
217         break;
218     }
219 }
```

Código 2.42. Algoritmo para modificar la dirección IP del dispositivo

```
226 /// entero que nos permitira devolver el tipo de error en caso de suscitarse
227 int idwErrorCode = 0;
228 /// lazo que recorre la lista de dispositivos obtenida de la bd
229 foreach (DispositivoServidor d in dispositivos)
230 {
231     /// si la ip del dispositivo de la iteracion coincide con la recibida en la trama
232     if (d.DirIP == datosCliente[2])
233     {
234         /// se llama a la funcion para desbloquear la cerradura del dispositivo
235         if (d.Manager.ACUnlock(1, Convert.ToInt32(datosCliente[3])))
236         {
237             /// se setea el resultado de la operacion en true
238             respuesta = "true";
239         }
240         /// caso contrario
241         else
242         {
243             /// se obtiene el error
244             d.Manager.GetLastError(ref idwErrorCode);
245         }
246         /// sale del lazo
247         break;
248     }
249 }
```

Código 2.43. Algoritmo para desbloquear cerradura electromagnética

```
289     /// entero que nos permitira devolver el tipo de error en caso de suscitarse
290     int idwErrorCode = 0;
291     /// lazo que recorre la lista de dispositivos obtenida de la bd
292     foreach (DispositivoServidor d in dispositivos)
293     {
294         /// si la ip del dispositivo de la iteracion coincide con la recibida en la trama
295         if (d.DirIP == datosCliente[2])
296         {
297             /// OJO POR SI NO FUNCIONA
298             /// se elimina todos los usuarios del dispositivo
299             if (d.DeleteAllUserTerminal())
300             {
301                 /// se llama a la funcion para almacenar todos los profesores de la bd en
302                 /// el dispositivo
303                 if (d.SetAllUsersTerminal(Componente.ObtenerProfesores()))
304                 {
305                     /// se setea el resultado de la operacion en true
306                     respuesta = "true";
307                 }
308             }
309             /// caso contrario
310             else
311             {
312                 /// se obtiene el error
313                 d.Manager.GetLastError(ref idwErrorCode);
314             }
315             /// se sale del lazo
316             break;
317         }
318     }
```

Código 2.44. Algoritmo para sincronizar dispositivo con la base de datos

- Eliminar dispositivo

En esta función, se selecciona un dispositivo del listado de dispositivos presente en pantalla, posteriormente se llama a la función `EliminarDispositivo` que descartará el dispositivo seleccionado de la base de datos con la sentencia DELETE, como se muestra en el Código 2.45.

```
2969    //se utiliza la conexion a la base
2970    using (SqlConnection cnx = ObtenerConexion())
2971    {
2972        //se ejecuta el comando para eliminar el registro en base al serial del dispositivo
2973        using (SqlCommand sqlWrite = new SqlCommand("DELETE FROM Dispositivos WHERE serial=@Serial", cnx))
2974        {
2975            //se insertael valor del serial en la consulta
2976            sqlWrite.Parameters.AddWithValue("@Serial", serial);
2977            //se guarda el resultado de la operacion
2978            exito = sqlWrite.ExecuteNonQuery();
2979        }
2980    }
```

Código 2.45. Proceso para eliminar el dispositivo de la base de datos

Si la operación se ejecuta con éxito se llama a la función `DesconectarDisp`, misma que ejecutará el comando tipo “12” que desconecta el dispositivo del sistema (Código 2.46).

```
526     /// entero que nos permitira devolver el tipo de error en caso de suscitarse
527     int idwErrorCode = 0;
528     /// entero auxiliar que nos permitira conoces el indice del dispositivo de la iteracion
529     int indexAux = 0;
530     /// lazo que recorre la lista de dispositivos obtenida de la bd
531     foreach (DispositivoServidor d in dispositivos)
532     {
533         /// si la serial del dispositivo de la iteracion coincide con el dato recibido en la trama
534         if (d.Serial == datosCliente[2])
535         {
536             /// se desconecta el dispositivo
537             d.Manager.Disconnect();
538             /// se obtiene el error
539             d.Manager.GetLastError(ref idwErrorCode);
540             /// si el idwErrorCode es igual a 0
541             if (idwErrorCode == 0)
542             {
543                 /// se setea el resultado de la operacion en true
544                 respuesta = "true";
545                 /// se elimina el dispositivo de la lista de la clase DispositivoServidor
546                 dispositivos.RemoveAt(indexAux);
547             }
548             /// sale del lazo
549             break;
550         }
551         /// caso contrario
552         else
553         {
554             /// se incrementa el contador
555             indexAux++;
556         }
```

Código 2.46. Algoritmo para desconectar el dispositivo del sistema

- Reiniciar Dispositivo

En esta función, se selecciona un dispositivo del listado de dispositivos presente en pantalla, posteriormente se llama a la función `ReiniciarDisp` para ejecutar el comando tipo “7”, que reiniciará el dispositivo y esperará un tiempo dependiendo del modelo del dispositivo para reconectarlo al sistema (Código 2.47).

- Resetear Dispositivo

En esta función, se selecciona un dispositivo del listado de dispositivos presente en pantalla, posteriormente se llama a la función `RestablecerDisp` para ejecutar el comando tipo “9”, en el cual se busca el dispositivo seleccionado, para proceder a restablecer la información en el mismo (Código 2.48).

2.3.7.3 Entregable

El entregable del presente *sprint* consiste en el módulo Dispositivos completo, es decir sus submódulos Agregar dispositivo y Manejo de dispositivos funcionales y con sus respectivas validaciones.

```
363  int idwErrorCode = 0;
364  /// lazo que recorre la lista de dispositivos obtenida de la bd
365  foreach (DispositivoServidor d in dispositivos)
366  {
367      /// si la ip del dispositivo de la iteracion coincide con la recibida en la trama
368      if (d.DirIP == datosCliente[2])
369      {
370          /// se llama a la funcion para reiniciar al dispositivo
371          if (d.Manager.RestartDevice(1))
372          {
373              if (d.Codigo == "K50")
374              {
375                  /// si se ejecuto con exito, se espera un tiempo para volver a conectar el dispositivo
376                  Thread.Sleep(20500);
377              }
378              else
379              {
380                  /// si se ejecuto con exito, se espera un tiempo para volver a conectar el dispositivo
381                  Thread.Sleep(5500);
382              }
383              /// se llama a la funcion para conectar al dispositivo
384              if (d.ConnectTerminal())
385              {
386                  /// si se ejecuto con exito
387                  /// se setea el resultado de la operacion en true
388                  respuesta = "true";
389              }
390          }
391          /// caso contrario
392          else
393          {
394              /// se obtiene el error
395              d.Manager.GetLastError(ref idwErrorCode);
396          }
397          /// se sale del lazo
398          break;
399      }
400  }
```

Código 2.47. Algoritmo para reiniciar un dispositivo

```
430  /// entero que nos permitira devolver el tipo de error en caso de suscitarse
431  int idwErrorCode = 0;
432  /// lazo que recorre la lista de dispositivos obtenida de la bd
433  foreach (DispositivoServidor d in dispositivos)
434  {
435      /// si la ip del dispositivo de la iteracion coincide con la recibida en la trama
436      if (d.DirIP == datosCliente[2])
437      {
438          /// se llama a la funcion para eliminar a todos los usuarios del dispositivo
439          if (d.DeleteAllUserTerminal())
440          {
441              /// si se ejecuto con exito
442              /// se setea el resultado de la operacion en true
443              respuesta = "true";
444          }
445          /// caso contrario
446          else
447          {
448              /// se obtiene el error
449              d.Manager.GetLastError(ref idwErrorCode);
450          }
451          /// se sale del lazo
452          break;
453      }
454  }
```

Código 2.48. Algoritmo para restablecer un dispositivo

Tabla 2.15. Tipos de comando a enviarse en la trama de comandos

Tipo	Comando
0	Conectar a terminal fuera del sistema
1	Configurar hora y fecha del terminal
2	Configurar dirección IP del terminal
3	Desbloquear cerradura
4	Obtener estado del terminal
5	Sincronizar terminal con la base de datos
7	Reiniciar dispositivo
9	Restablecer datos en terminal
10	Agregar nuevo usuario a todos los terminales conectados
11	Actualizar un usuario en todos los terminales conectados
12	Desconectar terminal
13	Conectar un nuevo terminal al sistema
14	Actualizar horario en terminal
15	Reasignar un horario a un terminal

2.3.8 SPRINT 8

En este *sprint* se implementará el submódulo Mostrar Aulas, en el cual el usuario podrá visualizar los horarios permanentes de las aulas y su capacidad; al dar clic en algún horario, se podrá observar toda la información del docente. Adicionalmente, se implementará el submódulo Cambios, en el cual se realizarán cambios permanentes en horarios, aulas, o profesores. Ambos submódulos se encuentran dentro del módulo Aulas.

2.3.8.1 Diseño

El proceso para realizar cambios de horarios en aulas a través de la aplicación de administración se presenta en el diagrama de secuencia de la Figura 2.58.

2.3.8.2 Implementación

Para la implementación del submódulo Mostrar Aulas es necesario recuperar la lista de aulas de la base de datos.

El usuario debe inicialmente desplegar la lista de aulas, y seleccionar un aula de la misma. Consecuentemente aparecerá el horario del aula para el semestre en curso. El código correspondiente se explicará a continuación.

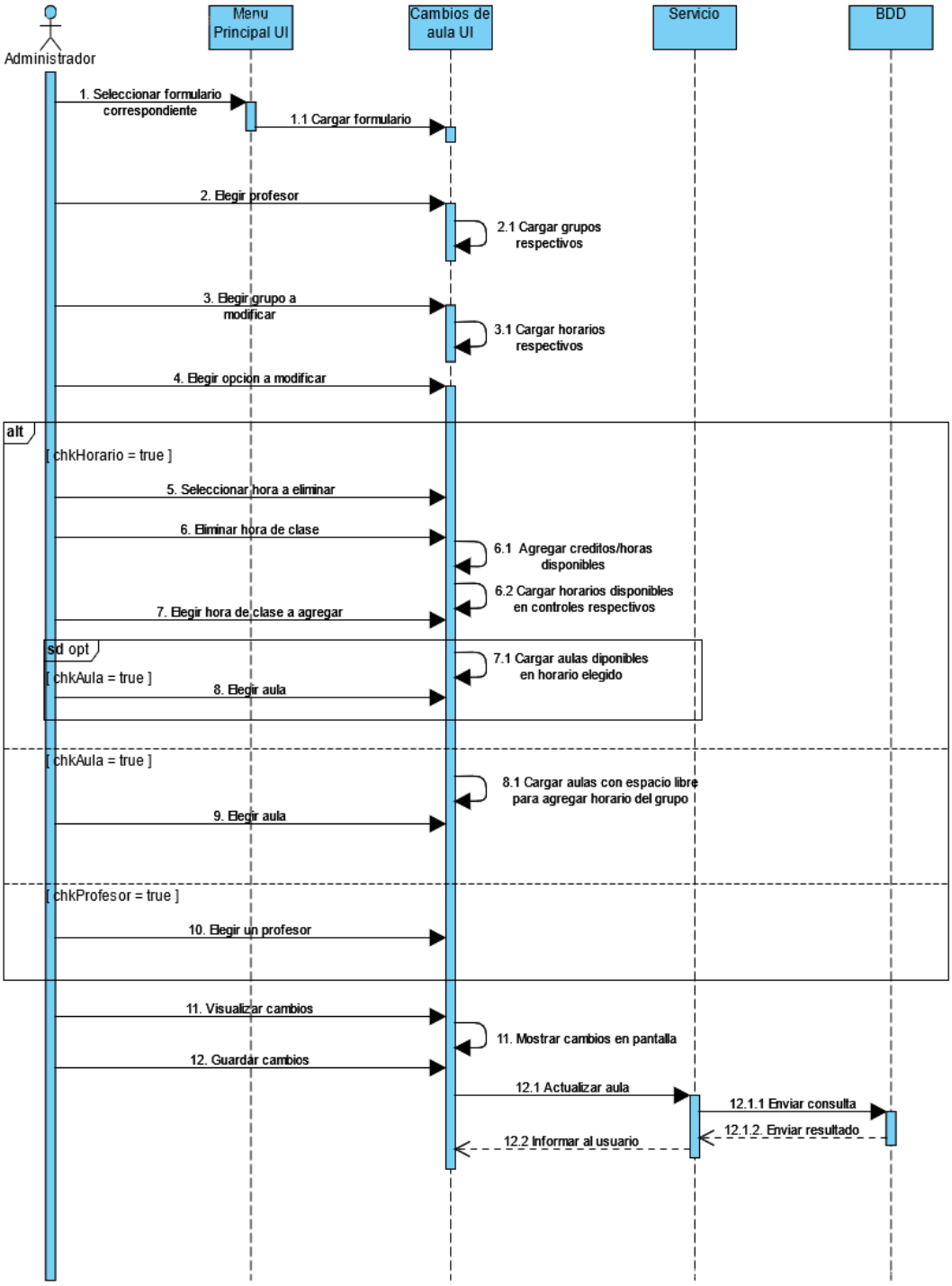


Figura 2.58. Diagrama de secuencia del submódulo Cambios

Al iniciar el submódulo se llama a la función `CargarAulas`, la cual obtiene la lista de aulas correspondiente al semestre en curso y la almacena en la lista auxiliar `aulas`. Se recorre la lista, añadiendo las aulas al control correspondiente.

```
148  public void CargarAulas()
149  {
150             //se llama a la funcion para obtener las listas de la bd
151          aulas = proxy.ObtenerAulas(frmLogin.cliente, frmLogin.semestre);
152
153          foreach (Aula a in aulas)
154          {
155              Console.WriteLine(a.NombreAu + " " + a.Grupos.Count);
156          }
157
158          //se limpia el cmb
159          cmbAulas.Items.Clear();
160              //si se recupero la lista exitosamente
161              if (aulas != null)
162              {
163                  //se organiza la lista por el nombre de aula
164                  aulas = aulas.OrderBy(x => x.NombreAu).ToList();
165                  //se recorre la lista
166                  foreach (Aula a in aulas)
167                  {
168                      //se añade una a una las aulas en el cmb
169                      cmbAulas.Items.Add(a.NombreAu);
170                  }
171              }
172      }
```

Código 2.49. Fragmento función CargarAulas

Cuando el usuario seleccione el aula deseada, se llama a la función `CargarHorarios` que será explicada a continuación, adicionalmente se muestra en pantalla la capacidad del aula.

En la función `CargarHorarios` se recorre los grupos del aula seleccionada y los horarios correspondientes de los mismos, para que puedan ser insertados en la tabla mediante la función `LlenarCeldaHorario` y así poder visualizar el horario del aula completo en pantalla.

```
199  //Funcion para cargar el horario de un aula en el datagrid
     1 referencia | CarrionB, Hace 181 días | 2 autores, 6 cambios
200  public void CargarHorarios(Aula aula)
201  {
202      //se limpia las celdas del dgHorario
203      LimpiarCeldas(dgHorario);
204      //se recorre los grupos del aula seleccionada
205      foreach (Grupo g in aula.Grupos)
206      {
207          //se valida que no sean grupos de recuperacion
208          if (g.Recuperacion == 0)
209          {
210              //se recorre las ranuras del grupo
211              foreach (RanuraTiempo r in g.Horario)
212              {
213                  //si la dia de la ranura coincide con el string del dia
214                  if (r.Dia == "Lunes")
215                  {
216                      //se llena la celda respectiva en el horario
217                      LlenarCeldaHorario(g, r, 0);
218                  }
219                  //este proceso se repite para todos los dias
220                  if (r.Dia == "Martes")
221                  {
222                      LlenarCeldaHorario(g, r, 1);
223                  }
```

Código 2.50. Fragmento función CargarHorarios

En la función `LlenarCeldaHorario`, se crea un lazo con el horario habitual de clases, y se compara la hora de la iteración con la hora de la ranura de tiempo de clases enviada como parámetro de entrada, si existe coincidencia, esta ranura es impresa en pantalla con la información correspondiente, es decir Materia y grupo.

```
260     //Funcion que permite llenar el horario del aula en el dg
        6 referencias | CarrionB, Hace 131 días | 2 autores, 3 cambios
261     public void LlenarCeldaHorario(Grupo g, RanuraTiempo r, int i)
262     {
263         //entero que permite saber la fila en la cual se escribira el horario
264         int inicioFil = 0;
265         //lazo que recorre las filas del dg
266         for (int h = 7; h <= 19; h++)
267         {
268             //si la hora de inicio de la ranura coincide con la hora de la iteracion
269             if (r.HoraInicio.Hour == h)
270             {
271                 //se escribe en la fila y columna correspondiente el nombre de la materia y su grupo
272                 dgHorario.Rows[inicioFil].Cells[i].Value = g.Materia.Nombre + "-" + g.NombreGr;
```

Código 2.51. Fragmento función LlenarCeldaHorario

Para la implementación del submódulo Cambios, es necesario la obtención de las listas de materias, profesores, grupos y aulas de la base de datos. Adicionalmente, se tiene la lista `aulasCambioAux` que será la lista donde se guardarán las aulas modificadas para ser enviada al servidor, y la variable tipo entero `creditosDisponibles`, que controlará el número de créditos/horas disponibles a ser agregados.

```
        1 referencia | CarrionB, Hace 187 días | 2 autores, 5 cambios
99      private void dgHorario_CellClick(object sender, DataGridViewCellEventArgs e)
100     {
101         //se instancia y se inicializa la variable con el valor del indice de la fila seleccionada
102         int rowIndex = e.RowIndex;
103         //si existe seleccion
104         if (rowIndex > -1)
105         {
106             //se instancia y se inicializa la variable con el valor del indice de la fila seleccionada
107             DataGridViewRow row = dgHorario.Rows[rowIndex];
108             //si existe una columna seleccionada, y el valor de la celda no esta vacio
109             if (e.ColumnIndex > -1 && row.Cells[e.ColumnIndex].Value != null && row.Cells[e.ColumnIndex].Value.ToString() != "")
110             {
111                 //se recupera el valor de la celda y se lo almacena en la variable stringAux
112                 string stringAux = row.Cells[e.ColumnIndex].Value.ToString();
113                 //se separa el string recuperado y se almacena la informacion en las siguientes variables
114                 string nombreGr = (stringAux.Split('-'))[1];
115                 string nombreMat = (stringAux.Split('-'))[0];
116                 //entero que permitira encontrar el indice del grupo de la celda selecciona
117                 int index = aulaAux.Grupos.FindIndex(i => i.NombreGr == nombreGr && i.Materia.Nombre == nombreMat
118                 && i.Recuperacion == 0);
119                 //se muestra en el label el nombre del profesor del grupo
120                 lblProfesor.Text = aulaAux.Grupos[index].Profesor.Nombres + " " + aulaAux.Grupos[index].Profesor.Apellidos;
121                 //se muestra en el label el nombre del grupo
122                 lblGr.Text = aulaAux.Grupos[index].NombreGr;
123                 //se muestra en el label los cupos del grupo
124                 lblCupos.Text = aulaAux.Grupos[index].Cupos.ToString();
125                 //si el profesor del grupo seleccionado tiene foto
126                 if (aulaAux.Grupos[index].Profesor.ImagenUrl != null &&
127                     aulaAux.Grupos[index].Profesor.ImagenUrl != "")
128                 {
129                     //se muestra la foto en el picturebox
130                     pctImagenUsuario.Load(aulaAux.Grupos[index].Profesor.ImagenUrl);
131                 }
```

Código 2.52. Algoritmo para mostrar la información de un horario específico

El usuario en primera instancia debe llenar el campo materia, posteriormente seleccionar el grupo en el cual se realizarán cambios; consecuentemente el horario del mismo se mostrará en pantalla.

En este submódulo se podrán realizar distintas operaciones. La función `ManejoOpciones` controlará el ingreso a cada caso y las acciones a realizarse en cada uno de ellos antes de efectuar los cambios permanentes. A continuación, se detallan los diferentes casos:

- **Caso 1:** Agregar o Eliminar un Horario sobre un aula en la que ya se encuentra el grupo (opción Agregar/Eliminar Horario seleccionada).

En este caso se deberá escoger un aula (ya que el grupo puede tener horarios en dos aulas distintas). Cabe mencionar que, para agregar un horario, el grupo deberá tener créditos/horas disponibles respecto a la cantidad de créditos/horas de la materia, para ello se deberá en primera instancia eliminar un horario ya existente, dando clic en una de las celdas que contenga un horario y presionando el botón ***eliminar***, los créditos/horas eliminados se transformarán en créditos/horas disponibles; posteriormente estos se podrán agregar en otro horario de la misma aula.

El código correspondiente se explica a continuación.

Una vez dentro del caso número uno, mediante la variable `creditosDisponibles` se controla que el botón ***Eliminar Horario*** esté o no disponible. Si la variable es igual al número de créditos de la materia seleccionada, no se puede eliminar más horas. Si la variable es igual a cero, no se puede agregar créditos/horas.

```
2010    // Se ingresa en el caso 1
2011    if (chkHorario.Checked && !chkAula.Checked)
2012    {
2013        //se comprueba si existen creditos disponibles
2014        if (creditosDisponibles > 0)
2015        {
2016            //se valida que no se pueda eliminar mas creditos de los que posee la materia
2017            if (creditosDisponibles == matsAux[cmbMateria.SelectedIndex].Creditos)
2018            {
2019                //se deshabilita el boton eliminar
2020                btnEliminarHora.Enabled = false;
2021            }
2022            //caso contrario si se puede eliminar creditos
2023            else
2024            {
2025                //se habilita el boton eliminar
2026                btnEliminarHora.Enabled = true;
2027            }
```

Código 2.53. Verificación de créditos disponibles

Cuando se dispone de créditos para poder agregar un horario, se habilitan los controles en la sección Elegir Horario, la cual, basada en el aula que se esté modificando, presenta los horarios disponibles en la misma, para que el usuario realice la selección. Al terminar de elegir el horario, se valida que el rango de horas seleccionado no sea mayor al número de `créditosdisponibles`, y que sea válido mediante la función `ControlHorarioIngresado`(Código 2.54).

A su vez la función `ControlHorarioIngresado` llama a la función `HabilitarCambios` cuando el horario seleccionado cumple con las condiciones, la cual al verificar que todos los campos estén llenos habilita el botón ***Visualizar Cambios*** cuya función será explicada en breve.

- **Caso 2:** Cambiar un horario completo de un aula a otra (opción Cambiar Aula seleccionada).

En este caso no se requiere de créditos/horas disponibles, únicamente se da clic en la tabla que contiene el horario del grupo (en la celda que indica el aula), y automáticamente el sistema generará un grupo de aulas con ese horario disponible, mismo que se mostrará en la lista al querer seleccionar la nueva aula donde se moverá todo el horario.

```
1780    //si el resultado de la operacion es menor a cero, significa que se escogieron mal las horas
1781    else if (restAux <= 0)
1782    {
1783        //se muestra el siguiente mensaje al usuario
1784        string message = "La hora de fin no puede ser menor a la hora de inicio";
1785        string caption = "Aviso";
1786        MessageBoxButtons buttons = MessageBoxButtons.OK;
1787        MessageBoxIcon icon = MessageBoxIcon.Warning;
1788        MessageBox.Show(message, caption, buttons, icon);
1789        //se limpian los combo box
1790        cmbHoraIni.Text = String.Empty;
1791        cmbHoraIni.Focus();
1792        //para cambiar el indice seleccionado y que se limpie el combo box
1793        cmbHoraFin.SelectedIndex = -1;
1794    }
1795    //si el resultado de la resta es mayor a los creditos disponibles, entonces no se puede agregar este horario
1796    else
1797    {
1798        //se muestra el siguiente mensaje al usuario
1799        string message = "Las horas seleccionadas excede el limite de creditos disponibles (" + creditosDisponibles + ")";
1800        string caption = "Aviso";
1801        MessageBoxButtons buttons = MessageBoxButtons.OK;
1802        MessageBoxIcon icon = MessageBoxIcon.Warning;
1803        MessageBox.Show(message, caption, buttons, icon);
1804        //se limpian los combo box
1805        cmbHoraIni.Text = String.Empty;
1806        cmbHoraIni.Focus();
1807        //para cambiar el indice seleccionado y que se limpie el combo box
1808        cmbHoraFin.SelectedIndex = -1;
1809    }
```

Código 2.54. Fragmento función ControlHorarioIngresado

El código de este caso, se explica a continuación.

Se tiene una variable de tipo *booleano* `aulaValida` que valida cuáles aulas de la lista podrían integrar el grupo que se desea cambiar, para esto se recorre la lista recuperada de la base de datos y una por una se evalúa que la capacidad del aula sea superior o igual a la cantidad de cupos del grupo. Adicionalmente se evalúa que el horario del grupo no coincida con los horarios de los grupos del aula Código 2.55.

En caso de que el aula cumpla con estas condiciones, es válida. Si el aula fue sujeta a cambios anteriormente se encontrará en la lista `aulasCambioAux`; para finalizar se verifica que el grupo a cambiar no se encuentre en el aula, si esto se cumple, el aula es añadida a la lista de aulas disponibles para el grupo.

- **Caso 3:** Cambiar profesor (opción Cambiar Profesor seleccionada).

En este caso el grupo (una vez seleccionado en la tabla de grupos) cambiará de docente, al seleccionar un nuevo docente de la lista; el sistema validará que el mismo tenga disponibilidad de tiempo en el horario del grupo para que se pueda realizar el cambio. Cabe señalar que este cambio es aislado, es decir si esta opción está seleccionada ninguna de las otras dos opciones podrá ser seleccionadas simultáneamente.

```
2125    bool aulaValida = true;
2126    //lazo que permite recorrer la lista de aulas recuperada de la bd
2127    foreach (Aula a in aulasAux)
2128    {
2129        //se valida que la capacidad del aula no sea menor a la cantidad de cupos del grupo
2130        if (a.Capacidad < aulaAux.Grupos[indexGr].Cupos)
2131        {
2132            //si es menor, el aula inmediatamente es descartada como aula valida
2133            aulaValida = false;
2134        }
2135        //caso contrario
2136        else
2137        {
2138            //se recorre los grupos de cada aula
2139            foreach (Grupo g in a.Grupos)
2140            {
2141                //se comprueba que sean grupos de horarios permanentes
2142                if (g.Recuperacion == 0)
2143                {
2144                    //se recorre el horario del grupo en la iteracion
2145                    foreach (RanuraTiempo r in g.Horario)
2146                    {
2147                        //se recorren las ranuras del grupo seleccionado
2148                        foreach (RanuraTiempo rGr in aulaAux.Grupos[indexGr].Horario)
2149                        {
2150                            //si alguna de estas ranuras coincide con las del grupo de la iteracion
2151                            if (r.Dia == rGr.Dia && r.HoraInicio.Hour >= rGr.HoraInicio.Hour && r.Horafin.Hour <= rGr.Horafin.Hour)
2152                            {
2153                                //entonces los horarios se cruzan y el aula es invalida
2154                                aulaValida = false;
2155                                break;
2156                            }
2157                        }
2158                        //si el aula no es valida sale del lazo y deja de comprobar el resto de horarios
2159                        if (!aulaValida)
2160                        {
2161                            break;
2162                        }
2163                    }
```

Código 2.55. Algoritmo para validar aulas

El código correspondiente se explica a continuación.

Se recorre la lista de `aulasCambioAux`, en busca del grupo que fue seleccionado para ser modificado, cuando es encontrado, se recupera el nombre del profesor original del grupo para ser mostrado en el control correspondiente, el cual, al ser presionado, mostrará la lista de profesores disponibles para el grupo, con el fin de que se seleccione el nuevo docente.

```
2214    //se ingresa en el caso 3
2215    if (chkProfesor.Checked)
2216    {
2217        //se habilita el cmbProfesor
2218        cmbProfesor.Enabled = true;
2219        //lazo que recorre la lista de aulas en las que se encuentra el grupo
2220        foreach (Aula a in aulasCambioAux)
2221        {
2222            //lazo que recorre los grupos por aula
2223            foreach (Grupo g in a.Grupos)
2224            {
2225                //si el grupo de la iteracion coincide con el grupo seleccionado
2226                if (cmbMateria.Text == g.Materia.Nombre && g.NombreGr == nombreGr && g.Recuperacion == 0)
2227                {
2228                    //entonces se carga por defecto en el cmb el nombre del profesor actual del grupo
2229                    cmbProfesor.Text = g.Profesor.Nombres + " " + g.Profesor.Apellidos;
2230                }
2231            }
2232        }
2233    }
```

Código 2.56. Proceso para mostrar el docente a cambiar

- **Caso 4:** Ingresar un horario extra en otra Aula (opción Agregar/Eliminar Horario y opción Cambiar Aula seleccionadas).

Para entrar en este caso, se debe previamente haber eliminado un horario (mediante el primer caso), para disponer de créditos/horas. En esta opción no es necesario haber realizado una selección en la tabla horario, puesto que, se añadirá un horario en un aula en la que no se encuentre ya el grupo. Se deberá en primera instancia seleccionar el horario deseado en los controles, el sistema automáticamente devolverá aulas disponibles en el horario elegido.

El código correspondiente se explica a continuación.

Una vez comprobado que se tengan créditos/horas disponibles, se habilitan los controles correspondientes y se llenan con los valores por defecto (Código 2.57).

El usuario seleccionará el horario deseado, consecuentemente el sistema retornará aulas disponibles para el horario seleccionado que cumplan con la capacidad necesaria para el grupo que está siendo modificado. Esto se logrará mediante la función `CargarAulasValidas` (Código 2.58).

```
2240    if (creditosDisponibles > 0)
2241    {
2242        //se recupera el valor del nombre del aula del datagrid
2243        string nombreAu = dgHorario.Rows[0].Cells[0].Value.ToString();
2244        //se busca el aula correspondiente en la lista proveniente de la bd
2245        int index = aulasAux.FindIndex(i => i.NombreAu == nombreAu);
2246        //se instancia una nueva aula
2247        aulaAux = new Aula();
2248        //se le asigna el objeto encontrado
2249        aulaAux = aulasAux[index];
2250        //se obtiene el indice del grupo en el aula
2251        indexGr = aulaAux.Grupos.FindIndex(i => i.NombreGr == nombreGr && i.Materia.Nombre == cmbMateria.Text && i.Recuperacion == 0);
2252        //se habilita el cmbDia
2253        cmbDia.Enabled = true;
2254        //se cargan los dias disponibles
2255        CargarCmbDia();
2256        // se habilita el cmbAula, sin embargo no permite realizar una seleccion hasta que se haya
2257        //selecciona un horario completo.
2258        cmbAula.Enabled = true;
2259        //se inhabilita el boton eliminar, puesto que esta accion solo esta permitida en
2260        //el caso uno, para mayor organizacion
2261        btnEliminarHora.Enabled = false;
2262        //se limpian los cmb de horas
2263        cmbHoraIni.Items.Clear();
2264        cmbHoraFin.Items.Clear();
2265        //se instancia un entero para llenar los cmb de horas con todos los horarios posibles
2266        int hour = 7;
2267        //lazo que permite llenar los cmb hasta las 19:00 horas que es el horario habitual de la FIEE
2268        for (int i = 0; i < 13; i++)
2269        {
2270            cmbHoraIni.Items.Add(hour + ":00");
2271            cmbHoraFin.Items.Add(hour + 1 + ":00");
2272            hour++;
2273        }
2274    }
```

Código 2.57. Fragmento función ManejoOpciones

```
676     //se comprueba que los cmb del horario esten escritos
677     if (cmbDia.Text != "" && cmbHoraIni.Text != "" && cmbHoraFin.Text != "")
678     {
679         //si se ingresa en el caso 4
680         if (chkAula.Checked)
681         {
682             //se cargan las aulas validas en el cmb
683             CargarAulasValidas();
684         }
685     }
```

Código 2.58. Llamado función CargarAulasValidas una vez llenos los controles de horarios

La función `CargarAulasValidas` tiene una variable de tipo *booleano* `aulaValida` que valida cuáles aulas de la lista tienen disponibilidad para el horario escogido; para esto se recorre la lista de aulas recuperada de la base de datos, adicionalmente se evalúa una por una que la capacidad del aula sea superior a la cantidad de cupos del grupo (Código 2.59).

Al realizar los procedimientos especificados anteriormente, se debe presionar el botón Visualizar Cambios para observar el resultado en pantalla.

El código del botón visualizar se explica a continuación.

Para el cuarto caso, se obtiene el aula seleccionada por el usuario, y se le añade el grupo que está siendo modificado (Código 2.60).

```
1150  public void CargarAulasValidas()
1151  {
1152      //se crea un booleano que permitira definir si un aula es valida
1153      bool aulaValida = true;
1154      //se limpia el cmb aula
1155      cmbAula.Items.Clear();
1156      //lazo que recorre todas las aulas almacenadas en la bd
1157      foreach (Aula a in aulasAux)
1158      {
1159          //se valida que la capaidad del aula que no sea menor que la del grupo en cuestion
1160          if (a.Capacidad < aulaAux.Grupos[indexGr].Cupos)
1161          {
1162              aulaValida = false;
1163          }
1164          //enn caso de cumplir dicha condicion
1165          else
1166          {
1167              //lazo que recorre los grupos del aula
1168              foreach (Grupo g in a.Grupos)
1169              {
1170                  //valida que el grupo en cuestion no sea de recuperacion
1171                  if (g.Recuperacion == 0)
1172                  {
1173                      //se  recorre las ranuras del horario del grupo
1174                      foreach (RanuraTiempo r in g.Horario)
1175                      {
1176                          //si las ranuras del horario del grupo coinciden con los horarios elegidos por el usuario en los cmb
1177                          if (r.Dia == cmbDia.Text && r.HoraInicio.Hour >= Convert.ToInt32(cmbHoraIni.Text.Split(':')[0])
1178                                  && r.Horafin.Hour <= Convert.ToInt32(cmbHoraFin.Text.Split(':')[0]))
1179                          {
1180                              //entonces el aula no es valida
1181                              aulaValida = false;
1182                              break;
1183                          }
1184                      }
1185                      //si no es valida, sale del lazo
1186                      if (!aulaValida)
1187                      {
1188                          break;
1189                      }
```

Código 2.59. Fragmento función CargarAulasValidas

```
264  //se obtiene el indice del objeto aula en base a lo seleccionado en el cmb
265  indexA = aulasAux.FindIndex(i => i.NombreAu == cmbAula.Text);
266  aulaAux = aulasAux[indexA];
267  //se aniade el nuevo objeto grupo a la nueva aula
268  aulaAux.Grupos.Add(grAux);
269  //obtenemos la posicion del grupo en el aula
270  indexGr = aulaAux.Grupos.FindIndex(i => i.NombreGr == nombreGr && i.Materia.Nombre == cmbMateria.Text && i.Recuperacion == 0);
```

Código 2.60. Proceso para visualizar cambios en el caso 4

Para el segundo caso, se remueve el grupo del aula original, y se actualiza el aula ya modificada en la lista `aulasCambioAux`. Posteriormente, se obtiene la nueva aula seleccionada por el usuario y se agrega el nuevo horario; después se añade a la lista de aulas modificadas `aulasCambioAux` (Código 2.61).

Para el primero y cuarto caso, en el botón Visualizar Cambios se realiza la asignación del nuevo horario elegido en el aula sobre la que se está trabajando, y se actualiza el número de créditos/horas disponibles. Si el aula se encuentra en la lista auxiliar `aulasCambioAux` se la actualiza, caso contrario se añade a la lista. Seguidamente se comprueba para el caso 4 que no esté el grupo del aula que se retiró para que éste no aparezca visualmente en la tabla (Código 2.62).

```
285 //se remueve el grupo del aula en la que se encontraba el grupo
286 aulaAux.Grupos.RemoveAt(indexGr);
287 //se obtiene el indice del objeto aula en donde se encontraba el gr
288 indexA = aulasCambioAux.FindIndex(i => i.NombreAu == aulaAux.NombreAu);
289 //se aniade el aula modificada en la posicion correspondiente
290 aulasCambioAux[indexA] = aulaAux;
291 //se obtiene el indice del objeto aula en la lista recuperada de la bd en base a lo seleccionado en el cmb
292 indexA = aulasAux.FindIndex(i => i.NombreAu == cmbAula.Text);
293 //se  obtiene la nueva aula en la que agregara el gr
294 aulaAux = aulasAux[indexA];
295 //obtiene el indice del grupo en base a nombre grupos, materia, y teniendo en cuenta que sea un grupo permanente
296 int indexAux = aulaAux.Grupos.FindIndex(i => i.NombreGr == nombreGr && i.Materia.Nombre == cmbMateria.Text && i.Recuperacion == 0);
297 //si se encuentra en la lista de la bd
298 if (indexAux > -1)
299 {
300     //lazo que recorre el horario del grupo seleccionado para poder agregarlo a la nueva aula
301     foreach (RanuraTiempo r in grAux.Horario)
302     {
303         aulaAux.Grupos[indexAux].Horario.Add(r);
304     }
305 }
306 //si no se encuentra el gr en el aula
307 else
308 {
309     //se aniade el grupo al aula
310     aulaAux.Grupos.Add(grAux);
311     //se comprueba que el aula no haya sido modificada antes
312     indexA = aulasCambioAux.FindIndex(i => i.NombreAu == aulaAux.NombreAu);
313     //si no fue modificada
314     if (indexA == -1)
315     {
316         //se aniade a la lista de aulas modificadas
317         aulasCambioAux.Add(aulaAux);
318     }
319     //caso contrario
320     else
321     {
322         //se iguala a la referencia en la posicion correspondiente
323         aulasCambioAux[indexA] = aulaAux;
324     }
325 }
```

Código 2.61. Proceso para visualizar cambios en el caso 2

```
332 //se ingresaen el caso uno
333 if (chkHorario.Checked)
334 {
335     //se ingresa la ranura creada en el horario del grupo
336     aulaAux.Grupos[indexGr].Horario.Add(rtAux);
337     //se disminuyen los creditos disponibles segun el horario escogido
338     creditosDisponibles = creditosDisponibles - (rtAux.Horafin.Hour - rtAux.HoraInicio.Hour);
339     //se busca si el aula esta en aulas cambio
340     indexA = aulasCambioAux.FindIndex(i => i.NombreAu == aulaAux.NombreAu);
341     //si no le encuentra
342     if (indexA == -1)
343     {
344         //se aniade el aula a aulas cambio
345         aulasCambioAux.Add(aulaAux);
346     }
347     //caso contrario
348     else
349     {
350         //se iguala a la referencia en la posicion correspondiente
351         aulasCambioAux[indexA] = aulaAux;
352     }
353     if (chkAula.Checked)
354     {
355         bool horarioVacio = true;
356         if(dgHorario.Rows.Count == 1)
357         {
358             for(int  i = 1; i < dgHorario.Rows[0].Cells.Count; i++)
359             {
360                 if(dgHorario.Rows[0].Cells[i].Value != null)
361                 {
362                     horarioVacio = false;
363                     break;
364                 }
365             }
366         }
```

Código 2.62. Proceso para visualizar cambios en el caso 1 y caso 4

Para el tercer caso, se controla directamente desde el control que muestra la lista de profesores; el código correspondiente se explica a continuación.

Se recorre la lista de aulas que han sido modificadas, en busca de grupos que le pertenezcan al nuevo docente al que se le va asignar el grupo sobre el cual se está trabajando. Cuando se encuentran grupos del nuevo docente se verifica que no exista cruce de horarios entre los grupos del docente y el nuevo grupo que se le va a asignar (Código 2.63).

En caso de no existir cruce de horarios se realiza el cambio de docente en el grupo sobre el cual se está trabajando (Código 2.64).

Si el usuario está conforme con la vista previa, se debe seleccionar el botón guardar cambios; seguidamente se verifica que el grupo tenga el horario completo, y se envíen los cambios al servidor mediante la función `ActualizarGr` para que los cambios sean efectuados de forma permanente tanto en la base local como en Firebase (Código 2.65).

```
841 //boleano que nos permitira saber si el horario del gr en el que estamos trabajando
842 //no coincide con los horarios del profesor seleccionado
843 bool grValido = true;
844 //lazo que recorre las aulas donde se encuentra el gr
845 foreach (Aula a in aulasCambioAux)
846 {
847     //lazo que recorre los grs de cada aula
848     foreach (Grupo g in a.Grupos)
849     {
850         //si coincide el gr de la iteracion con el gr en el que estamos trabajando
851         if (cmbMateria.Text == g.Materia.Nombre && g.NombreGr == nombreGr && g.Recuperacion == 0)
852         {
853             //lazo que recorre las ranuras del grupo
854             foreach (RanuraTiempo R in g.Horario)
855             {
856                 //lazo que recorre todos los grupos para encontrar los horarios del profesor
857                 foreach (Grupo gr in grsAux)
858                 {
859                     //si encuentra un grupo del profesor al que se le va a asignar el horario
860                     if (gr.Profesor.Cedula == cedula && gr.Recuperacion == 0 && gr.Profesor.Cedula != g.Profesor.Cedula)
861                     {
862                         //lazo que recorre los horarios de dicho gr del nuevo profesor
863                         foreach (RanuraTiempo r in gr.Horario)
864                         {
865                             //si se encuentra un cruce de horarios entre los horarios del profesor y del gr en el que se esta trabajando
866                             if (r.Dia == R.Dia && r.HoraInicio.Hour >= R.HoraInicio.Hour && r.Horafin.Hour <= R.Horafin.Hour)
867                             {
868                                 //entonces el gr no es valido
869                                 grValido = false;
870                                 break;
871                             }
872                         }
873                         //si el gr no es valido, avanza al siguiente gr
874                         if (!grValido)
875                         {
876                             break;
877                         }
878                     }
879                 }
880             }
```

Código 2.63. Proceso para verificación del no cruce de horarios del docente

```
885 //si el gr es valido para el nuevo profesor
886 if (grValido)
887 {
888     //lazo que recorre las aulas donde esta el gr sobre el que se esta trabajando
889     foreach (Aula a in aulasCambioAux)
890     {
891         //lazo que recorre los grupos de cada aula
892         foreach (Grupo g in a.Grupos)
893         {
894             //si el grupo de la iteracion coincide con el grupo sobre el que estamos trabajando
895             if (cmbMateria.Text == g.Materia.Nombre && g.NombreGr == nombreGr && g.Recuperacion == 0)
896             {
897                 //se le asigna el nuevo profesor
898                 g.Profesor.Cedula = profsAux[cmbProfesor.SelectedIndex].Cedula;
899                 //se habilita el boton guardar
900                 btnGuardar.Enabled = true;
901             }
902         }
903     }
```

Código 2.64. Algoritmo modificación de docente

```
418 //se comprueba si el grupo tiene sus creditos completos
419 if (creditosDisponibles == 0)
420 {
421     //se utiliza un objeto _proxy para poder  agregar los cambios al servidor
422     //si se efectuaron los cambios con exito, la funcion devolvera un entero mayor a cero
423     int res = proxy.ActualizarGr(frmLogin.cliente, aulasCambioAux);
```

Código 2.65. Proceso para enviar los cambios al servidor

2.3.8.3 Entregable

El entregable de este *sprint* consiste en el submódulo Cambios completo y funcional.

2.3.9 SPRINT 9

En este *sprint* se implementará el submódulo Reservas que se encuentra dentro del módulo Aulas, en el cual se realizarán reservas de aulas para recuperación de horas de clases. La interfaz gráfica de este submódulo se presenta en la Figura 2.45 (b).

2.3.9.1 Diseño

El proceso para reservar un aula de clase a través de la aplicación de administración se presenta en el diagrama de secuencia de la Figura 2.59.

2.3.9.2 Implementación

Para la implementación del submódulo Reservas, es necesario la obtención de las listas de materias, profesores, grupos y aulas de la base de datos.

En este submódulo existen dos formas de realizar el proceso de reserva.

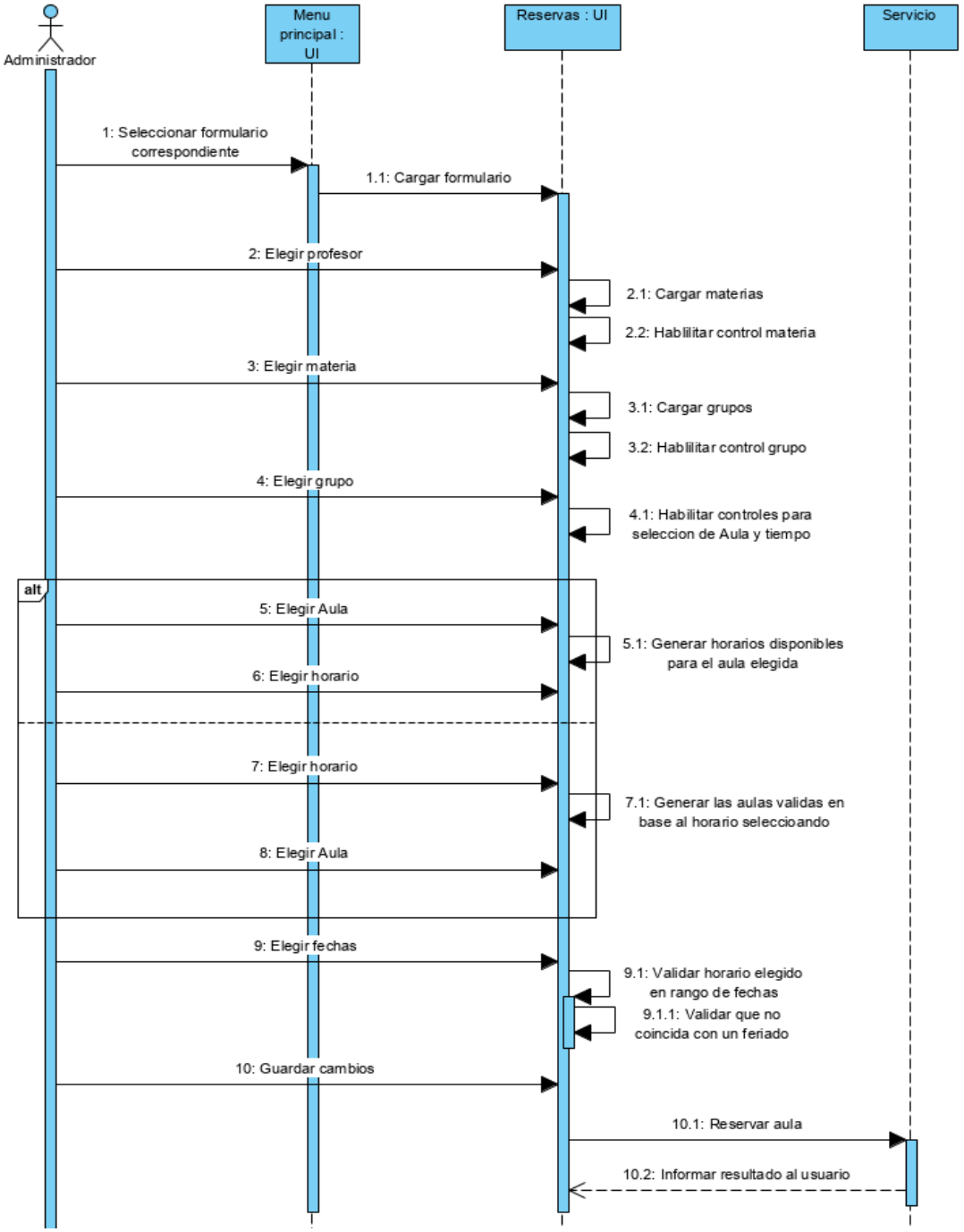


Figura 2.59. Diagrama de secuencia del submódulo Reservas

En la primera forma, el usuario debe llenar los campos profesor, materia, grupo y seleccionar un aula de la lista presentada. Se mostrará en pantalla el horario del aula incluyendo las horas ya reservadas según la fecha seleccionada; posteriormente, el usuario escogerá el horario de la lista de horarios disponibles.

El código correspondiente se explica a continuación.

Si existe selección en el `cmbAulas`, se instancia un aula auxiliar que corresponde al aula seleccionada, se carga el horario de la misma en la tabla mediante la función `CargarHorarios`, se evalúa si ya se ha seleccionado un horario, en cuyo caso se actualizarán los horarios disponibles para cualquier modificación y se habilita el botón para realizar la reserva.

```
408  private void cmbAulas_SelectedIndexChanged(object sender, EventArgs e)
409  {
410          //valida que haya una seleccion en el cmb
411      if(cmbAulas.SelectedIndex > -1)
412      {
413          //se inicializa el aula
414          aulaAux = new Aula();
415          //se busca el aula en la listade aulas de la bd
416          int indexA = aulasAux.FindIndex(i => i.NombreAu == cmbAulas.Text);
417          //se iguala al aula encontrada en la lista
418          aulaAux = aulasAux[indexA];
419          //se carga el horario en el dg
420          CargarHorarios(aulaAux);
421          //se habilita el dg
422          dgHorario.Enabled = true;
423          //si se ha seleccionado un horario
424          if (cmbDia.Text != "" && cmbHoraIni.Text != "" && cmbHoraFin.Text != "" && cmbAulas.Text != "")
425          {
426              //Se cargan los horarios en base al aula seleccionada
427              CargarHorariosCmb();
428              //se habilita el boton reservar
429              btnReservar.Enabled = true;
430          }
```

Código 2.66. Proceso de habilitación de controles al elegir un aula

En la función `CargarHorarios`, se recorren los grupos del aula seleccionada, con el propósito de mostrar en la tabla Horario las horas de recuperación existentes en la semana de la fecha seleccionada, o la primera semana del rango seleccionado (Código 2.67).

En la función `CargarHorariosCmb`, se instancian dos listas de ranuras de tiempo; en la primera se almacenan las ranuras correspondientes a los horarios del aula para el día seleccionado incluyendo horarios de recuperación; y, en la segunda se almacenan los horarios por defecto del día seleccionado (Código 2.68).

Posteriormente, se realiza una comparación de muchos a muchos para poder obtener los horarios disponibles en el aula para el día seleccionado (Código 2.69).

En la segunda forma, el usuario primero selecciona el horario deseado, y el sistema devolverá una lista de aulas disponibles en dicho horario. De igual forma al seleccionar el aula se presentará el horario de la misma según la fecha seleccionada.

```
1220  public void CargarHorarios(Aula aula)
1221  {
1222      //se limpian las celdas del dg
1223      LimpiarCeldas(dgHorario);
1224
1225      //booleano que permite saber que grupos de recuperacion son validos
1226      bool grupoValido = true;
1227
1228      //lazo que recorre los grupos del aula
1229      foreach (Grupo g in aula.Grupos)
1230      {
1231          //si el grupo es de recuperacion
1232          if (g.Recuperacion == 1)
1233          {
1234              //y la seleccion del dtp es para un solo dia de recuperacion y no para un rango
1235              if (fechaInicio == fechaFin)
1236              {
1237                  //genero la fecha inicio, del inicio de esa semana
1238                  fechaInicio = dtpIni.Value.Date.StartOfWeek(DayOfWeek.Monday);
1239                  //y la fecha fin del fin de esa semana
1240                  fechaFin = dtpIni.Value.Date.EndOfWeek(DayOfWeek.Sunday);
1241              }
1242              //si el grupo de recuperacion no se encuentra dentro, engloba, o interseca con el rango de fecha inicio
1243              //y fecha fin, entonces el grupo no es valido
1244              if (!((g.FechaInicio.Date >= fechaInicio && g.FechaInicio.Date <= fechaFin) ||
1245                  (g.FechaFin.Date >= fechaInicio && g.FechaFin.Date <= fechaFin) ||
1246                  (g.FechaInicio.Date < fechaInicio && g.FechaFin.Date > fechaFin)))
1247              {
1248                  grupoValido = false;
1249              }
1250          }
```

Código 2.67. Fragmento función CargarHorarios

```
1076  //se instacia una lista de ranuras de tiempo que almacenara las ranuras correspondientes al aula
1077  List<RanuraTiempo> rtsAux = new List<RanuraTiempo>();
1078
1079  //se instancia una lista de ranuras de tiempo que almacenara todas las ranuras posibles
1080  List<RanuraTiempo> rtsAux2 = new List<RanuraTiempo>();
```

Código 2.68. Instanciación de listas auxiliares para guardar horario

```
1188  //se crea un booleano que permitira definir las horas en que esta disponible el aula en cuestion
1189  bool horaValida = true;
1190  //lazo que recorre las ranuras posibles de un dia
1191  foreach (RanuraTiempo r2 in rtsAux2)
1192  {
1193      //lazo que recorre las ranuras del aula para realizar una comparacion de muchos a muchos
1194      //con el fin de conocer las horas disponibles en  el aula
1195      foreach (RanuraTiempo r1 in rtsAux)
1196      {
1197          //se compara entre ranuras, si estas coinciden, el aula no estara disponible en esta ranura
1198          if (r2.HoraInicio.Hour <= r1.HoraInicio.Hour && r2.Horafin.Hour >= r1.Horafin.Hour)
1199          {
1200              horaValida = false;
1201              break;
1202          }
1203      }
1204      //si no hubo coincidencia la ranura esta disponible, por tanto se aniade a los cmb correspondientes
1205      if (horaValida)
1206      {
1207          cmbHoraIni.Items.Add(r2.HoraInicio.Hour + ":00");
1208          cmbHoraFin.Items.Add(r2.Horafin.Hour + ":00");
1209      }
1210
1211      //caso contrario se reinicia la variable horaValida y el lazo continua
1212      else
1213      {
1214          horaValida = true;
1215      }
1216  }
```

Código 2.69. Obtención de horarios disponibles del aula seleccionada

El código se explica a continuación.

Si existe selección de día, se asigna el día en la variable global `rtAux` del tipo `RanuraTiempo`. Si existe selección de aula, se habilitan los controles para ingresar el horario y se cargarán en éstos los horarios disponibles del aula que ya fue seleccionada, caso contrario se cargan los horarios por defecto en base al día con las validaciones de festividades y eventos culturales.

En caso de modificaciones en el horario, automáticamente se cargarán las aulas válidas para ese horario en el control respectivo.

Si se tienen todos los controles llenos se habilita el botón para realizar la reserva.

```
455     //si existe seleccion en el cmb
456     if (cmbDia.SelectedIndex > -1)
457     {
458         //se igualael dia de la ranura al dia del cmb
459         rtAux.Dia = cmbDia.Text;
460         //si se seleccion un aula
461         if (cmbAulas.SelectedIndex >-1)
462         {
463             //se habilitan los cmb de horas
464             cmbHoraIni.Enabled = true;
465             cmbHoraFin.Enabled = true;
466 
467             //se limpia la propiedad de los cmb
468             cmbHoraIni.Text = "";
469             cmbHoraFin.Text = "";
470             //se caraga el horario disponible del aula en los cmb
471             CargarHorariosCmb();
472 
473         }else
474         {
475             //se habilitan los cmb de horas
476             cmbHoraIni.Enabled = true;
477             cmbHoraFin.Enabled = true;
478             //se cargan los horarios por defecto en base al dia con las validaciones respectivas
479             CargarHorariosSinAula();
480         }
481         //si se ha seleccionado un horario
482         if (cmbDia.Text != "" && cmbHoraIni.Text != "" && cmbHoraFin.Text != "" && cmbAulas.Text == "")
483         {
484             //se cargan las aulas validas en el horairo
485             CargarAulasValidas();
486         }
487         //si se tienen todos los campos llenos
488         if (cmbDia.Text != "" && cmbHoraIni.Text != "" && cmbHoraFin.Text != "" && cmbAulas.Text != "")
489         {
490             //se cargan las aulas validas en el horairo
491             CargarAulasValidas();
492             //se habilita el boton reservar
493             btnReservar.Enabled = true;
494         }
```

Código 2.70. Proceso de habilitación de controles al elegir un día

En la función `CargarHorariosSinAula`, se instancia una variable entera `horaMax` para llenar los controles de horas con el horario por defecto según el día. Para los días laborables exceptuando el día jueves por los eventos culturales realizados en la EPN, se llenarán los controles con el horario disponible de 7 a 20 horas, restringiendo el horario del almuerzo de 13:00 a 14:00, en el caso del día sábado la hora máxima será a las 12:00.

En el caso de los días jueves, se restringe el horario de 11:00 a 14:00, en el cual no es permitido dictar clases (Código 2.71).

En la función `CargarAulasValidas`, se filtran las aulas válidas para el horario seleccionado, que adicionalmente tengan la capacidad suficiente para cubrir los cupos del grupo. Para ello, se recorren los grupos de cada aula de la lista de aulas recuperada de la base de datos, comparando las ranuras de los horarios de los grupos con el horario seleccionado, si no existe coincidencia alguna, el aula es válida y es agregada al control (Código 2.72).

```
1294  public void CargarHorariosSinAula()
1295  {
1296      //se limpian ambos cmb
1297      cmbHoraIni.Items.Clear();
1298      cmbHoraFin.Items.Clear();
1299      //se setea una variable para poder controlar la hora maxima
1300      int horaMax = 19;
1301
1302      //si el dia es sabado
1303      if (cmbDia.Text == "Sabado")
1304      {
1305          //la hora maxima sera hasta las 12
1306          horaMax = 12;
1307      }
1308
1309      //se crean las ranuras posibles para un dia en los horarios de 7 hasta el valor de hora maxima
1310      for (int i = 7; i <= horaMax; i++)
1311      {
1312          //se restringe el horario de 1 a 2 de la tarde
1313          if (i != 13)
1314          {
1315              //para el caso de todos los dias, exceptuando el jueves
1316              if (cmbDia.Text != "Jueves")
1317              {
1318                  cmbHoraIni.Items.Add(i + ":00");
1319                  cmbHoraFin.Items.Add(i + 1 + ":00");
1320              }
1321              else
1322              {
1323                  //para el caso del dia jueves se restringe el horario de 11 a 2 de la tarde
1324                  if ((i < 11 || i > 13) && cmbDia.Text == "Jueves")
1325                  {
1326                      cmbHoraIni.Items.Add(i + ":00");
1327                      cmbHoraFin.Items.Add(i + 1 + ":00");
1328                  }
1329              }
1330          }
1331      }
1332  }
```

Código 2.71. Función CargarHorariosSinAula

Una vez realizado este proceso, el usuario debe presionar el botón Reservar, el cual llama a la función `ValidarRangoFechas`, que realiza varias verificaciones que se explicarán a continuación.

Para realizar las validaciones se genera una lista de días contenidos en el rango de fechas seleccionado, se recorre la lista comprobando que el día de la iteración no corresponda a un feriado mediante la función `ComprobarFechaNoFeriado`. Adicionalmente se compara que el día seleccionado en el control “Día” se encuentre dentro del rango seleccionado y que la reserva corresponda a una fecha y una hora posterior a la presente (Código 2.73).

```
1016    foreach (Aula a in aulasAux)
1017    {
1018        //si la capacidad del aula es menor
1019        if (a.Capacidad < grAux.Cupos)
1020        {
1021            //el aula inmediatamente es invalida
1022            aulaValida = false;
1023        }
1024        else
1025        {
1026            //caso contrario se recorren los grupos del aula de iteracion para validar horarios
1027            foreach (Grupo g in a.Grupos)
1028            {
1029                //se recorre el horario del grupo de la iteracion
1030                foreach (RanuraTiempo r in g.Horario)
1031                {
1032                    //si el horario del gr coincide con el elegido por el usuario
1033                    if (r.Dia == cmbDia.Text && r.HoraInicio.Hour >= Convert.ToInt32(cmbHoraIni.Text.Split(':')[0])
1034                        && r.Horafin.Hour <= Convert.ToInt32(cmbHoraFin.Text.Split(':')[0]))
1035                    {
1036                        //el aula inmediatamente es invalida
1037                        aulaValida = false;
1038                        //se sale del lazo
1039                        break;
1040                    }
1041                }
1042                //si el aula es invalida
1043                if (!aulaValida)
1044                {
1045                    //avanza a la siguiente aula
1046                    break;
1047                }
1048            }
1049        }
1050        //si el aula es valida
1051        if (aulaValida)
1052        {
1053            //se agrega al combobox
1054            cmbAulas.Items.Add(a.NombreAu);
1055        }
```

Código 2.72. Fragmento de la función CargarAulasValidas

```
1552    public int ValidarRangoFechas()
1553    {
1554    //entero que nos permitira validar si el rango es valido
1555    int fechaValida = 1;
1556    //objeto de tipo cultureinfo que permice mantener el idioma español
1557    CultureInfo ci = new CultureInfo("ES-es");
1558    //se generan una lista de dias contenidos en el rango seleccionado
1559    IEnumerable<DateTime> range =
1560    Enumerable.Range(0, (dtpFin.Value.Date - dtpIni.Value.Date).Days + 1).Select(d => dtpIni.Value.Date.AddDays(d));
1561    //se recorre la lista de dias
1562    foreach (DateTime d in range)
1563    {
1564        if (new Utilidades().ComprobarFechaNoFeriado(d.Date))
1565        {
1566            //se transforma el dia de la lista al formato deseado (todo minusculas y en español)
1567            string strAux = d.ToString("dddd", ci);
1568            //adicionalmente se retiran las tildes
1569            strAux = strAux.Replace('é', 'e').Replace('á', 'a');
1570            //si el dia de la lista transformado es igual al dia seleccionado en el cmb
1571            if (strAux == cmbDia.Text.ToLower())
1572            {
1573                //entonces el rango seleccionado es valido
1574                fechaValida = 0;
1575                //se sale del lazo
1576                break;
1577            }
1578        }
1579    }
1580
1581    //si el rango no es valido
1582    if (fechaValida == 0)
1583    {
1584        //si el rango seleccionado corresponde a un solo dia, y el dia es el dia presente
1585        if (dtpFin.Value.Date == dtpIni.Value.Date && dtpIni.Value.Date == DateTime.Now.Date)
1586        {
1587            //si el horario escogido es mayor a la hora actual
1588            if (DateTime.Now.Hour >= Convert.ToInt32(cmbHoraIni.Text.Split(':')[0]))
1589            {
1590                //entonces la fecha es valida
1591                fechaValida = 2;
```

Código 2.73. Fragmento función ValidarRangoFechas

```
//Se genera una lista de objetos de tipo Nager.Date.Model.PublicHoliday para obener las fechas de las festividades
var feriadosEc = DateSystem.GetPublicHoliday(fecha.Year, CountryCode.EC);
//Se genera un objeto de tipo CultureInfo para mantener el idioma espaniol
CultureInfo ci = new CultureInfo("ES-es");
//Lazo que recorre las fechas de las festividades
foreach (Nager.Date.Model.PublicHoliday d in feriadosEc)
{
    //Se obtiene el dia de la fecha en cuestion
    string strAux = d.Date.ToString("dddd", ci);
    strAux = strAux.Replace('é', 'e').Replace('á', 'a');

    //Se excluye al Carnaval y Viernes Santo por ser fijos, y al Dia de Difuntos e independencia de Cuenca
    //ya que son dias seguidos y son tratados de manera distinta
    if (d.Name != "Carnival" && d.Name != "Good Friday" &&
        d.Name != "All Souls' Day" && d.Name != "Independence of Cuenca" && d.Name != "New Year's Day")
    {
        //Dependiendo del dia, se aniada o sustrae una cantidad basandose en la ley de Feriados
        switch (strAux)
        {
            case "martes":
                d.Date = d.Date.AddDays(-1);
                break;
            case "miercoles":
                d.Date = d.Date.AddDays(+2);
                break;
            case "jueves":
                d.Date = d.Date.AddDays(+1);
                break;
            case "sabado":
                d.Date = d.Date.AddDays(-1);
                break;
            case "domingo":
                d.Date = d.Date.AddDays(+1);
                break;
        }
    }
```

Código 2.74. Fragmento función ComprobarFechaNoFeriado

```
if (ValidarRangoFechas() == 0)
{
    //se setean el resto de valores al grupo auxiliar
    grAux.FechaInicio = dtpIni.Value.Date;
    grAux.FechaFin = dtpFin.Value.Date;
    grAux.Horario.Add(rtAux);
    grAux.Semestre = frmLogin.semestre;
    //se añade el grupo auxiliar al aula
    aulaAux.Grupos.Add(grAux);
    //se ejecuta la funcion para reservar el aula
    int res = proxy.ReservarAula(frmLogin.cliente, aulaAux);
```

Código 2.75. Proceso para enviar la reserva al servidor

En la función `ComprobarFechaNoFeriado`, se utiliza la librería `DateSystem` para obtener las fechas de los feriados en una lista. Posteriormente se recorren estos feriados, y en base a la Ley de Feriados, se añade o sustrae días despendiendo del día del feriado de la iteración. Cabe recalcar que algunos feriados inamovibles se manejan de forma diferente (Código 2.74) .

Para finalizar en el botón reservar, se envía la reserva al servidor mediante la función `ReservarAula`, para que ésta conste en el sistema (Código 2.75).

2.3.9.3 Entregable

El entregable de este *sprint* consiste en el submódulo Reservas completamente funcional y con sus respectivas validaciones.

2.3.10 SPRINT 10

En este *sprint* se implementará el submódulo Reportes de Asistencia y Reportes de Inasistencia que se encuentran dentro del módulo Reportes.

2.3.10.1 Diseño

El proceso de generación de reportes de asistencia o inasistencia a través de la aplicación de administración se presenta en el diagrama de secuencia de la Figura 2.60.

2.3.10.2 Implementación

Para la implementación del submódulo Reportes de Asistencia cuya interfaz gráfica se observa en la Figura 2.47 (b), es necesario la obtención de las listas de registros, profesores, y grupos de la base de datos, así como la creación de distintas variables que serán utilizadas para el cálculo de estadísticas y obtención de datos.

En este submódulo se podrán generar reportes de asistencia para un docente en particular, dentro de un periodo de tiempo seleccionado. Adicionalmente, los registros de asistencia del docente pueden ser filtrados por materia y por grupo.

Para iniciar el proceso, el usuario debe llenar los campos profesor y rango de fechas de forma obligatoria, opcionalmente se puede agregar un filtro por materia y por grupo para una consulta más específica.

Posteriormente debe presionar el botón ***consultar*** para que el sistema devuelva el listado de registros de asistencia correspondiente, con las estadísticas resultantes del cálculo realizado. Este listado podrá ser exportado a un documento formato Excel o PDF. El código correspondiente se explica a continuación.

Al presionar el botón ***consultar***, se inicializan las variables correspondientes a horas de asistencia y horas de recuperación en cero, y se llama a la función `ObtenerRegistros`, que recupera los registros del docente de la base de datos y los almacena en la variable `regsAux` (Código 2.76).

Se obtiene el horario del profesor mediante la función `GenerarHorarioProf`, que se explicará en breve, para luego mediante éste, calcular las estadísticas. Se recorre la lista `regsAux` y se evalúa que el registro de la iteración se encuentre dentro del rango de fechas seleccionado (Código 2.77).

Consecuentemente, se recorre el horario del grupo evaluando el número de horas de clase por día, mediante esto se conocerá la hora inicio y hora fin de la clase, y se recuperarán en cada iteración los registros que se hayan realizado dentro de este periodo de tiempo.

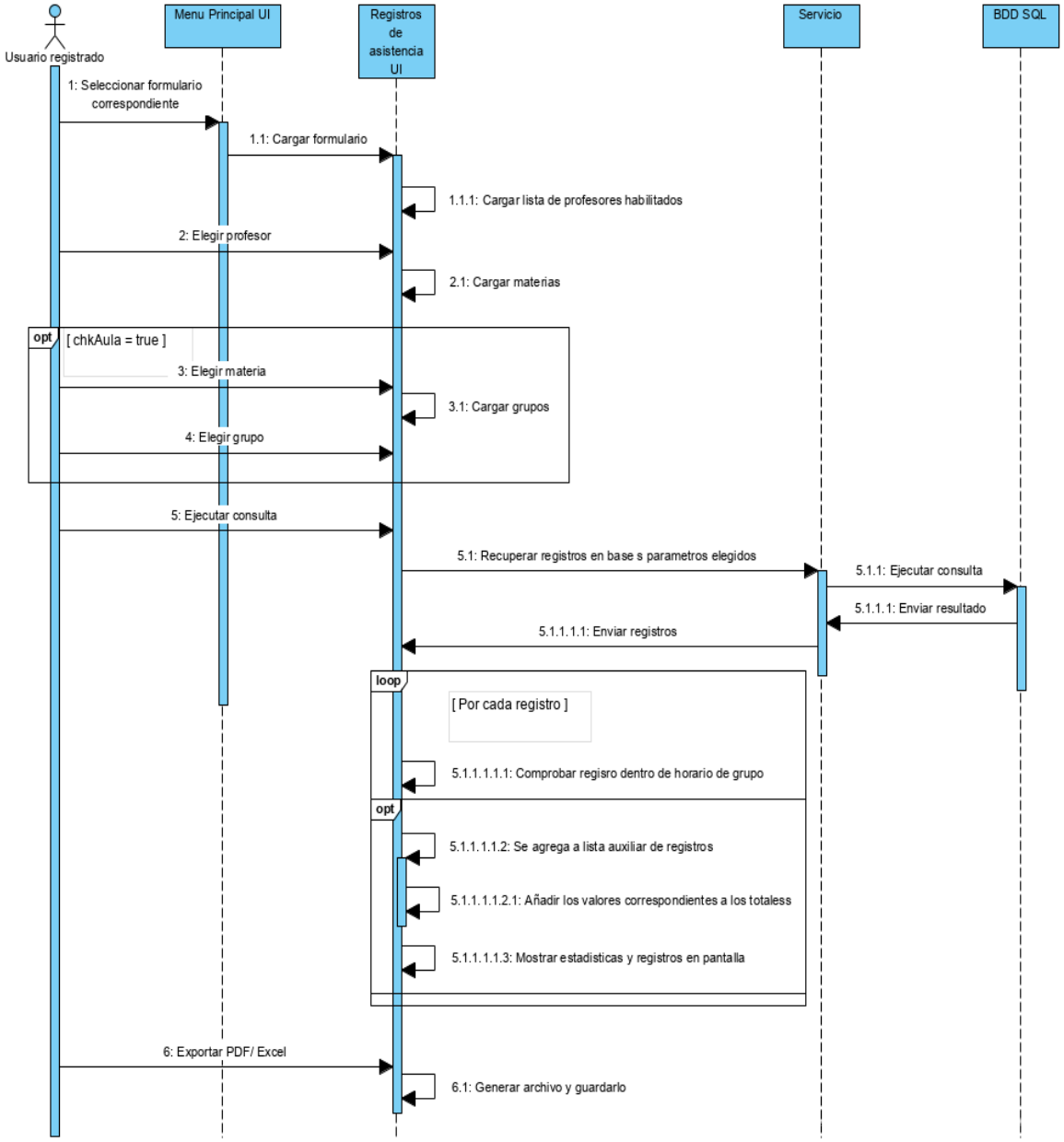


Figura 2.60. Diagrama de secuencia del submódulo Reportes de Asistencia

```
117 totalHorasAsistidas = 0;
118 totalHorasRecuperacion = 0;
119 //se crea un objeto profesor auxiliar que sera seleccionado por el combobox
120 Usuario p = profsAux[cmbProfesor.SelectedIndex];
121 if (chkMateria.Checked)
122 {
123     regsAux = proxy.ObtenerRegistros(frmLogin.cliente, 2, frmLogin.semestre,
124     profsAux[cmbProfesor.SelectedIndex].Cedula + "-" + cmbMateria.Text + "-" + cmbGr.Text);
125 }
126 else
127 {
128     //se obtiene la lista de registros del profesor de la bd
129     regsAux = proxy.ObtenerRegistros(frmLogin.cliente, 0, frmLogin.semestre, profsAux[cmbProfesor.SelectedIndex].Cedula);
130 }
```

Código 2.76. Proceso para obtener los registros del docente seleccionado

```
133     //se obtiene el horario del profesor
134     GenerarHorarioProf();
135     //se recorre la lista de registros filtrados por el nombre del profesor y obtenidos desde la base
136     foreach (Registro reg in regsAux)
137     {
138         //se comprueba que el registro este entre el rango de fechas elegido por el usuario
139         if (dtFechaInicio.Value.Date <= reg.Fecha.Date && dtFechaFin.Value.Date >= reg.Fecha.Date)
140         {
141             //se instancia un objeto cultureInfo que permitira obtener los nombres de los dias de los registros en español
142             CultureInfo ci = new CultureInfo("ES-es");
143             //se recupera la fecha del registro
144             DateTime dateValue = new DateTime(reg.Fecha.Year, reg.Fecha.Month, reg.Fecha.Day);
145             //se obtiene el nombre del dia en base a la fecha, en el formato especificado
146             string strAux = dateValue.ToString("dddd", ci);
147             strAux = strAux.Replace('é', 'e').Replace('á', 'a');
```

Código 2.77. Filtrado de registros en base a fecha seleccionada

Si los registros son pertenecientes al mismo grupo y corresponden al mismo día son almacenados en la lista auxiliar `rsAux`, la cual mediante la función `LlenarFilaReg` permite obtener el primer y el último registro realizado durante la clase correspondiente a la hora de ingreso, y hora de salida del docente (Código 2.78).

```
150 //recorre el horario del gr del registro
151 for (int j = 0; j < reg.Gr.Horario.Count; j++)
152 {
153     //comprueba que el horario tenga un objeto mas respecto al indice actual para realizar la comparacion siguiente (que exista con que comparar y no salte un error)
154     if ((j + 1) != reg.Gr.Horario.Count)
155     {
156         //se comprueba que  el dia de la ranura que se esta recorriendo sea igual al dia de la siguiente ranura a para comprobar si el horario posee 2 horas de clase
157         if (reg.Gr.Horario[j].Dia == reg.Gr.Horario[j + 1].Dia)
158         {
159             //comprueba que el horario tenga dos objetos mas respecto al indice actual para realizar la comparacion siguiente
160             if ((j + 2) != reg.Gr.Horario.Count)
161             {
162                 //se comprueba que  el dia de la ranura j+1 sea igual al dia de la siguiente ranura para comprobar si el horario posee 3 horas de clase
163                 if (reg.Gr.Horario[j + 1].Dia == reg.Gr.Horario[j + 2].Dia)
164                 {
165                     //se llena la ranura auxiliar con el dia de la ranura que estamos recorriendo
166                     rAux.Dia = reg.Gr.Horario[j].Dia;
167                     //se llena la hora inicio con la hora inicio de la ranura j
168                     rAux.HoraInicio = new DateTime(2019, 1, 1, reg.Gr.Horario[j].HoraInicio.Hour, 0, 0);
169                     //se llena la hora inicio con la hora inicio de la ranura j+2
170                     rAux.Horafin = new DateTime(2019, 1, 1, (reg.Gr.Horario[j].Horafin.Hour + 2), 0, 0);
171                               //se comprueba que el registro se encuentre dentro de las horas de clase
172                               if (strAux == rAux.Dia.ToLower() && reg.Fecha.Hour >= rAux.HoraInicio.Hour && reg.Fecha.Hour <= rAux.Horafin.Hour)
173                               {
174                                   //se llena la fila del registro en el datagrid
175                                   LlenarFilaReg(reg, rsAux, rAux);
176                                   //se avanza al siguiente registro
177                                   break;
178
179                               }
```

Código 2.78. Proceso de comparación horas clase-hora registro

A continuación, se explicará la función `GenerarHorarioProf`.

Inicialmente, se obtiene la lista de grupos de la base de datos mediante la función `ObtenerGrs`, se limpia la lista `horarioProf`, posteriormente se recorre la lista de grupos, y se evalúa si está activado el filtro por materia, en cuyo caso se recupera únicamente el horario de la materia y grupo seleccionados; si el filtro no está activado se recupera el horario completo del profesor (Código 2.79).

En la función `LlenarFilaReg`, se evalúa la lista auxiliar `rsAux`, la cual almacena los registros dentro de un horario de clase, con el fin de segmentar los registros capturados.

Si la lista rsAux está vacía, se agrega el registro de la iteración, y se avanza al siguiente registro para evaluar si se encuentra dentro del mismo horario en cuyo caso es agregado a la lista; cuando la lista tiene más de un registro se realizan los cálculos estadísticos (explicados a continuación), y se toma el primer registro de la lista como registro de entrada, y el último como registro de salida para que la información sea mostrada en pantalla. Cuando el siguiente registro no se encuentra dentro del mismo horario, el proceso se reinicia (Código 2.80).

```
1194    //se obtiene la lista de grupos de la bd
1195    List<Grupo> grsAux = proxy.ObtenerGrs(frmLogin.cliente, frmLogin.semestre);
1196    //se limpia la lista
1197    horarioProf.Clear();
1198    //se obtiene el valor de la cedula del profesor seleccionado en el cmb
1199    string cedula = profsAux[cmbProfesor.SelectedIndex].Cedula;
1200    //si la lista fue recuperada correctamente
1201    if (grsAux != null)
1202    {
1203        //se recorre la lista
1204        foreach (Grupo gr in grsAux)
1205        {
1206            if (chkMateria.Checked)
1207            {
1208                if (gr.Profesor.Cedula == cedula && gr.Recuperacion == 0 && gr.Materia.Nombre == cmbMateria.Text)
1209                {
1210                    horarioProf.Add(gr);
1211                }
1212            }
1213            else
1214            {
1215                //si el gr coincide con un gr del profesor que sea de horario normal
1216                if (gr.Profesor.Cedula == cedula && gr.Recuperacion == 0)
1217                {
1218                    horarioProf.Add(gr);
1219                }
1220            }
1221        }
1222    }
```

Código 2.79. Fragmento función GenerarHorarioProf

Para los cálculos, se crea la clase `ContadorMateria`, la cual permite calcular las horas de asistencia y el tiempo de clase neto para cada grupo del profesor elegido.

Además, se tienen 6 variables globales, 4 totales, y 2 valores de referencia. Dos de estos totales corresponden a la cantidad de horas asistidas a clase y a recuperaciones; y los otros dos corresponden a la cantidad de tiempo neta en las que se dieron clases normales, y de recuperación. Los valores de referencia corresponden a la cantidad de horas que el docente obligatoriamente debió asistir a clase en el periodo ingresado, tanto en horas obligatorias, como de recuperación en caso de que las tenga.

```
1406    //comprueba que la lista de registros auxiliar este vacia
1407    if (rsAux.Count == 0)
1408    {
1409        //si lo esta, agrega el registro enviado
1410        rsAux.Add(reg);
1411    }
1412    //si la lista tiene al menos un registro
1413    else if (rsAux.Count > 0)
1414    {
1415        //comprueba que el registro enviado tenga la fecha del primer registro enviado en la lista
1416        if (rsAux[0].Fecha.Date == reg.Fecha.Date && rsAux[0].Gr.Recuperacion == reg.Gr.Recuperacion)
1417        {
1418            //si coincide, se agrega a la lista
1419            rsAux.Add(reg);
1420        }
1421        //caso contrario
1422        else
1423        {
1424            //se limpia la lista
1425            rsAux.Clear();
1426            //se agrega el registro enviado
1427            rsAux.Add(reg);
1428        }
1429    }
```

Código 2.80. Fragmento función LlenarFilaReg

Conforme se van listando los registros y mostrándose en pantalla, se calcula el total de horas asistidas en base a la suma de las diferencias entre la hora de fin e inicio de clase de los registros listados (Ecuación (2.1)); y el total de tiempo neto que se impartió clases, en base a la suma de las diferencias entre la hora de salida y de entrada (última y primera captura de huella correspondientemente dentro de la hora de clase) de los registros listados (Ecuación (2.2)).

$$totalHoras = \sum_{i=0}^{totalRegistros} registros[i].HoraFinClase - registros[i].HoraInicioClase \quad (2.1)$$

$$totalHorasNetas = \sum_{i=0}^{totalRegistros} registros[i].HoraEntrada - registros[i].HoraSalida \quad (2.2)$$

Para las variables correspondientes a los totales de horas de recuperación, se usa el mismo concepto mostrado en las ecuaciones anteriores.

En el caso de los totales de horas que debió cumplir, se recorren los días dentro del periodo de tiempo seleccionado, buscando coincidencia con los días de los horarios del grupo de la iteración, sumando el número de horas correspondiente. Este proceso se repite para las horas de recuperación a cumplir.

$$horasACumplir = \sum_{i=0}^{diasCoincideEnRango} RanuraHorarioGrupo.HoraFin - RanuraHorarioGrupo.HoraInicio \quad (2.3)$$

Adicionalmente, se calcula el porcentaje correspondiente al rendimiento del docente, utilizando las horas netas de clase (totalHorasNetas) respecto al total de horas obligatorias (horasACumplir).

Para obtener el documento en formato Excel, inicialmente se llama a la función `ExportarExcel`, la cual se explicará a continuación.

En la función, se procede a instanciar la variable `filename` que almacena la ruta donde se encuentra el Excel con el formato deseado. Se instancian varios objetos, entre ellos un objeto tipo aplicación de Excel para proceder a editar, un objeto tipo libro basado en la ruta especificada, y un objeto tipo hoja de cálculo sobre la cual se trabajará. Posteriormente se inserta toda la información en las celdas específicas del documento Excel incluyendo las estadísticas calculadas (Código 2.81).

```
1005    bool exito = false;
1006    //se instancia una variable en la que se guardara la ruta en la que se encuentra el formato
1007    string filename = "";
1008
1009    //se setea la variable con la ruta en base a la ruta de ejecucion del programa
1010    filename = Directory.GetParent(
1011                Directory.GetParent(
1012                Directory.GetParent(AppDomain.CurrentDomain.BaseDirectory).ToString()).ToString()).ToString() +
1013                "\\Resources\\Formato.xlsx";
1014
1015    //se instancia un objeto tipo aplicacion de excel para poder editar la hoja de calculo
1016    Microsoft.Office.Interop.Excel._Application app = new Microsoft.Office.Interop.Excel.Application();
1017    // se crea un nuevo objeto de tipo libro de excel en base a la ruta especificada
1018    Microsoft.Office.Interop.Excel._Workbook libro = app.Workbooks.Open(filename, 0, false, 5, "", "", false,
1019    Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "", true, false, 0, true, false, false); ;
1020    //se crea una nueva hoja de calculo
1021    Microsoft.Office.Interop.Excel._Worksheet hojaCalculo = null;
```

Código 2.81. Fragmento función ExportarExcel

Para obtener el documento en formato PDF se llama a la función `TransformarExcelPdf`, la cual tiene como parámetros de entrada la ruta de un documento Excel auxiliar previamente creado con la información requerida mediante la función `ExportarExcel`, y una ruta previamente creada que será la ruta del documento PDF. La función `TransformarExcelPdf` será explicada a continuación.

Se instancia un objeto tipo aplicación de Excel y un objeto tipo libro de Excel, los cuales servirán para recuperar el documento que se encuentra en la ruta enviada como parámetro de entrada. Posteriormente se procede a exportar este documento en formato PDF insertando la nueva ruta en donde éste será almacenado (Código 2.82).

Para la implementación del submódulo Reportes de Inasistencia, cuya interfaz gráfica es similar a la del submódulo Reportes de Asistencia, es necesario la obtención de las listas de registros, profesores y grupos de la base de datos.

En este submódulo se podrán generar reportes de inasistencia para un docente en particular, dentro de un periodo de tiempo seleccionado. Adicionalmente, los registros de inasistencia del docente pueden estar filtrados por materia y por grupo.

Para iniciar el proceso, el usuario debe llenar los campos profesor y rango de fechas de forma obligatoria, opcionalmente se puede agregar un filtro por materia y por grupo.

```
1523    public bool TransformarExcelPdf(string rutaExcel, string rutaPdf)
1531    // Se crea los objetos de aplicacion de excel y libro de excel
1532    Microsoft.Office.Interop.Excel.Application app;
1533    Microsoft.Office.Interop.Excel.Workbook libro;
1534
1535    // se crea una nueva instancia de la aplicacion de excel
1536    app = new Microsoft.Office.Interop.Excel.Application();
1537
1538    //el proceso se hace invisible para el usuario
1539    app.ScreenUpdating = false;
1540
1541    //el proceso se hace silencioso (sin alertas) para el usuario
1542    app.DisplayAlerts = false;
1543
1544    //se abre el libro de excel que se quiere exportar a pdf
1545    libro = app.Workbooks.Open(rutaExcel);
1563        //se exporta el excel a pdf, y se enviaa la ruta especificada donde se debe guadar
1564        libro.ExportAsFixedFormat(Microsoft.Office.Interop.Excel.XlFixedFormatType.xlTypePDF, rutaPdf);
```

Código 2.82. Fragmento función TransformarExcelPdf

Posteriormente debe presionar el botón consultar para que el sistema devuelva el listado de días no asistidos correspondiente. Este listado podrá ser exportado a un documento formato Excel o PDF.

El código correspondiente se explica a continuación.

Al presionar el botón ***consultar***, se llama a la función `GenerarInasistencias` la cual a su vez llama a la función `ObtenerRegistros`, que recupera los registros del docente de la base de datos y los almacena en la variable `regsAux`.

Posteriormente se recupera el valor de la cédula del profesor, y se recorre la lista `grsAux` que contiene los grupos obtenidos de la base de datos, con el fin de recuperar los grupos a los que dicta clases el docente. Consecuentemente, se recorren los mismos con el fin de establecer la hora inicio y fin de los horarios de clase (Código 2.83).

Se recorren los días dentro del rango seleccionado, así como los horarios de los grupos, y la lista `regsAux`, evaluando que el día de la iteración tenga la misma fecha que el registro de la iteración. Adicionalmente se verifica que el registro se encuentre dentro de la hora

de clase. Si el registro no cumple con todas estas condiciones, implica que el docente no asistió a la clase en esa fecha, ingresando por tanto la fecha en la tabla (Código 2.84).

```
//se obtiene la lista de grupos de la bd
List<Grupo> grsAux = proxy.ObtenerGrs(frmLogin.cliente, frmLogin.semestre);
//se limpia la lista
horarioProf.Clear();
//se obtiene el valor de la cedula del profesor seleccionado en el cmb
string cedula = profsAux[cmbProfesor.SelectedIndex].Cedula;
//si la lista fue recuperada correctamente
if (grsAux != null)
{
    //se recorre la lista
    foreach (Grupo gr in grsAux)
    {
        //si el chk esta chequeado
        if (chkMateria.Checked)
        {
            //y la cedula del profesor coincide con la cedula del profesor de la iteracion, es un gr normal, y de la materia
            //seleccionada  en el cmb
            if (gr.Profesor.Cedula == cedula && gr.Recuperacion == 0 && gr.Materia.Nombre == cmbMateria.Text)
            {
                //se añade el gr a una lista
                horarioProf.Add(gr);
            }
        }
        else
        {
            //si el gr coincide con un gr del profesor que sea de horario normal
            if (gr.Profesor.Cedula == cedula && gr.Recuperacion == 0)
            {
                //se añade a la lista
                horarioProf.Add(gr);
            }
        }
```

Código 2.83. Fragmento función GenerarInasistencias

```
//se recorre los dias del rango
foreach (DateTime d in range)
{
    //se realizan las tranformaciones del dia del rango al formato requerido
    string strAux = d.ToString("dddd", ci);
    strAux = strAux.Replace('é', 'e').Replace('á', 'a');

    //booleano que nos permitira conocer las fechas de inasistencia
    bool fechaAsistida = false;

    if ((new Utilidades().ComprobarFechaNoFeriado(d)))
    {
        //se recorre el horario del profesor
        foreach (Grupo g in horarioProf)
        {
            //se recorren las ranuras del horario
            foreach (RanuraTiempo r in g.Horario)
            {
                // se compara los dias dentro del rango con los dias del horario del grupo
                if (strAux == r.Dia.ToLower())
                {
                    //se recorren los registros
                    foreach (Registro reg in regsAux)
                    {
                        //si la fecha del registro de iteracion coincide con la fecha del dia del rango de iteracion
                        if (d.Date == reg.Fecha.Date)
                        {
                            //y la hora del registro se encuentra dentro del horario de clase del dia del grupo
                            if (r.HoraInicio.Hour <= reg.Fecha.Hour && r.Horafin.Hour >= reg.Fecha.Hour)
                            {
                                //el profesor asistio a clases
                                fechaAsistida = true;
                                //sale del lazo
                                break;
                            }
                        }
                    }
```

Código 2.84. Algoritmo para generar lista de Inasistencia

Para exportar la lista de fechas de inasistencia tanto a un documento Excel como a PDF, el proceso es el mismo que el explicado anteriormente, la diferencia principal radica en la posición de las celdas para la inserción de datos en el documento.

2.3.10.3 Entregable

El entregable de este *sprint* consiste en el submódulo Reportes de Asistencia completamente funcional.

2.3.11 SPRINT 11

En este *sprint* se implementará el submódulo Reportes de Aulas que se encuentra dentro del módulo Reportes, en el cual se generarán reportes sobre la utilización de las aulas.

2.3.11.1 Diseño

El proceso de generación de reportes de utilización de aulas a través de la aplicación de administración se presenta en el diagrama de secuencia de la Figura 2.61.

2.3.11.2 Implementación

Para la implementación del submódulo Reportes de Aulas cuya interfaz gráfica es similar a la del submódulo Reportes de Asistencia, es necesario la obtención de las listas de registros y aulas de la base de datos, así como la creación de distintas variables que serán utilizadas para el cálculo de estadísticas y obtención de datos.

En este submódulo se podrán generar reportes de utilización para un aula en particular, dentro de un periodo de tiempo seleccionado. Para iniciar el proceso, el usuario debe llenar los campos aula y rango de fechas de forma obligatoria.

Posteriormente debe presionar el botón ***consultar*** para que el sistema devuelva el listado de registros de clases dictadas en el aula, con las estadísticas resultantes del cálculo realizado. Este listado podrá ser exportado a un documento formato Excel o PDF. El código correspondiente se explica a continuación.

Al presionar el botón ***consultar***, se configuran las variables correspondientes en cero, y se llama a la función `ObtenerRegistros`, que recupera los registros realizados en el aula desde la base de datos y los almacena en la variable `regsAux`.

Se recorre la lista `regsAux` comprobando que los registros se encuentren dentro del periodo de tiempo seleccionado; posteriormente se recorre el horario del grupo correspondiente evaluando el número de horas de clase por día. Mediante esto se conocerá la hora inicio y hora fin de la clase, y se recuperarán en cada iteración los registros que se hayan realizado dentro del horario de clase.

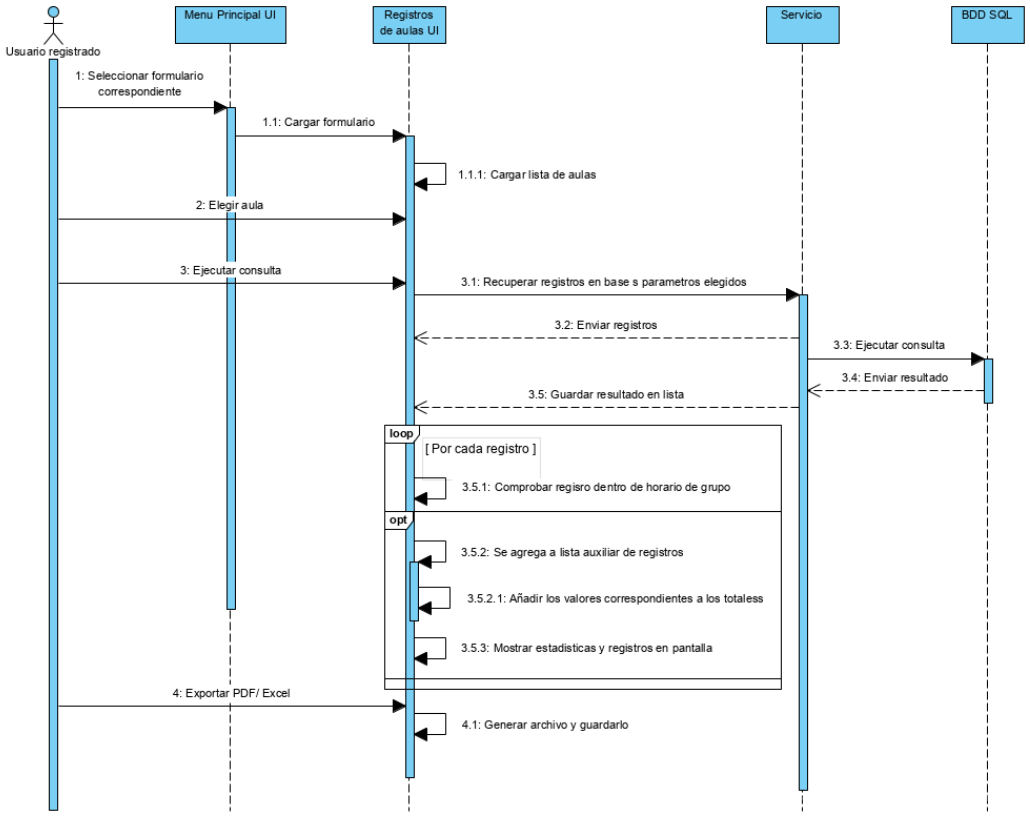


Figura 2.61. Diagrama de secuencia de submódulo Reportes de Aulas

```
94      //se obtiene la lista de registros de la bd
95      regsAux = proxy.ObtenerRegistros(frmLogin.cliente, 1, frmLogin.semestre,
96  aulasAux[cmbAulas.SelectedIndex].NombreAu);
```

Código 2.85. Obtención de los registros realizados en el aula seleccionada

```
119 //recorre el horario del gr del registro
120 for (int j = 0; j < reg.Gr.Horario.Count; j++)
121 {
122     //comprueba que el horario tenga un objeto mas respecto al indice actual para realizar la comparacion siguiente
123     if ((j + 1) != reg.Gr.Horario.Count)
124     {
125         //se comprueba que  el dia de la ranura que se esta recorriendo sea igual al dia de la siguiente ranura a para comprobar si el horario posee 2 horas
126         if (reg.Gr.Horario[j].Dia == reg.Gr.Horario[j + 1].Dia)
127         {
128             //comprueba que el horario tenga dos objetos mas respecto al indice actual para realizar la comparacion siguiente
129             if ((j + 2) != reg.Gr.Horario.Count)
130             {
131                 //se comprueba que  el dia de la ranura j+1 sea igual al dia de la siguiente ranura para comprobar si el horario posee 3 horas de clase
132                 if (reg.Gr.Horario[j + 1].Dia == reg.Gr.Horario[j + 2].Dia)
133                 {
134                     //se llena la ranura auxiliar con el dia de la ranura que estamos recorriendo
135                     rAux.Dia = reg.Gr.Horario[j].Dia;
136                     //se llena la hora inicio con la hora inicio de la ranura j
137                     rAux.HoraInicio = new DateTime(2019, 1, 1, reg.Gr.Horario[j].HoraInicio.Hour, 0, 0);
138                     //se llena la hora inicio con la hora inicio de la ranura j+2
139                     rAux.Horafin = new DateTime(2019, 1, 1, (reg.Gr.Horario[j].Horafin.Hour + 2), 0, 0);
140                     //se comprueba que el registro se encuentre dentro de las horas de clase
141                     if (strAux == rAux.Dia.ToLower() && reg.Fecha.Hour >= rAux.HoraInicio.Hour && reg.Fecha.Hour <= rAux.Horafin.Hour)
142                     {
143                         //se llena la fila del registro en el datagrid
144                         LlenarFilaReg(reg, rsAux, rAux);
145                         //se avanza al siguiente registro
146                         break;
147 
148                     }
```

Código 2.86. Proceso de comparación horas clase-hora registro

En la función `LlenarFilaReg`, se evalúa la lista auxiliar `rsAux`, la cual almacena los registros dentro de un horario de clase con el fin de segmentar los registros capturados durante una misma clase.

Si la lista `rsAux` está vacía, se agrega el registro de la iteración, y se avanzará al siguiente registro para evaluar si se encuentra dentro del mismo horario en cuyo caso es agregado a la lista; cuando la lista tiene más de un registro se realizan los cálculos estadísticos (explicados a continuación), y se toma el primer registro de la lista como registro de entrada, y el último como registro de salida para que la información sea mostrada en pantalla. Cuando el siguiente registro no se encuentra dentro del mismo horario, el proceso se reinicia (Código 2.87).

Para los cálculos, se tienen 4 variables globales: Dos variables para las horas en las que el aula debió haber sido utilizada tanto para las horas obligatorias de clase como para las horas de recuperación, una variable para las horas netas que el aula ha sido utilizada, y una variable para las horas que el aula está disponible.

```
908     //comprueba que la lista de registros auxiliar este vacia
909     if (rsAux.Count == 0)
910     {
911         //si lo esta, agrega el registro enviado
912         rsAux.Add(reg);
913     }
914     //si la lista tiene al menos un registro
915     else if (rsAux.Count > 0)
916     {
917         //comprueba que el registro enviado tenga la fecha del primer registro enviado en la lista
918         if (rsAux[0].Fecha.Date == reg.Fecha.Date)
919         {
920             //si coincide, se agrega a la lista
921             rsAux.Add(reg);
922         }
923         //caso contrario
924         else
925         {
926             //se limpia la lista
927             rsAux.Clear();
928             //se agrega el registro enviado
929             rsAux.Add(reg);
930         }
931     }
```

Código 2.87. Fragmento función LlenarFilasReg

En el caso de las variables en las que el aula debió haber sido utilizada, se recorren los días dentro del periodo de tiempo seleccionado, buscando coincidencia con los días de los horarios del grupo de la iteración, sumando el número de horas correspondiente. Este proceso se repite para las horas a cumplir de recuperación (Ecuación (2.4)).

$$horasClaseDesignadas = \sum_{i=0}^{diasCoincideEnRango} RanuraHorarioGrupo.HoraFin - RanuraHorarioGrupo.HoraInicio \quad (2.4)$$

Para las horas netas que el aula ha sido utilizada, conforme se van listando los registros y mostrándose en pantalla, se calcula el total de horas asistidas en base a la suma de las diferencias entre la hora de fin e inicio de clase (Ecuación (2.5)).

$$horasUtilizadas = \sum_{i=0}^{totalRegistros} registros[i].HoraFin - registros[i].HoraInicio \quad (2.5)$$

Para las horas en que el aula se encuentra disponible, se recorren los días dentro del periodo de tiempo seleccionado, y por cada día hábil se adicionan las horas totales posibles para la ocupación del aula (Ecuación (2.6)).

$$horasPosibleOcupacion = \sum_{i=0}^{diasEnRango} horasDiariasPosibles \quad (2.6)$$

Adicionalmente, se calcula el porcentaje correspondiente al rendimiento del aula, utilizando el total de horas utilizadas respecto a la cantidad de horas que el aula puede ser ocupada durante periodo del tiempo seleccionado.

Para exportar la lista de registros tanto a un documento Excel como a PDF, el proceso es el mismo que el explicado en el *sprint* 10; la diferencia principal radica en la posición de las celdas para la inserción de datos en el documento.

2.3.11.3 Entregable

El entregable en este *sprint* consiste en el submódulo Reportes de Aulas completamente funcional.

2.3.12 SPRINT 12

En este *sprint* se implementarán los componentes gráficos necesarios para la realización de la aplicación móvil.

2.3.12.1 Diseño

El diseño gráfico de cada uno de los componentes se presenta en la Figura 2.62.

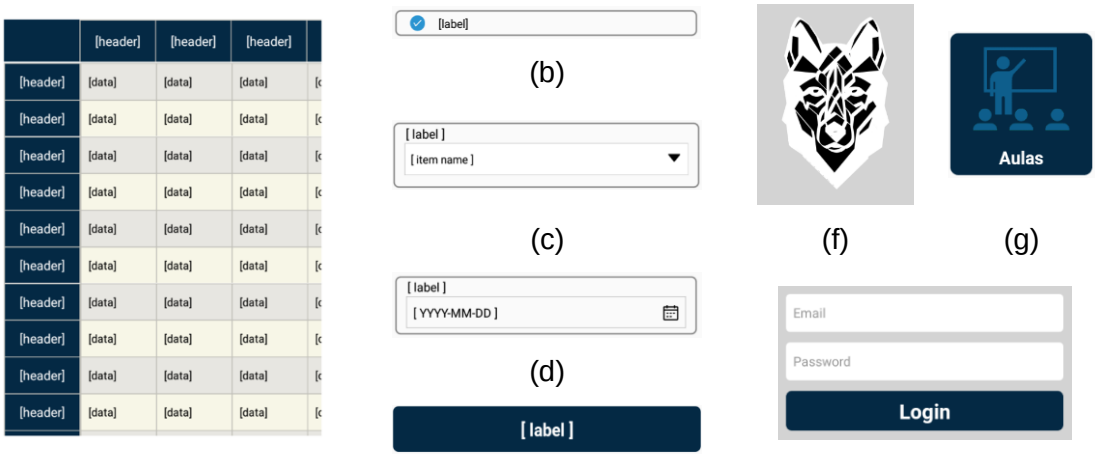


Figura 2.62. Bosquejo de los componentes (a) DataTable, (b) Checkbox, (c) Picker, (d) DateTimePicker, (e) Button, (f) Logo, (g) ImageButton, (h) EmailAndPassword

2.3.12.2 Implementación

Para la implementación de toda la aplicación móvil, se instalan las dependencias listadas a continuación:

- `react-native-elements`: Esta dependencia contiene varios componentes altamente personalizables, entre éstos el que se usará en este proyecto, un `CheckBox`.
- `react-native-firebase`: Esta dependencia permite la conexión con Firebase y realizar múltiples operaciones con todos los servicios que ofrece.
- `react-native-table-component`: Contiene los componentes necesarios para generar una tabla.
- `react-native-vector-icons`: Contiene un sinnúmero de iconos disponibles gratis para el desarrollo de aplicaciones.
- `react-navigation`: Permite crear una estructura de navegación para la aplicación.
- `react-navigation-drawer`: Permite crear un patrón para tener un menú que aparece en la parte izquierda de la aplicación.
- `react-redux`: Permite crear enlaces entre React y Redux y crear componentes “inteligentes” que escuchan los cambios de estado y re-renderiza los componentes dependiendo de estos cambios.
- `redux`: Permite implementar un contenedor de estado denominado *store* para aplicaciones hechas en Javascript.

- `redux-thunk`: Crea un *middleware* que genera creadores de acciones para devolver acciones. El *thunk* permite ralentizar el despachador de acciones, o despacharla siempre y cuando las condiciones se cumplan, o dichas acciones hayan terminado.

Para la creación del componente `Button`, se toman los componentes `TouchableOpacity` y `Text` con los cuales se podrá añadir una animación al botón, además de una etiqueta y el evento a ejecutarse a través de las *props* (Código 2.88). Las *props* son atributos que se asignan a un componente en React.

```
14  <TouchableOpacity
15    style = {styles.buttonContainer}
16    onPress = {this.props.onButtonPress}>
17    <Text style = {styles.buttonText}>
18      {this.props.label}
19    </Text>
20  </TouchableOpacity>
```

Código 2.88. Implementación del componente Button

Para la implementación del `CheckBoxContainer`, se hace uso del componente `CheckBox` de la librería `react-native-elements`, el cual es altamente editable por el número de propiedades que posee como se observa en el Código 2.89. Adicionalmente por medio de las *props* se le enviará su etiqueta, valor de *checked* y el evento a accionarse.

```
15  <View style = {styles.checkBoxContainer}>
16    <CheckBox
17      size = {22}
18      containerStyle = {styles.checkBox}
19      checked = {this.props.checked}
20      iconType='antdesign'
21      checkedIcon='checkcircle'
22      uncheckedIcon = 'checkcircleo'
23      checkedColor='#2a94d1'
24      onPress = {this.props.onPress}
25    />
26    <Text style = {styles.checkBoxLabel}>
27      {this.props.label}
28    </Text>
```

Código 2.89. Implementación del componente CheckBoxContainer

Para el componente `DataTable`, se importan los componentes `Table` y `Row` de `react-native-table-component`, y `ScrollView` de `react-native`. Se controla la visibilidad de las cabeceras con un *booleano* para la correspondiente a las filas, y otro para las columnas. Es necesario señalar que los datos a mostrar en la tabla, se pasan a través de *props*. Se muestra una parte de la implementación de la tabla, y el mapeo de datos dentro de ésta en el Código 2.90.

```
<Table
  borderStyle={{
    borderWidth: 1,
    borderColor: '#C1C0B9'
  }}
>
  {
    rowHeader.map((row, index) => (
      <Row
        key={index}
        data={row}
        widthArr={rowHeaderWidth}
        style={[
          styles.tableHeader,
          {
            height: 35 * resizeFactor
          }
        ]}
        textStyle={styles.tableHeaderText}
      />
    ))
  }
</Table>
```

Código 2.90. Implementación del componente Table

El componente `DateTimeBox` permite accionar el `DateTimePicker` que se encuentra en el contenedor, además de mostrar el parámetro recuperado en pantalla. También cuenta con un ícono que actúa como botón, el cual permitirá ejecutar la acción que se le asigne, cuyo código se muestra en el Código 2.91.

```
<TouchableWithoutFeedback
  onPress = {this.props.onDateChange}
>
  <Icon
    name = {iconType}
    color = 'black'
    size = {25}
    style= {styles.icon}
  />
</TouchableWithoutFeedback>
```

Código 2.91. Implementación de un ícono como botón

El componente `ImageButton` funciona de manera similar a `Button`, con la diferencia que tiene agregado el componente `Image`, al cual se le cargará la imagen correspondiente a la definida por las *props* del mismo. En el Código 2.92 se muestra la implementación del componente `Image`.

Este componente `Image`, también es usado para el componente `Logo`, que utiliza el mismo código, pero con la imagen del logo como predeterminada.

```
<Image
  source = {this.props.icon}
  style={styles.iconButton}
/>
```

Código 2.92. Implementación del componente Image

Para el componente `PickerBox` se usó el `Picker` de React Native además de un ícono de flecha hacia abajo. El mapeo de los ítems se hace de manera similar que para la tabla, como se puede apreciar en el Código 2.93.

```
54  {this.props.pickerItems.map( (s, i) => {
55    return <Picker.Item
56      key={i} value={s} label={s.replace('*','/')} />
57    })
58  }
```

Código 2.93. Mapeo de elementos para el Picker

Para el componente `FadeInView`, se importa `Animated` y `useState`, este último se usa para tener un estado, en un componente sin éste, y las variables puedan ser conservadas por React Native; en este caso se inicializa el valor de la animación como se muestra en Código 2.94.

```
7   const [fadeAnim] = useState(new Animated.Value(0))
```

Código 2.94. Implementación de useState

Seguidamente con la ayuda de `useEffect`, que es una función que permite añadir a un componente efectos adicionales, se inicia la función `timing` de `Animated`, pasándole como argumento de entrada el valor inicial de opacidad, el valor al que se quiere llegar, el tiempo de la animación y activando el uso del *driver* nativo, el cual permite tener una animación más pulida y evitar pérdida de *frames*.

```
11  Animated.timing(
12      fadeAnim,
13      {
14      toValue: 1,
15      duration: 1000,
16      useNativeDriver: true,
17      }
18  ).start();
```

Código 2.95. Implementación de la función timing del componente Animated

Finalmente, se retorna un `Animated.View` como se muestra en el Código 2.96. Este componente se usa en la pantalla `Login`, para tener una transición más suave entre la pantalla `Presentation` y esta última.

El componente `MenuHeader` cuenta con dos botones de íconos, uno de ellos permite mostrar el menú con la función `toggleDrawer`, y el otro permite cerrar la sesión con la función que se creará más adelante.

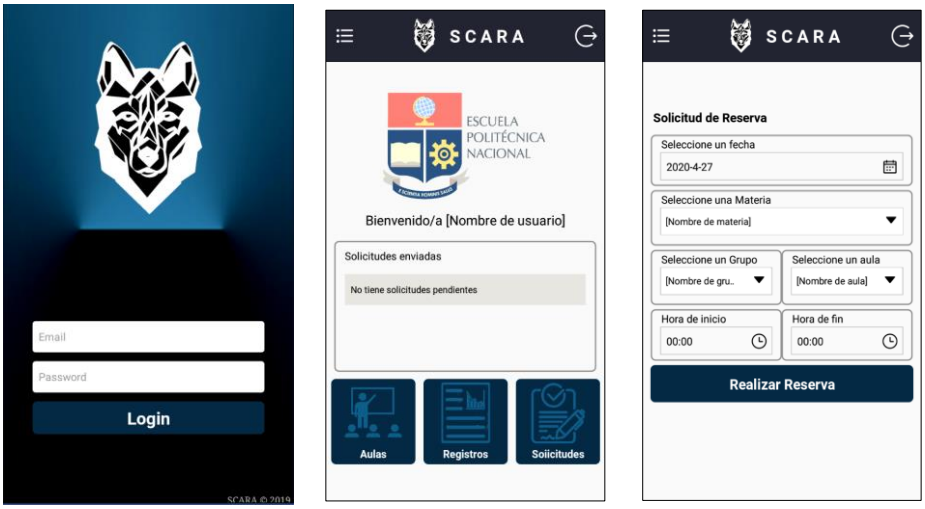
```
22  return (
23    <Animated.View
24      style={{
25        ...props.style,
26        opacity: fadeAnim,
27      }}
28    >
29      {props.children}
30    </Animated.View>
31  );
```

Código 2.96. Implementación del componente Animated.View

Para la creación del resto de pantallas, se empiezan a colocar los componentes en base a los diseños ya establecidos, y enviando las *props* necesarias.

2.3.12.3 Entregable

El entregable para este *sprint* son las interfaces gráficas ya terminadas, una muestra de las pantallas se puede observar en la Figura 2.63.



(a) (b) (c)

Figura 2.63. Interfaz gráfica de las pantallas (a) Ingreso, (b) Principal (c) Solicitudes.

2.3.13 SPRINT 13

En este *sprint* se crearán las funciones para el ingreso y salida de la aplicación móvil, además de la navegación entre las pantallas correspondientes junto con el menú.

2.3.13.1 Diseño

El proceso para el ingreso del usuario a la aplicación móvil y la navegación a través de esta, se presenta en el diagrama de actividades de la Figura 2.64.

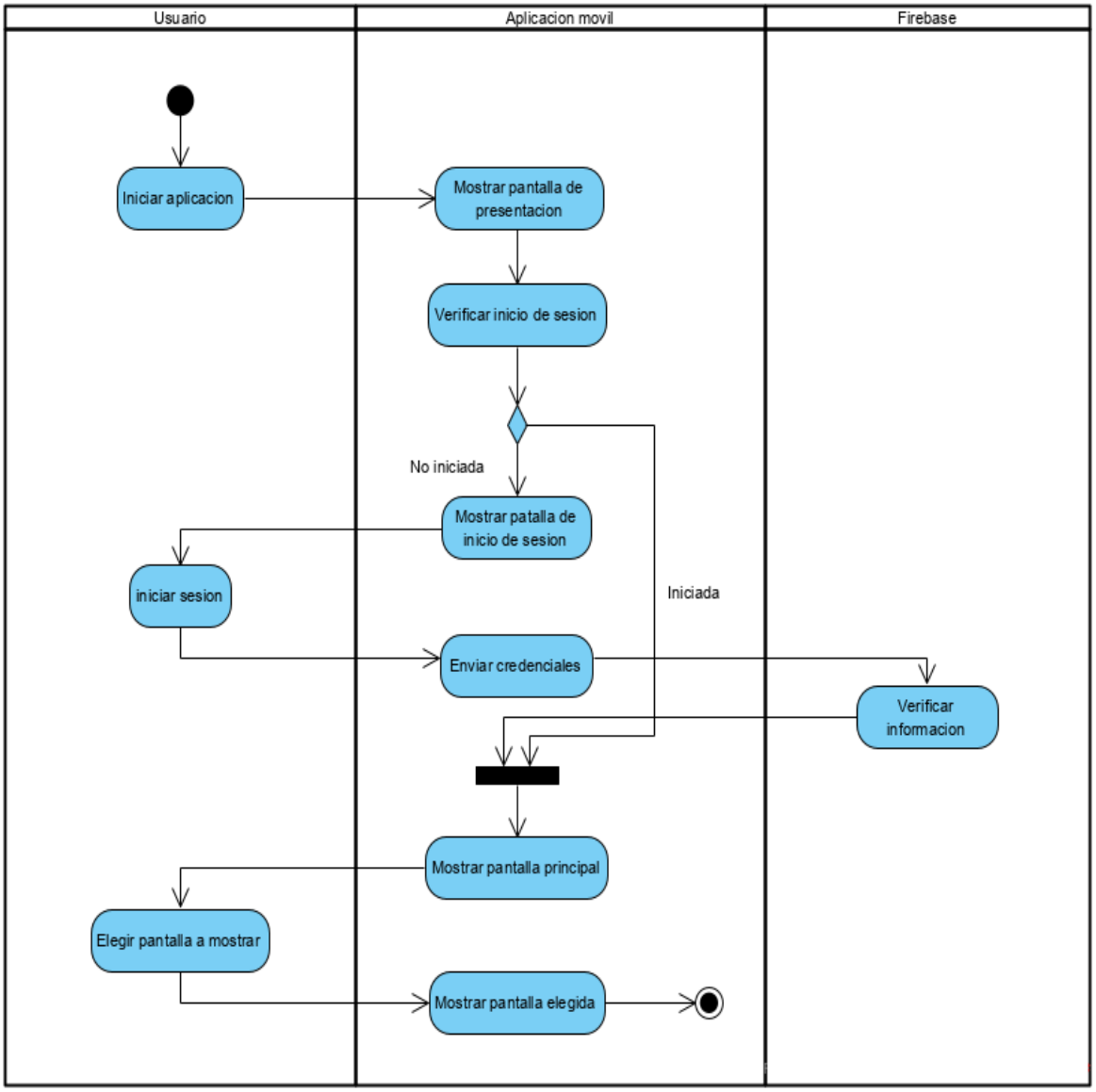


Figura 2.64. Diagrama de Actividades para ingreso y navegación de la aplicación móvil

2.3.13.2 Implementación

Se trabaja en primer lugar sobre el archivo `SCARAMain.js` ya que a partir de éste se mostrará la pantalla de `Login` o la principal de la aplicación, dependiendo de si se ha ingresado o no.

Para poder comprobar si ha ingresado o no, se empieza por la implementación de Redux en el proyecto; para ello se inicia por crear los archivos `authActions` y `authReducers`, que servirán para almacenar los accionadores y reductores para la autenticación de la aplicación. Además se crea también el archivo `rootReducer` mediante el cual permitirá re-direccionar a los distintos reductores que tendrá la aplicación con la ayuda de `combineReducers`; esto se muestra en el Código 2.97.

```
import authReducer from './authReducer'
import {combineReducers} from 'redux'

const rootReducer = combineReducers({
    auth: authReducer,
})

export default rootReducer;
```

Código 2.97. Archivo rootReducer.js

Seguidamente en el archivo `index.js`, se define la *store* de la aplicación, para el efecto se importa: `thunk` para poder usar los métodos de Firebase en las acciones que lo necesiten, `applyMiddleware`, y `createStore`. Esto se puede observar en el Código 2.98.

```
import { createStore, applyMiddleware } from 'redux'
import rootReducer from './reducers/rootReducer'
import thunk from 'redux-thunk'
import firebase from 'react-native-firebase';

export default store = createStore(
  rootReducer,
  applyMiddleware(
    thunk.withExtraArgument({firebase})
  )
);
```

Código 2.98. Archivo index.js de la carpeta *store*

Para la implementación de las acciones de autenticación se definió la constante `authErrors` dentro del archivo `constants` para así poder ver qué tipo de error se efectuó y poder informar al usuario al respecto. Esta constante se muestra en el Código 2.99.

Se tienen 3 acciones `login`, `logout` e `isLoggedIn`. La acción `login` necesita como parámetro de entrada las credenciales para ingresar al sistema, y ésta retornará una función que toma como parámetros de entrada el *dispatcher* que es el que comunica al *reducer* el resultado de la operación, `getState` que permite regresar al árbol de estado y el argumento extra añadido en *thunk* entre llaves, en este caso, Firebase. Esta estructura de la función a retornar, se cumple para el resto de acciones a implementarse (Código 2.100).

```
const authErrors = {
  error0: 'The email address is badly formatted.',
  error1: 'The password is invalid or the user does not have a password.',
  error2: 'There is no user record corresponding to this identifier. The user may have been deleted.',
  error3: 'An internal error has occurred. [ 7: ]',
}
```

Código 2.99. Mensajes de error en autenticación

```
3   export const login = (credentials) => {
4     return (dispatch, getState, {firebase}) => {
```

Código 2.100. Fragmento de función login

Dentro de la función, se llama a `firebase.auth().signInWithEmailAndPassword` y se envían el *email* y contraseña presentes en el objeto `credentials`; si la operación tuvo éxito, se envía en el *dispatcher* un `AUTH_SUCCESS`, caso contrario, se discernirá cual fue el error, y se enviará éste junto a un `AUTH_ERROR` como se muestra en el Código 2.101.

```
12  .catch((err) =>
13    {
14      var errorMessage = '';
15      if(err.message === authErrors.error0)
16      {
17        errorMessage = 'Email con formato invalido'
18      }
19      else if(err.message === authErrors.error1 ||
20              err.message === authErrors.error2)
21      {
22        errorMessage = 'Correo o contrasenia incorrectos'
23      }
24      else if(err.message === authErrors.error3)
25      {
26        errorMessage = 'No posee conexion a internet'
27      }
28      else
29      {
30        errorMessage = err.message
31      }
32      dispatch({type: 'AUTH_ERROR', errorMessage})
33    })
```

Código 2.101. Proceso para discernir mensaje de error en autenticación

Para la acción `logout`, simplemente se llama a `firebase.auth().signOut()` para cerrar la sesión, y en el *dispatcher* se envía un `LOGOUT_SUCCESS`; se puede observar la operación en el Código 2.102.

```
37  export const logout = () => {
38    return (dispatch, getState, {firebase}) => {
39      firebase.auth().signOut().then(()=>{
40        dispatch({type: 'LOGOUT_SUCCESS'})
41      })
42    }
```

Código 2.102. Función logout

La acción `isLoggedIn` permite comprobar al inicio de la aplicación si se ha iniciado o no sesión, y así mostrar las pantallas correspondientes. A través de la función `firebase.auth().onAuthStateChanged` se comprueba el estado de la sesión, si se encuentra activa, se recuperan los datos del usuario y se envía en el *dispatcher* un

`LOGGED_IN`, si no, se envía un `LOGOUT_SUCCESS`. El código de esta función se muestra en el Código 2.103.

En el *reducer* se decide cómo actualizar el estado dependiendo de lo enviado en el *dispatcher*. Un ejemplo de esto se muestra en el Código 2.104, en donde se inicializa primero el estado a enviar y se denomina *action* a lo recibido por el *dispatcher*. Seguidamente con una sentencia *switch*, se discierne qué tipo de respuesta se obtuvo con la acción ejecutada, para así cambiar el valor de la variable de estado correspondiente.

```
export const isLoggedIn = () => {
  return (dispatch, getState, {firebase}) => {
    firebase.auth().onAuthStateChanged((user) => {
      if(user)
      {
        var userName = firebase.auth().currentUser.displayName;
        var userPhotoUrl = firebase.auth().currentUser.photoURL;

        var userInfo = {
          name: userName,
          photoUrl: userPhotoUrl
        }
        dispatch({type: 'LOGGED_IN', userInfo});
      }
      else
      {
        dispatch({type: 'LOGOUT_SUCCESS'})
      }
    });
  }
}
```

Código 2.103. Función isLoggedIn

```
7   const authReducer = (state = initState, action) => {
8     console.log(action)
9     switch(action.type){
10      case 'AUTH_SUCCESS':
11        console.log('Usuario ingresado');
12        return {
13          ...state,
14          authError: false,
15        };
16      case 'AUTH_ERROR':
17        console.log('error')
18        console.log(action.errorMessage)
19        return {
20          ...state,
21          authError: true,
22          errorMessage: action.errorMessage
23        };
24      case 'LOGGED_IN':
25        console.log('isloggedin')
26        return {
27          ...state,
28          loggedIn: true,
29          userInfo: action.userInfo
30        };
```

Código 2.104. Fragmento de función authReducer

Para poder acceder a la *store* de Redux, primero se importa la función `connect` de `react-redux`, la cual permite establecer la conexión entre el componente y la *store*. A esta función, se le pueden enviar dos funciones más como argumentos de entrada `mapStateToProps` y `mapDispatchToProps`.

La función `mapStateToProps` permite definir como *prop* la variable de estado que se requiera por el componente, y `mapDispatchToProps` permite definir como *prop* cualquiera de las acciones a ser utilizadas por el componente; un ejemplo de su implementación se puede observar en el Código 2.105.

La función `login` permitirá validar el ingreso de sesión por parte del usuario, mientras que la función `isLoggedIn` permitirá que se carguen las pantallas correspondientes; esta función se llamará tanto al inicio de la aplicación como cuando el ingreso haya sido exitoso.

En caso de que la función `login` no devuelva un resultado favorable, este hecho se le informa al usuario a través de un `Alert` que se construye a través de la función `showAlert` mostrada en el Código 2.106.

```
191  const mapStateToProps = (state) => {
192      return {
193        authError: state.auth.authError,
194        errorMessage: state.auth.errorMessage,
195        loggedIn: state.auth.loggedIn,
196      }
197  }
198
199  const mapDispatchToProps = (dispatch) =>{
200    return{
201      login: (credentials) => dispatch(login(credentials)),
202      isLoggedIn: () => dispatch(isLoggedIn()),
203    }
204  }
205
206  export default connect(mapStateToProps, mapDispatchToProps)(EmailAndPassword);
```

Código 2.105. Fragmento de componente EmailAndPassword

```
69   showAlert = (title, message) => {
70     Alert.alert(
71       title,
72       message,
73       [
74         {
75           text: "OK"
76         }
77       ],
78       {
79         cancelable: false
80       }
81     );
```

Código 2.106. Función showAlert

La función `logout` se acciona a través del botón correspondiente en la cabecera del menú, no sin antes preguntar al usuario si desea o no realizar dicha acción.

Para el caso de la navegación, se crea el componente `DrawerNavigator`, en el cual se colocan las distintas rutas para poder re-direccionar hacia las distintas pantallas disponibles. Primero se crea el objeto de configuración `DrawerConfig` en el cual se definen tres propiedades, el alto, ancho y el componente que renderiza el menú, que para este caso es el componente `MenuDrawer`, el cual se describirá más adelante.

A continuación, para la definición de `DrawerNavigator`, se usa la función `createDrawerNavigator` la cual recibe como argumentos las rutas, y la configuración; seguidamente se los exporta con la función `createAppContainer` la cual permite crear un estado para la navegación. Su implementación se puede observar en el Código 2.107.

```
23  const DrawerNavigator = createDrawerNavigator(
24    {
25      Home:{
26        screen: HomeScreen
27      },
28      ListaAulas:{
29        screen: ClassroomList
30      },
31      Registros:{
32        screen: AttendanceRecord
33      },
34      Solicitud:{
35        screen: ReservationRequest
36      }
37    },
38    DrawerConfig
39  )
40
41  export default createAppContainer(DrawerNavigator);
```

Código 2.107. Fragmento componente DrawerNavigator

Para mostrar el menú que permitirá accionar la navegación hacia las distintas pantallas disponibles, se implementa el componente `MenuDrawer`. Dentro de éste, se crea la función `navLink`, la cual genera un componente al cual se le asigna la ruta correspondiente, y la etiqueta a mostrar en el menú. Se puede ver la función y su implementación en el Código 2.108.

Además, se obtiene desde la *store* la información del usuario necesaria para mostrar tanto su foto de perfil, como su nombre en la parte superior del menú. Finalmente se coloca el nombre de la aplicación al pie del menú, junto a su versión.

2.3.13.3 Entregable

El entregable de este *sprint* consiste en la apertura y cierre de sesión en la aplicación, además de la navegación.

```
29    navLink(nav, text) {
30      return(
31        <TouchableOpacity
32          style={{height: 60*resizeFactor}}
33          onPress={() => this.props.navigation.navigate(nav)}
34        >
35          <Text style={styles.link}>{text}</Text>
36        </TouchableOpacity>
37      )
38    }

62    <View style={styles.bottomLinks}>
63      {this.navLink('Home', 'Inicio')}
64      {this.navLink('ListaAulas', 'Horarios de Aulas')}
65      {this.navLink('Registros', 'Registros de Asistencia')}
66      {this.navLink('Solicitud', 'Solicitud de reserva')}
67    </View>
```

Código 2.108. Función navLink e implementación

2.3.14 SPRINT 14

En este *sprint* se realizará todo lo relacionado a la obtención de información desde Firebase con la aplicación móvil.

2.3.14.1 Diseño

El diagrama de secuencia para las distintas operaciones de obtención de datos en la aplicación móvil se muestra en la Figura 2.65.

2.3.14.2 Implementación

Para la recuperación de datos de Firebase se implementan varias acciones en los archivos correspondientes, por ello se crean archivos para accionadores y reductores de inicialización, horarios, registros de asistencia y solicitudes de reserva.

En la parte de inicialización, se carga la información que será mostrada en los `Picker`, y las solicitudes enviadas por el profesor en la pantalla principal; para esto se tiene una única acción `getData` con la cual se recuperan las listas de aulas, grupos y materias del profesor.

En lo que respecta al acceso de datos dentro de Firebase, se usa `ref`, donde se coloca el valor a consultar, adicionalmente sobre esta función se usa `on` para obtener una captura de datos en ese instante, y para usar estos datos, se usa la función `val`. Un ejemplo de esto se muestra en el Código 2.109.

Una vez obtenidos los datos, se obtienen los nombres de las aulas y se añaden al objeto `classroomList`, luego se recorren los grupos en cada aula, buscando los grupos correspondientes al usuario y se los guarda en la variable `groupList`; además, en esta

misma operación se recuperan también los nombres de las materias que imparte el profesor en la variable `subjectList`, y todas éstas se guardan en una variable para ser enviadas en el *dispatcher*.

Para obtener las solicitudes del profesor, se realiza la consulta en Firebase y de igual forma, en base al id del usuario, se las recupera y se muestran solo las correspondientes a una fecha igual o mayor a la que se haga la consulta. Para esto se genera un arreglo en el cual se presentan los datos de la solicitud, además del estado de la misma en base a las condiciones presentadas en el Código 2.110.

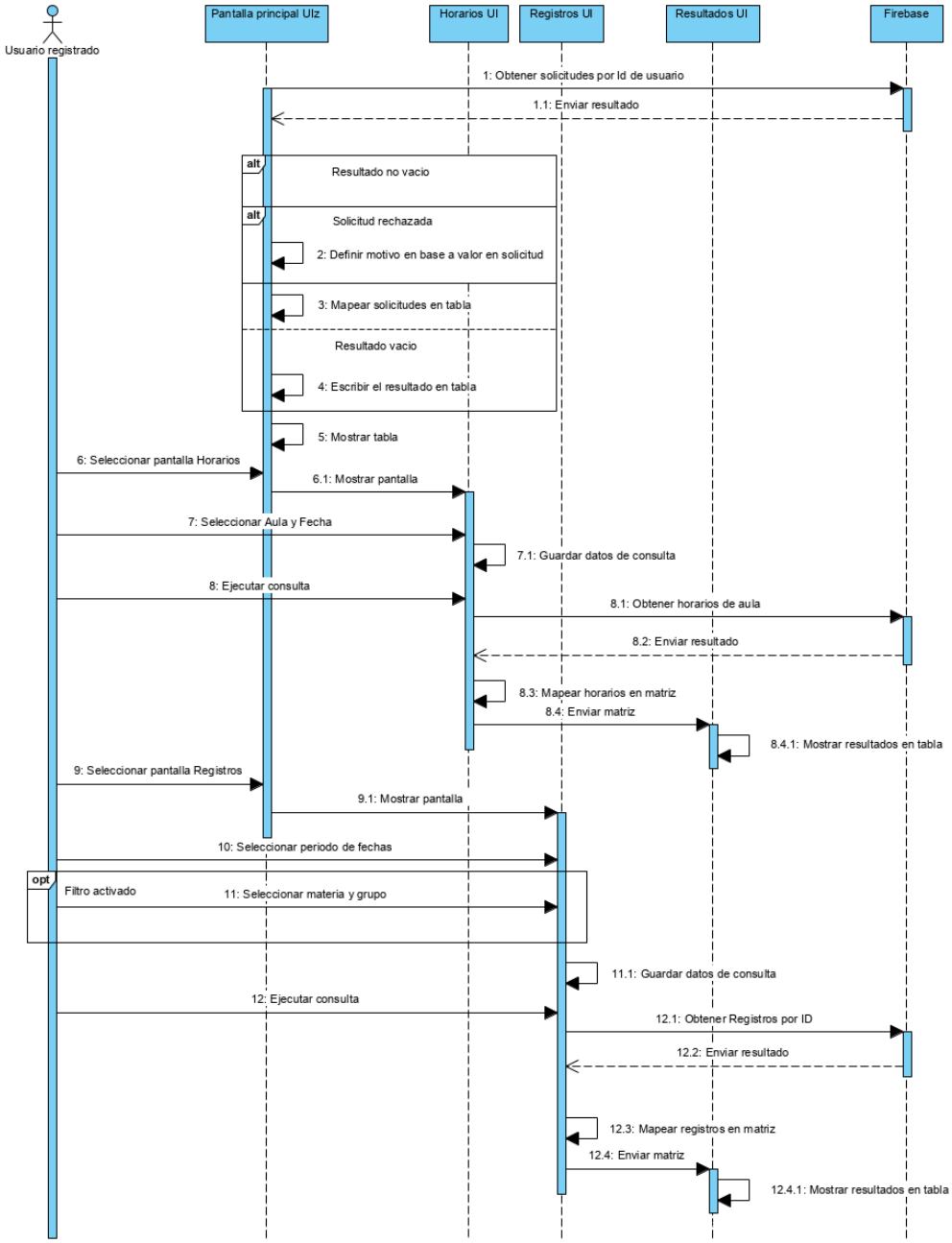


Figura 2.65. Diagrama de Secuencia de los distintos modos de obtención de información desde Firebase a la aplicación móvil

```
9   database.ref('Aulas/').on('value', function (snapshot){
10    var classrooms = snapshot.val();
```

Código 2.109. Obtención de datos de aulas de Firebase

Una vez inicializados los datos, se procede con la consulta de horarios de aulas; para ello primero se recupera el valor del aula y la fecha de la consulta que se almacenarán en la variable `queryData` dentro del estado de la clase `ClassroomList`. Para recuperar el valor del aula, al elegirla con el `Picker`, se llama a la función `onClassValueChange` la cual simplemente se encarga de actualizar el estado del objeto en la variable correspondiente, en este caso, `selectedClass` dentro de `queryData`, como se muestra en el Código 2.111.

```
141 if(requestList[key].Aprobacion == 0)
142 {
143   rowData.push('Pendiente');
144 }
145 else if(requestList[key].Aprobacion == 1)
146 {
147   rowData.push('Aprobada');
148 }
149 else
150 {
151   if(requestList[key].Error == 0)
152   {
153     rowData.push('Rechazada\nMotivo: Acercarse a Coordinacion');
154   }
155   else if(requestList[key].Error == 1)
156   {
157     rowData.push('Rechazada\nMotivo: Aula no disponible');
158   }
159   else if(requestList[key].Error == 2)
160   {
161     rowData.push('Rechazada\nMotivo: Horario no disponible');
162   }
163   else if(requestList[key].Error == 3)
164   {
165     rowData.push('Rechazada\nMotivo: La fecha es un feriado');
166   }
167 }
```

Código 2.110. Definición de estado de solicitud a mostrar

```
31  onClassValueChange = (itemValue, itemIndex) => {
32    this.setState(prevState => ({
33      queryData: {
34        ...prevState.queryData,
35        selectedClass: itemValue
36      }
37    }))
38  }
```

Código 2.111. Función onClassValueChange

Para obtener el valor de la fecha se recurre al componente `DateTimePicker` el cual es definido con modo fecha por defecto; su valor será el que tenga `selectedDate` dentro de `queryData` y llamará al método `setDate` el cual, al igual que el método anterior, solo actualiza la variable de estado correspondiente.

Una vez obtenida la información se la envía en la acción `getSchedule`, en donde inicialmente se genera las fechas de inicio y fin de la fecha seleccionada, mediante las cuales se validará si se muestra o no algún horario de reserva que todavía se encuentre vigente en esas fechas, además se genera un arreglo `schedule` que servirá como esqueleto para guardar los horarios.

```
12  if(date.getDay() !== 0){
13    var diff = ((date.getDay() - 1)%7)
14    startOfWeek.setDate(startOfWeek.getDate()-diff)
15
16    diff = (6 + (1 - date.getDay()))%7)
17    endOfWeek.setDate(endOfWeek.getDate()+diff)
18  }
19  else
20  {
21    startOfWeek.setDate(startOfWeek.getDate()-6)
22  }
```

```
24  for (let i = 0; i < 13; i += 1) {
25    const rowData = [];
26    for (let j = 0; j < 6; j += 1) {
27        rowData.push('');
28    }
29    schedule.push(rowData);
30  }
```

Código 2.112. Fragmento función getSchedule

Se recorren los grupos del aula y sus horarios, escribiendo en las casillas correspondientes la leyenda “Ocupado” y una vez terminado este proceso, se envía a través del *dispatcher* a la *store*. El modal `Results` recupera los datos de la *store*, y mediante éstos graficará la tabla con el horario obtenido, junto con una etiqueta indicando el aula, y el periodo de la consulta.

Para la consulta de registros de asistencia, primero se tiene un `CheckBox`, con el cual se habilitarían los `Picker` correspondientes al filtro que limita la muestra de registros a una materia y grupo. Si están habilitados, al seleccionar la materia, se llama a la función `getCorrespondingGroups` la cual busca los grupos correspondientes a la materia seleccionada, para cargarlos en el `Picker` de grupos, y poder seleccionar uno de ellos.

Adicionalmente se tienen dos controles más para elegir el periodo de fechas en los cuales se desea hacer la consulta. Una vez obtenidos todos los parámetros, se los envía en la acción `getRecords`, en la cual se obtienen los registros del usuario a través de la función `database.ref` de Firebase.

Una vez obtenidos, se comprueba que la fecha del registro esté entre las fechas indicadas en la consulta, además si el filtro se activó, se comprobará si el registro coincide o no con la materia y grupo seleccionados (Código 2.113).

Se crea un arreglo en el cual se guardarán los registros, y a medida que se van obteniendo, se guardan en el arreglo `rowData`, el cual se escribirá posteriormente en el arreglo `tableData`, mismo que será enviado a través del *dispatcher*, y que será la tabla a mostrarse en el modal `Results`.

```
26  if (+fechaReg >= +fechaRegAux && +fechaReg <= +fechaFinAux)
27  {
28    var filtrar = true;
29
30    if(queryData.filterActivated)
31    {
32      if(queryData.selectedGroup !== records[item].NombreGr ||
33        queryData.selectedSubject !== records[item].NombreMat)
34      {
35        filtrar = false;
36      }
37    }
```

Código 2.113. Validación de consulta

2.3.14.3 Entregable

El entregable de este *sprint* consiste en la funcionalidad de las pantallas Inicio, Lista de Aulas y Registros de Asistencia.

2.3.15 SPRINT 15

En este *sprint* se implementará la solicitud de reservas por parte de la Aplicación Móvil, y el manejo de las mismas en la Aplicación de Administración en el submódulo Solicitud de Reservas dentro del módulo Aulas.

2.3.15.1 Diseño

El proceso para enviar una solicitud de reserva desde la aplicación móvil y atenderla desde la aplicación de administración, se presenta en el diagrama de actividades en la Figura 2.66.

2.3.15.2 Implementación

Para poder realizar solicitudes de reserva, se recuperan 6 parámetros para la consulta: La fecha de reserva, la materia, el grupo, el aula, la hora de inicio y fin de la reserva. El grupo se genera de manera similar a la mostrada en el *sprint* anterior.

Se validará antes de crear la solicitud, considerando que la fecha elegida sea mayor o igual a la fecha en que se realiza la operación, y si es igual se valida que las horas elegidas

sean mayores a la hora actual. En ambos casos se valida que la hora de inicio sea menor a la hora fin del periodo de clase, además que sean como máximo 3 horas.

Una vez validada la solicitud, se envían los datos a la acción `createRequest`, la cual llama a la función `validateRequest`, misma a la que se le envía la solicitud y el aula en cuestión para aprobar o no su adición a la base de datos.

Una vez que se pase ese filtro, se crea una referencia en la cual se guardará la solicitud a enviar en base a los datos seleccionados, como se muestra en el Código 2.115.

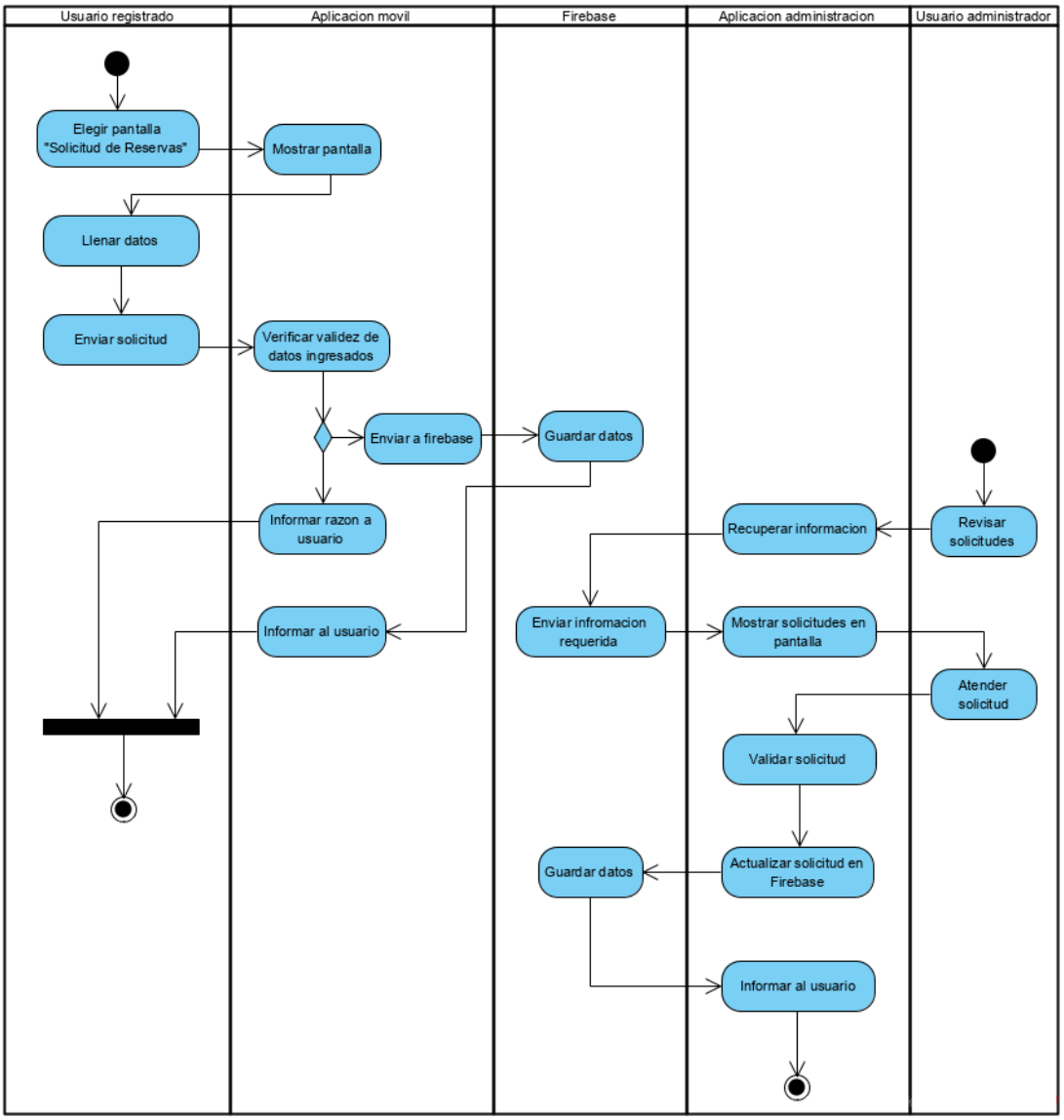


Figura 2.66. Diagrama de actividades del manejo de Solicitudes de Reserva

Finalmente, con la ayuda de la función `set`, se envían todos los datos de la solicitud, además de los respectivos *dispatcher* en caso de que la operación sea, o no exitosa, de lo cual se informará al usuario a través de un `Alert` (Código 2.116).

En el lado del servidor, desde la Aplicación de Administración se llama a la función `ObtenerSolicitudes`, la cual devolverá todas las solicitudes pendientes, es decir, con el valor `Aprobacion` en cero. Una vez que éstas se muestren en pantalla, se podrá elegir entre atender la solicitud, o denegarla. Es necesario señalar que no se pueden denegar directamente ya que se debe atender en primer lugar.

```
75  if(ranura.Dia === daysOfWeek[request.FechaReserva.getDay()-1])
76  {
77
78    if(horaFin.getUTCHours() >= request.HoraFin
79    && horaInicio.getUTCHours() <= request.HoraIni)
80    {
81        horaValida= false;
82    }
83    if(horaFin.getUTCHours() > request.HoraFin
84    && horaInicio.getUTCHours() <= request.HoraFin)
85    {
86        horaValida= false;
87    }
88    if(horaInicio.getUTCHours() <= request.HoraIni
89    && horaFin.getUTCHours() > request.HoraIni)
90    {
91        horaValida= false;
92    }
93    if(horaFin.getUTCHours() < request.HoraFin
94    && horaInicio.getUTCHours() > request.HoraIni)
95    {
96        horaValida= false;
97    }
98  }
```

Código 2.114. Algoritmo para validación de solicitud

```
37  var cedula = firebase.auth().currentUser.uid;
38
39  var referencia = database.ref('Solicitudes')
40    .child('@'+ cedula + '-' +
41    request.FechaReserva.getFullYear() + '-' + dateMonth + '-' + dateDay + '-' +
42    request.NombreMat.replace('/','*'));
```

Código 2.115. Creación de referencia donde se guarda la solicitud a enviar

```
43  referencia.set({
44    Aprobacion: 0,
45    Cedula: cedula,
46    Error: 0,
47    NombreMat: request.NombreMat,
48    NombreGr: request.NombreGr,
49    NombreAu: request.NombreAu.replace('*','/'),
50    FechaReserva: request.FechaReserva,
51    HoraIni: request.HoraIni,
52    HoraFin: request.HoraFin
53  })
54  .then(() =>{
55    dispatch({type: 'CREATE_REQUEST', request})
56  })
57  .catch((error) => {
58    dispatch({type: 'CREATE_REQUEST_ERROR', error})
59  });
```

Código 2.116. Envío de solicitud

Para atender la solicitud, se crea un nuevo constructor en el submódulo de Reservas, el cual permite inicializarlo en base a un objeto `solicitud`. Conforme se empiezan a colocar los datos en los respectivos controles, se comprueba que el dato añadido está disponible y sea válido, en base a los algoritmos ya establecidos dentro del submódulo; en caso de no serlo, se informa al usuario administrador y se regresa al submódulo Solicitudes de Reserva.

Una vez que se ha regresado al submódulo Solicitudes, se puede denegar la solicitud, y actualizar la información de ésta, en conjunto a la razón por la que ésta no es válida. Para esto se llama a la función `AtenderSolicitud`, la cual se muestra en el Código 2.117.

```
3657  public bool AtenderSolicitud(Solicitud solicitud)
3658  {
3659      //Obtiene un objeto del tipo fbclient para poder ingresar los datos
3660      IFirebaseClient client = GenerateClientFB();
3661      //sobreescribe la solicitud que ya fue atendida
3662      SetResponse response =
3663          client.Set("Solicitudes/" + "@" + solicitud.Cedula + "-"
3664          + solicitud.FechaReserva.ToString("yyyy-MM-dd") + "-"
3665          + solicitud.NombreMat.Replace("/","*"), solicitud);
3666      //transforma el resultado a un objeto de tipo solicitud
3667      Solicitud result = response.ResultAs<Solicitud>();
3668      //si el objeto enviado y el objeto devuelto coinciden en el valor del atributo aprobacion
3669      if (solicitud.Aprobacion == result.Aprobacion)
3670      {
3671          //se retorna un booleano en true
3672          return true;
3673      }
3674      //caso contrario se retorna un booleano en false
3675      return false;
3676  }
```

Código 2.117. Función AtenderSolicitud

Si todos los datos son válidos, se permitirá al administrador ejecutar la reserva y, por ende, aprobar la solicitud, con lo cual se regresa al submódulo Solicitudes, cuya interfaz gráfica corresponde a la Figura 2.46 (a), mostrándose solo las que queden pendientes.

Una vez atendida la solicitud el usuario podrá visualizar el estado de la misma, a través de la aplicación móvil. Dentro de la pantalla principal se usa el componente `NavigationEvents`, del que se utiliza la función `OnWillFocus` para cargar las solicitudes del usuario una vez acceda a la pantalla correspondiente.

```
21  <NavigationEvents
22    onWillFocus={payload => {
23      this.props.getRequest();
24    }}
25  />
```

Código 2.118. Uso del componente NavigationEvents

2.3.15.3 Entregable

El entregable de este *sprint* es la generación y manejo de solicitudes de reserva.

2.3.16 SPRINT 16

En este *sprint* se documentará la implementación física del sistema, es decir las conexiones e instalaciones efectuadas tanto en el ambiente real como en el ambiente emulado (maqueta a construirse).

2.3.16.1 Diseño

La maqueta se diseñó con el propósito de emular un aula de clases, motivo por el cual basados en el escenario real, la maqueta consta de: Una pared, la puerta y una cavidad interna donde se encontrarán las conexiones soldadas de: El dispositivo biométrico, la cerradura electromagnética y el pulsador. Mediante este último se desbloqueará la cerradura desde el interior del aula.

Los materiales necesarios para la instalación se muestran en la Figura 2.67.

Figura 2.67. Materiales: a) Dispositivo Biométrico, b) Cerradura electromagnética de 300 libras, c) Soporte tipo Z, d) Soporte tipo L, e) Pulsador, f) Fuente de 12 VDC

Cabe mencionar que se adquirió dos unidades de cada uno de estos materiales para la instalación en ambos escenarios. Adicionalmente se requiere: Cable 24 AWG para las conexiones eléctricas, y cable FTP categoría 5E para las conexiones de red, estaño, y canaletas con sus respectivas uniones.

En la Figura 2.68, se presenta una vista isométrica realizada en AutoCAD para la maqueta de madera.

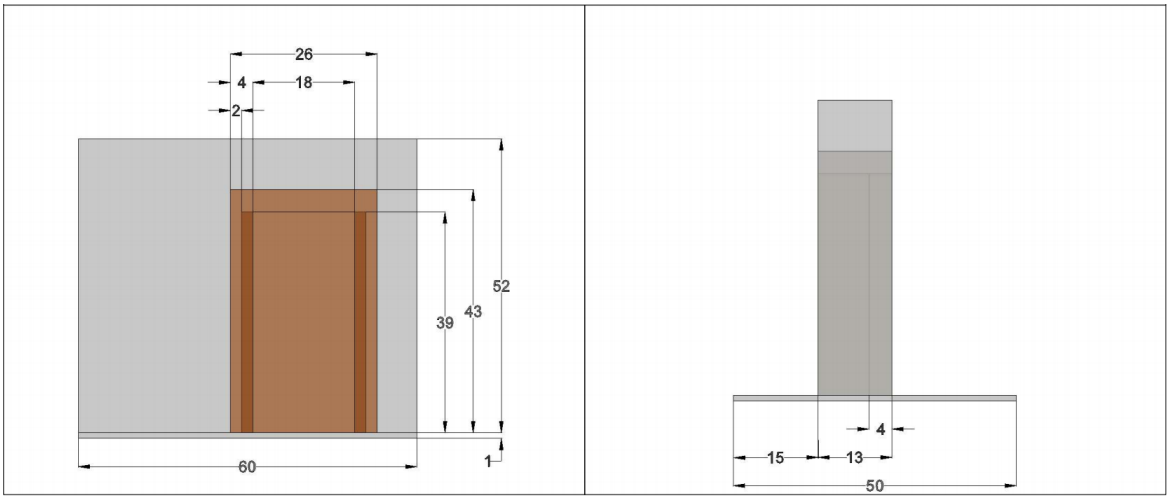


Figura 2.68. Plano acotado de maqueta

Adicionalmente se presenta el diagrama de conexiones entre los componentes de Hardware del sistema, ya mencionados anteriormente (Figura 2.69).

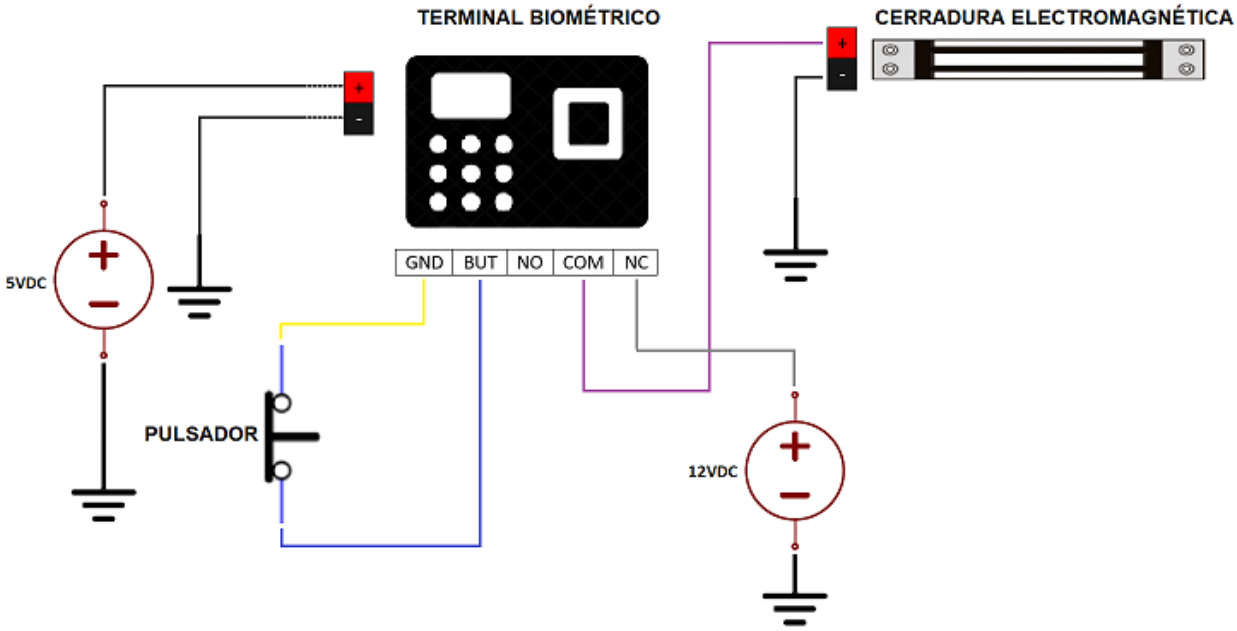


Figura 2.69. Diagrama de conexiones

2.3.16.2 Implementación

Para la implementación de los componentes en el aula Q/E701, con el debido permiso de las autoridades competentes, se procedió a fijar mediante soportes al dispositivo biométrico, la cerradura electromagnética y el pulsador. Posteriormente, se soldaron las conexiones entre el lector biométrico, el pulsador y la cerradora electromagnética, y se realizó la instalación de canaletas conjuntamente con el tendido de cables para respetar la fachada de la facultad.

Finalmente se realizó la conexión de red cableada entre el dispositivo biométrico y el punto de red más cercano. Este proceso se repitió para el ambiente emulado (maqueta).

En la Figura 2.70 se pueden apreciar los componentes instalados en ambos escenarios.

(a) (b)

(c) (d)

Figura 2.70. (a) Vista interior escenario real, (b) Vista exterior escenario real, c) Vista interior escenario emulado, (d) Vista exterior escenario emulado

2.3.16.3 Entregable

El entregable de este *sprint* consiste en el hardware del sistema interconectado tanto en el escenario real como en el simulado, respetando la fachada y estética en ambos casos.

3. RESULTADOS Y DISCUSIÓN

En el presente capítulo se exponen los resultados de las distintas pruebas aplicadas al sistema. Se realizan pruebas de conectividad entre los distintos componentes del mismo. Se hacen pruebas de funcionamiento tanto del servicio como de las bases de datos; de igual forma se puso a prueba la lógica del sistema, que abarca el control de acceso y de asistencia, entre otros. Finalmente se presenta el análisis de resultados, considerando algunas posibles situaciones de error.

3.1 PRUEBAS DE CONECTIVIDAD

Con el propósito de comprobar la conectividad entre los elementos que conforman el sistema, se utilizó el comando `ping` desde y hacia las distintas direcciones IP, mismas que se presentan en la Tabla 3.1.

Tabla 3.1. Direcciones IP asignadas a elementos del sistema

Elemento	Dirección IP
Servidor	172.31.44.209/25
Terminal biométrico asignado al aula	172.31.38.209/22
Terminal biométrico asignado a la maqueta	172.31.44.210/25
PC con Aplicación Administración	DHCP EPN

Al ejecutar el comando mencionado entre los elementos, se obtuvieron respuestas exitosas en todos los casos como se muestra en las Figura 3.1Figura 3.2Figura 3.3Figura 3.4.

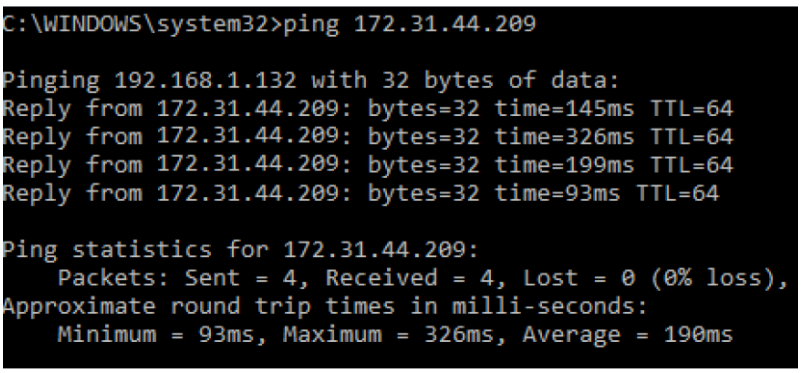


Figura 3.1. Prueba de conectividad Aplicación de administración-Servidor

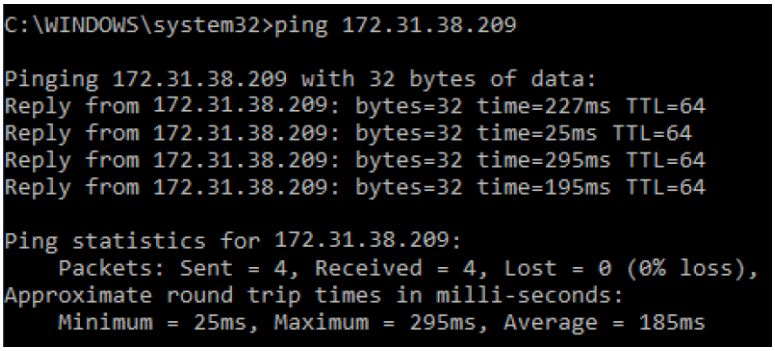


Figura 3.2. Prueba de conectividad Servidor-Terminal asignado al aula

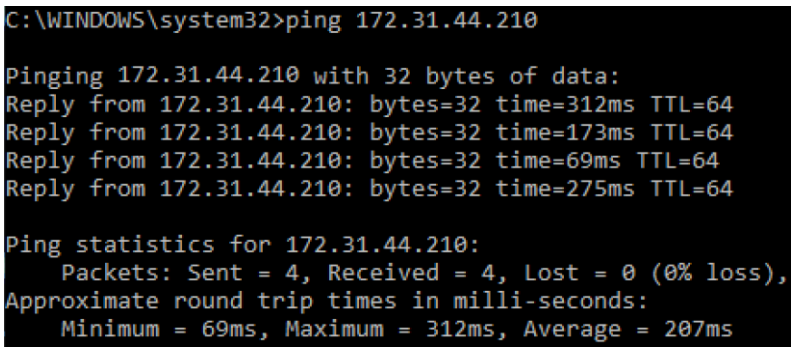


Figura 3.3. Prueba de conectividad Servidor-Terminal asignado a la maqueta

Para la verificación de conectividad entre el servidor y Firebase, dentro de la función que inicia esta comunicación, se invocó al método para imprimir por consola el nombre por defecto (“[DEFAULT]”) tomado por el objeto generado cuando dicha comunicación ha sido exitosa.

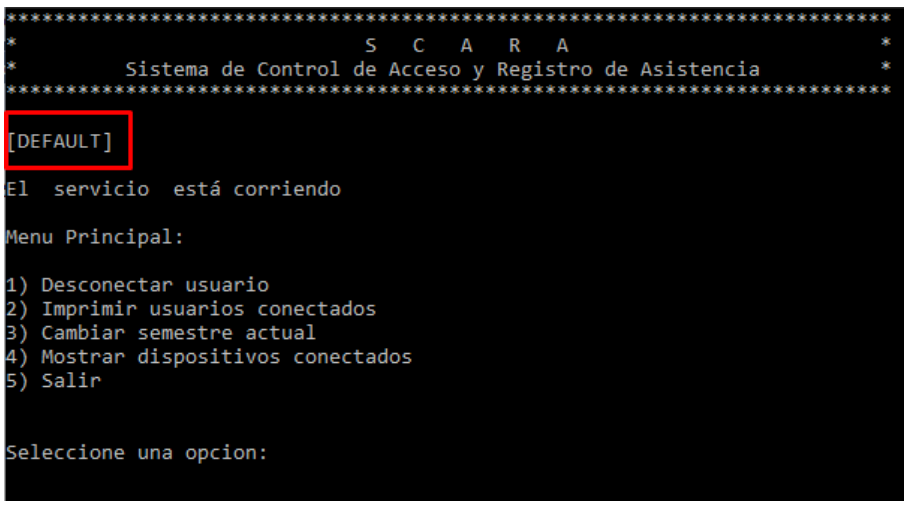


Figura 3.4. Prueba de conectividad Servidor-Firebase

Con base a las distintas pruebas realizadas se evidencia la correcta comunicación entre los distintos elementos del sistema y además del acceso exitoso desde el servidor a Firebase.

3.2 PRUEBAS DE FUNCIONAMIENTO

3.2.1 BASES DE DATOS

Para comprobar el funcionamiento correcto de las bases de datos, se realizan acciones de recuperación de información en cada una de ellas.

3.2.1.1 SQL Server

En la

	idPro	cedula	nombresPro	apellidosPro	contrasena	email	imagenUrl	huella	IsAdmin	Habilitado	IsProf
2	2	[redacted]	FELIPE ANTONIO	NAVAS NAVAS	NULL	NULL	NULL	NULL	0	1	1
3	3	[redacted]	JACKELINE	ABAD TORRES	NULL	NULL	NULL	NULL	0	1	1
4	4	[redacted]	ANDRES FERNANDO	CELA ROSERO	NULL	NULL	NULL	NULL	0	1	1

Figura 3.5 se presentan los resultados de la ejecución del comando `SELECT * FROM Usuarios` a la tabla Usuarios. Aquí se pueden observar todos los datos que deberá tener un usuario; los que se encuentran vacíos (NULL), serán modificados mediante la aplicación de administración en el submódulo Editar Usuarios durante el proceso recopilación de huellas del personal.

	idPro	cedula	nombresPro	apellidosPro	contrasena	email	imagenUrl	huella	IsAdmin	Habilitado	IsProf
2	2	[redacted]	FELIPE ANTONIO	NAVAS NAVAS	NULL	NULL	NULL	NULL	0	1	1
3	3	[redacted]	JACKELINE	ABAD TORRES	NULL	NULL	NULL	NULL	0	1	1
4	4	[redacted]	ANDRES FERNANDO	CELA ROSERO	NULL	NULL	NULL	NULL	0	1	1

Figura 3.5. Fragmento de resultados de la consulta a la tabla Usuarios

En la Figura 3.6 se presentan los resultados de la ejecución del comando `SELECT * FROM Aulas` a la tabla Aulas que contiene la información básica de las mismas.

	idAula	nombreAu	capacidad
1	1	Q/E503	49
2	2	Q/E304	49

Figura 3.6. Fragmento de resultados de la consulta a la tabla Aulas

En la Figura 3.7 se presentan los resultados del comando `INNERJOIN` el cual concatena información de varias tablas, en este caso las tablas Usuarios, Materias, Grupos, Aulas y RTs. Mediante esta ejecución se pueden observar los horarios de los docentes.

En la Figura 3.8 se presentan los resultados de la ejecución del comando `SELECT` a la tabla Dispositivos que contiene las direcciones IP asignadas y la información extraída de cada dispositivo.

	nombresPro	apellidosPro	nombreMat	nombreGr	nombreAu	dia	horaIni	horaFin
1	FELIPE ANTONIO	NAVAS NAVAS	ALGEBRA LINEAL	GR10	Q/E503	Lunes	14:00:00.0000000	15:00:00.0000000
2	FELIPE ANTONIO	NAVAS NAVAS	ALGEBRA LINEAL	GR10	Q/E503	Lunes	15:00:00.0000000	16:00:00.0000000
3	FELIPE ANTONIO	NAVAS NAVAS	ALGEBRA LINEAL	GR10	Q/E503	Miercoles	14:00:00.0000000	15:00:00.0000000
4	FELIPE ANTONIO	NAVAS NAVAS	ALGEBRA LINEAL	GR10	Q/E503	Miercoles	15:00:00.0000000	16:00:00.0000000
5	ANGEL PATRICIO	VILLOTA CADENA	CALCULO EN UNA VARIABLE	GR1	Q/E503	Martes	07:00:00.0000000	08:00:00.0000000
6	ANGEL PATRICIO	VILLOTA CADENA	CALCULO EN UNA VARIABLE	GR1	Q/E503	Martes	08:00:00.0000000	09:00:00.0000000
7	ANGEL PATRICIO	VILLOTA CADENA	CALCULO EN UNA VARIABLE	GR1	Q/E503	Jueves	07:00:00.0000000	08:00:00.0000000
8	ANGEL PATRICIO	VILLOTA CADENA	CALCULO EN UNA VARIABLE	GR1	Q/E503	Jueves	08:00:00.0000000	09:00:00.0000000
9	MARIA PAULINA	ROMERO OBANDO	CALCULO EN UNA VARIABLE	GR2	Q/E503	Martes	14:00:00.0000000	15:00:00.0000000
10	MARIA PAULINA	ROMERO OBANDO	CALCULO EN UNA VARIABLE	GR2	Q/E503	Martes	15:00:00.0000000	16:00:00.0000000
11	MARIA PAULINA	ROMERO OBANDO	CALCULO EN UNA VARIABLE	GR2	Q/E503	Jueves	14:00:00.0000000	15:00:00.0000000
12	MARIA PAULINA	ROMERO OBANDO	CALCULO EN UNA VARIABLE	GR2	Q/E503	Jueves	15:00:00.0000000	16:00:00.0000000

Figura 3.7. Fragmento de resultados de la consulta `INNERJOIN` para visualizar Horarios

idAula	ip	mac	codigo	fabricante	tipo	serial
3	172.31.44.210	00:17:61:AE:37:A6	K20	ZKTeco Inc.	JZ4725_TFT	A8LN183960236
12	172.31.38.209	00:17:61:10:98:11	K50	ZKTeco Inc.	ZLM60_TFT	A2QO193260106

Figura 3.8. Resultados de la consulta a la tabla Dispositivos

En la Figura 3.9 se presentan los resultados de la ejecución del comando `INNERJOIN` el cual concatena las tablas Usuarios, Registros, Materias, Grupos, y Aulas, para poder observar la información contenida en los registros de los docentes.

	apellidosPro	nombresPro	nombreMat	nombreGr	nombreAu	fecha
1	AVALOS CASCANTE	FAUSTO EDUARDO	TEORIA ELECTROMAGNETICA	GR2	Q/E305	2020-06-01 07:04:14.000
2	AVALOS CASCANTE	FAUSTO EDUARDO	TEORIA ELECTROMAGNETICA	GR2	Q/E305	2020-06-01 08:55:43.000
3	AVALOS CASCANTE	FAUSTO EDUARDO	ELECTRICIDAD Y MAGNETISMO	GR2	Q/E404	2020-06-02 07:03:25.000
4	AVALOS CASCANTE	FAUSTO EDUARDO	ELECTRICIDAD Y MAGNETISMO	GR2	Q/E404	2020-06-02 08:55:27.000

Figura 3.9. Fragmento de resultados de consulta `INNERJOIN` para visualizar Registros

3.2.1.2 Firebase

En la Figura 3.10 se hace uso del comando `curl` el cual envía a Firebase peticiones HTTP y obtiene la respuesta correspondiente. Este comando permite verificar el funcionamiento correcto de Firebase al responder exitosamente a las peticiones realizadas.

En la Figura 3.11, se presenta una muestra de las entidades que conforman la base de datos creada en Firebase.

Mediante las pruebas realizadas se puede demostrar para ambos casos el correcto funcionamiento de las bases de datos, ya que se evidencia la respuesta exitosa a comandos enviados a cada una de ellas. Particularmente para la base de datos de SQL Server se puede también verificar la correcta estructura del diagrama relacional presentado.

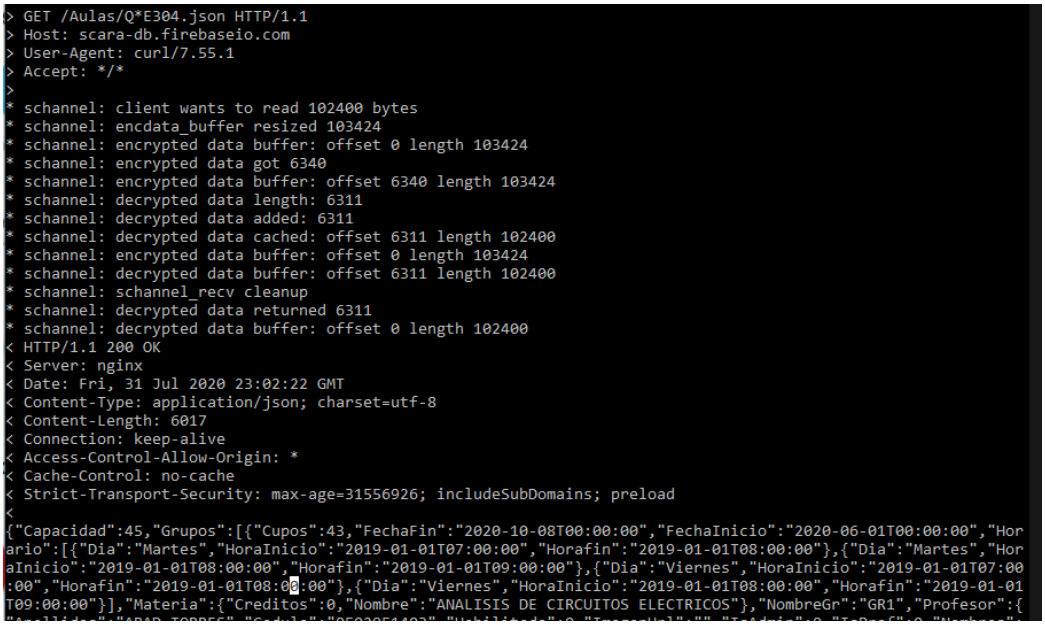


Figura 3.10. Petición/Respuesta a scara-db.firebaseio.com

Figura 3.11. Muestra de Entidades Firebase

3.2.2 APLICACIÓN DE ADMINISTRACIÓN

En esta sección se probará la operación de todos los submódulos que componen la aplicación de Administración, esto adicionalmente permitirá que en simultáneo se verifique la correcta funcionalidad de cada uno de los elementos que integran el sistema.

3.2.2.1 Módulo Actualización BD

Para la comprobación de este submódulo, se utiliza un archivo Excel que contiene la distribución horaria del semestre más reciente, mismo que fue proporcionado por el

subdecanato de la facultad con el fin de que la información utilizada en el presente proyecto sea actualizada y veraz.

Se procedió a extraer los datos del documento mediante el botón obtener datos e ingresar los campos obligatorios como son el periodo del semestre y sus fechas de inicio y fin, esto con el fin de exportar la información a las bases de datos (Figura 3.12).

Si la operación se realiza con éxito, se notifica al usuario administrador la cantidad de información exportada a la base de datos, caso contrario se indica al usuario el error que se produjo (Figura 3.13).

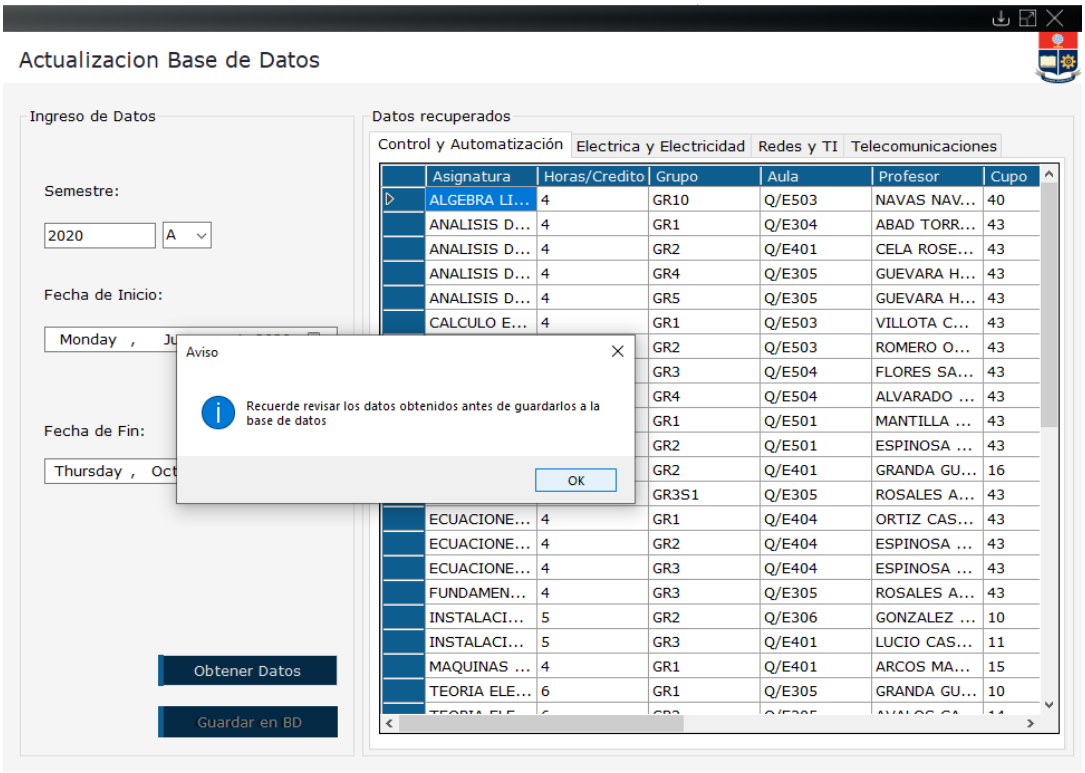


Figura 3.12. Extracción de datos de documento Excel a Aplicación Administración

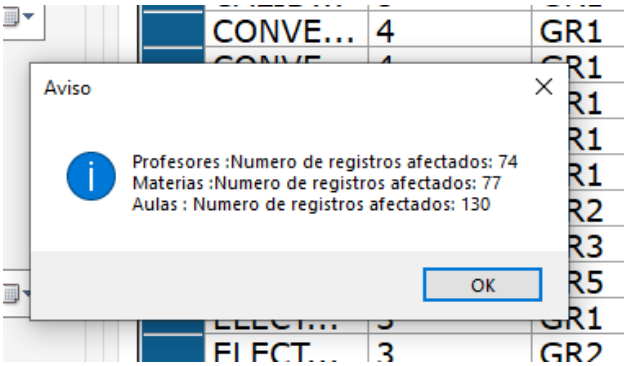


Figura 3.13. Mensaje de confirmación de exportación de datos

3.2.2.2 Módulo Usuarios

3.2.2.2.1 Submódulo Agregar Usuario

Para la comprobación de este submódulo, se agregó un usuario administrador llenando todos los campos requeridos por la interfaz. En la obtención y comprobación de la huella dactilar, se utilizó el escáner ZKTeco 9500. Posteriormente se verificó la inserción de la información del usuario en la base de datos.

En este submódulo se realiza la validación de cada uno de los campos, es decir: La cédula debe ser correcta, el usuario puede poseer uno o más nombres, el *email* institucional se coloca por defecto pero éste puede ser editado, la contraseña debe tener una longitud de 8 caracteres, para que la huella sea válida ésta debe ser comprobada y, se puede o no colocar foto del docente.

En caso de producirse una falla al agregar el usuario, la aplicación informará de ésta. Por ejemplo, si el usuario ya se encuentra registrado en la base de datos, se muestra este error mediante un mensaje en pantalla.

Figura 3.14. Inserción de datos de usuario a agregarse

	idPro	cedula	nombresPro	apellidosPro	contrasena	email
1	76	[redacted]	SERGIO NICOLAS	GOMEZ URIBE	12345678	sergio.gomez@epn.edu.ec

imagenUrl	huella	IsAdmin	Habilitado
https://firebasestorage.googleapis.com/v0/b/prue...	0x4DBD53523232000004FEFF0505050709CED0000028FF76...	1	1

Figura 3.15. Verificación de inserción de información en la base de datos

3.2.2.2.2 Submódulo Editar Usuarios

Para la comprobación de este submódulo, se edita a un usuario docente; cabe mencionar que cuando se edita a un docente por primera vez la única información que se recupera de la base de datos es la que se muestra en la Figura 3.16. Mediante este submódulo se podrá insertar los datos faltantes de los docentes pre-registrados a través del módulo Actualización BD, así como también se podrá editar los datos de un usuario que ya ha completado anteriormente su registro.

De igual forma que en el submódulo anterior, se verifica que todos los campos estén correctos y llenos. En caso de ocurrir un error en la modificación del usuario, la aplicación informará al respecto.

Figura 3.16. Obtención de información desde la base de datos del usuario pre-registrado

Figura 3.17. Inserción de datos

idPro	cedula	nombresPro	apellidosPro	contrasena	email
62	[redacted]	PABLO WILIAN	HIDALGO LASCANO	18015374	pablo.hidalgo@epn.edu.ec

imagenUrl	huella	IsAdmin	Habilitado	IsProf
https://firebasestorage.googleapis.com/v0/b/prue...	0x4AA353523232000003E0E20505050709CED000002FE176...	0	1	1

Figura 3.18. Verificación de información insertada en la base de datos

3.2.2.3 Módulo Aulas

3.2.2.3.1 Submódulo Mostrar Aulas

El propósito de este submódulo es únicamente informativo, por esta razón la comprobación de este submódulo consiste en la correcta recuperación de la información de la base de datos, y la representación de la misma en la interfaz gráfica.

Para esto, al seleccionar un aula del control respectivo, se muestra el horario de la misma y su capacidad; adicionalmente al dar clic a cualquier celda que contenga un horario se presentará la información del grupo correspondiente.

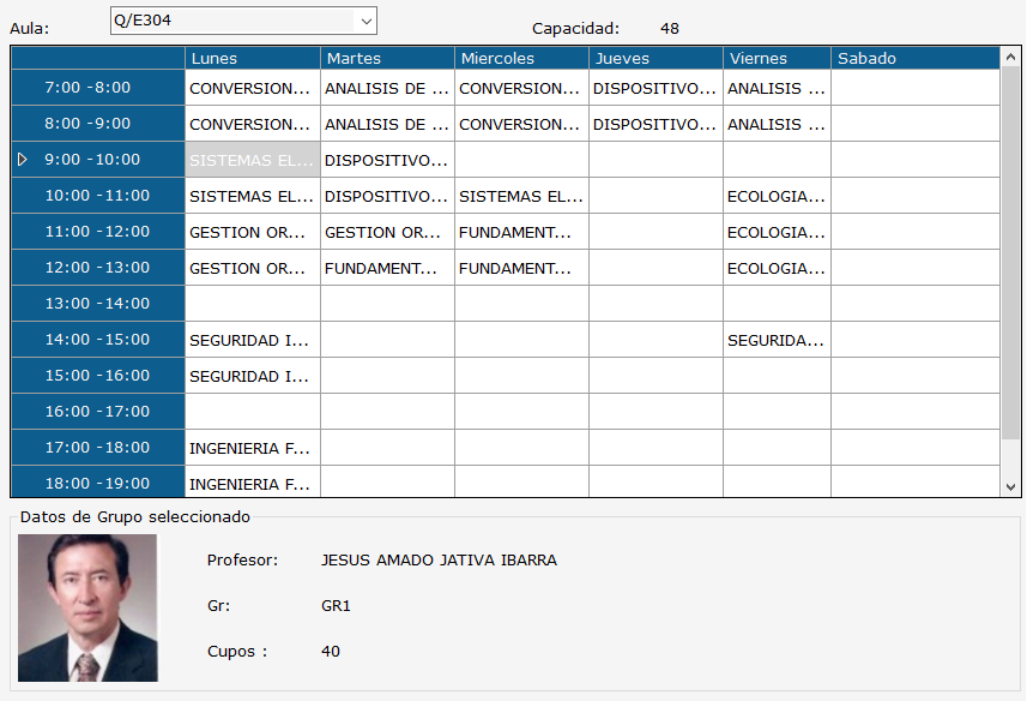


Figura 3.19. Presentación de horario e información de grupo en submódulo Mostrar Aulas

3.2.2.3.2 Submódulo Cambios Horarios

Para la comprobación del este submódulo, se puso a prueba los diferentes modos de operación del mismo como se presentará a continuación; para todas las pruebas

realizadas se selecciona la materia “SISTEMAS ELÉCTRICOS DE POTENCIA”, y el grupo uno “GR1”.

- **Caso 1:** Agregar o Eliminar un Horario sobre un aula en la que ya se encuentra el grupo (opción Agregar/Eliminar Horario seleccionada).

Una vez seleccionados la materia y el grupo, se observa en la primera tabla mostrada en pantalla el horario inicial del mismo, que para este caso es el día lunes de 9:00 a 11:00 y el miércoles de 10:00 a 11:00 (Figura 3.20).

Para proceder a la modificación, en primer lugar se deben eliminar horas de clase del horario inicial, con el fin de poder agregarlas posteriormente en otro horario de la misma aula. Se procedió a eliminar las horas de clase del día lunes, y posteriormente se eligió el nuevo horario el día martes. Cabe mencionar que una vez seleccionado el día deseado, en el control para escoger el periodo de tiempo, la aplicación únicamente presenta las horas en las cuales el aula se encuentra disponible para ese día (Figura 3.20); por esto se escogió una de las opciones mostradas en pantalla (14:00-16:00).

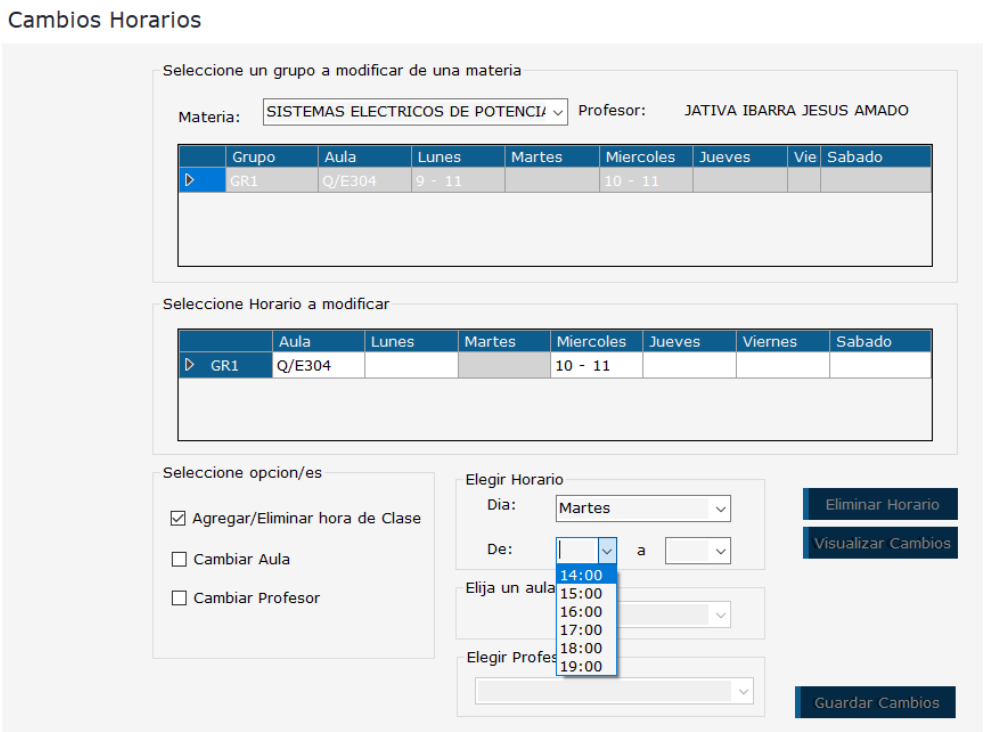


Figura 3.20. Selección de Grupo sujeto a modificación horaria y agregación de nuevo horario

Para verificar que los cambios realizados se efectuaron exitosamente en el sistema, se muestra en pantalla el nuevo horario recuperado de la base de datos.

Figura 3.21. Visualización de cambios efectuados exitosamente

- **Caso 2:** Cambiar un horario completo de un aula a otra (opción Cambiar Aula seleccionada).

Posterior a la selección inicial, se observa en la primera tabla de la Figura 3.22 el horario y el aula del grupo a modificar.

Se procede a seleccionar la nueva aula de la lista presentada en pantalla, misma que corresponde a las aulas que se encuentran disponibles en el horario del grupo y por tanto podrían alojarlo. Para la presente prueba se seleccionó el aula 402.

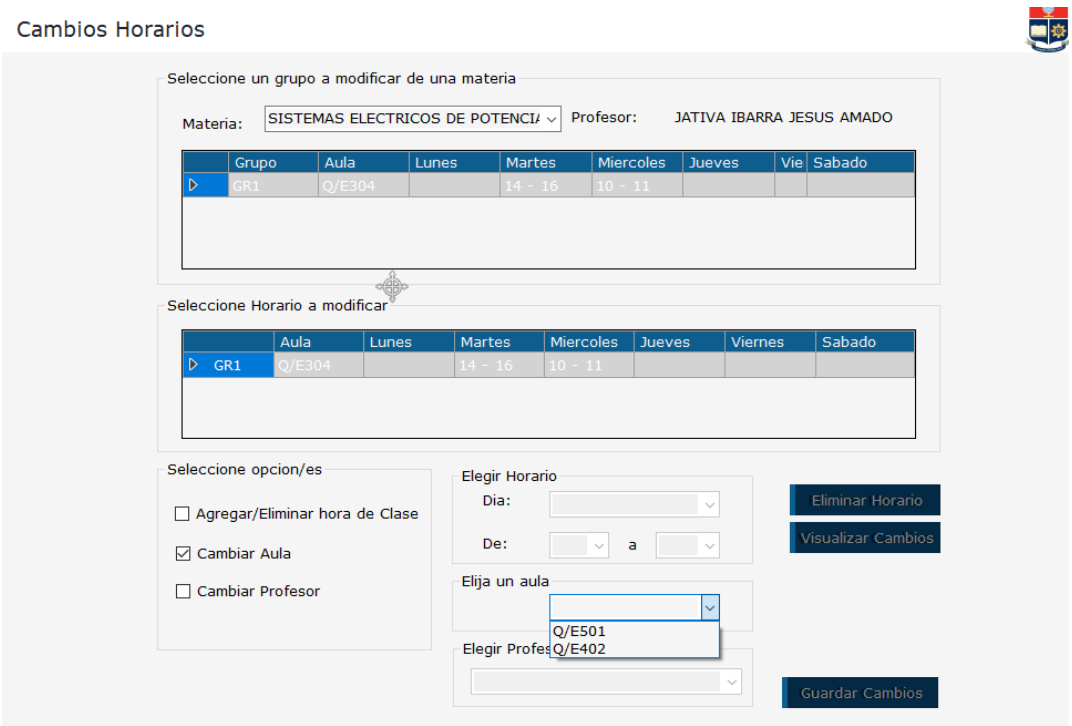


Figura 3.22. Selección de grupo sujeto a modificación de aula y selección de nueva aula

Para verificar que los cambios realizados se efectuaron exitosamente en el sistema, se muestra en pantalla la información recuperada de la base de datos.

Figura 3.23. Visualización de cambio de aula efectuado exitosamente

- **Caso 3:** Cambiar profesor (opción seleccionada Cambiar Profesor).

Una vez realizada la selección inicial, se observa en la parte superior derecha de la pantalla el docente actual del grupo seleccionado, en este caso el Dr. “JATIVA IBARRA JESUS AMADO” (Figura 3.24).

Se procede a seleccionar el nuevo docente “NELSON VICTORIANO GRANDA GUTIERREZ” de la lista mostrada en pantalla. En caso que el horario del docente seleccionado se cruce con el horario del grupo, el sistema informará al usuario administrador que el cambio no podrá ser efectuado.

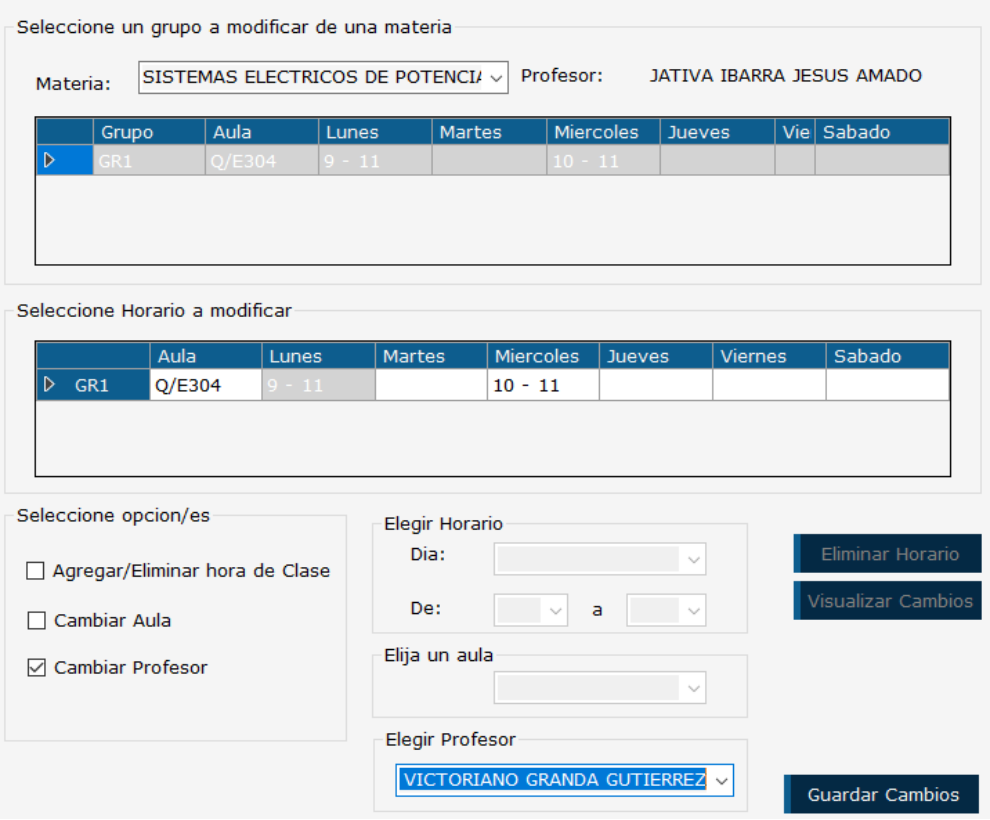


Figura 3.24. Selección de grupo sujeto a modificación de docente y selección del nuevo docente

Para verificar que los cambios realizados se efectuaron exitosamente en el sistema, se muestra en pantalla la información recuperada de la base de datos.

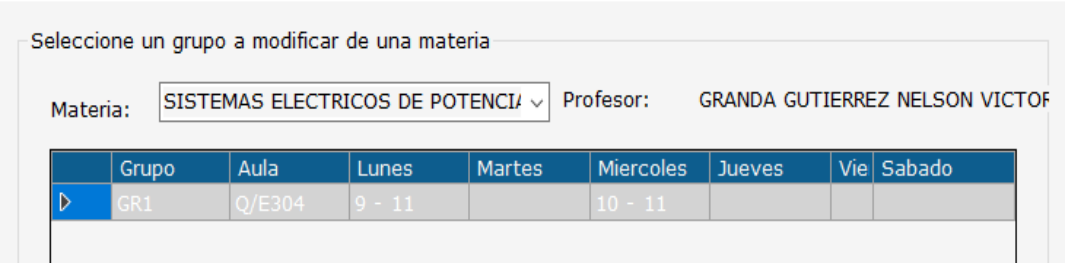


Figura 3.25. Visualización de cambio de docente efectuado exitosamente

- **Caso 4:** Ingresar un horario extra en otra Aula (opción Agregar/Eliminar Horario y opción Cambiar Aula).

Una vez seleccionados la materia y el grupo, se procederá a realizar las modificaciones. Para esto en primer lugar se eliminará las horas de clase del horario inicial, con el fin de poder agregarlas posteriormente en otro horario de otra aula.

Se procede a eliminar las horas de clase del día lunes, y posteriormente se elige el nuevo horario en la nueva aula, en este caso el día lunes de 9:00 a 11:00. Cabe mencionar que una vez seleccionado el día, en el control para escoger el periodo de tiempo, la aplicación muestra el rango de horas de 7:00 a 19:00 con las excepciones mencionadas anteriormente de la hora de almuerzo, y los días jueves.

Posteriormente, al elegir la nueva aula, se muestra en pantalla las aulas que presentan disponibilidad en el horario seleccionado; para la presente prueba se escogió el aula Q/E403 (Figura 3.26).

Para verificar que los cambios realizados se efectuaron exitosamente en el sistema, se muestra en pantalla la información recuperada de la base de datos (Figura 3.27).

Adicionalmente, si el cambio no se efectuó correctamente en cualquiera de los casos, la aplicación informará la causa del error al usuario administrador.

3.2.2.3.3 Submódulo Reservas

Para la comprobación de este submódulo, se elige arbitrariamente un docente, materia y grupo. El docente realizará una reserva de un aula en un horario específico; para ello una vez seleccionada la fecha o el periodo de reserva, se procede a seleccionar el aula. En este caso se selecciona el aula “ELEE206” de la lista presentada, consecuentemente se observa en pantalla el horario de la misma, incluyendo horarios fijos y reservas ya realizadas, las mismas que presentarán otro color.

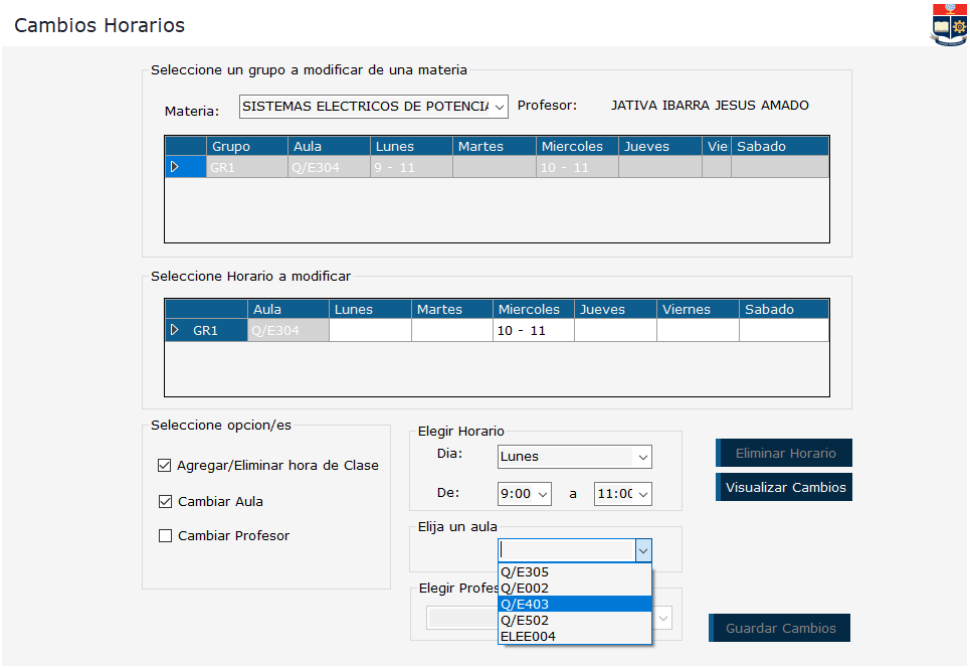


Figura 3.26. Selección de grupo sujeto a modificación de docente y selección del nuevo horario y nueva aula

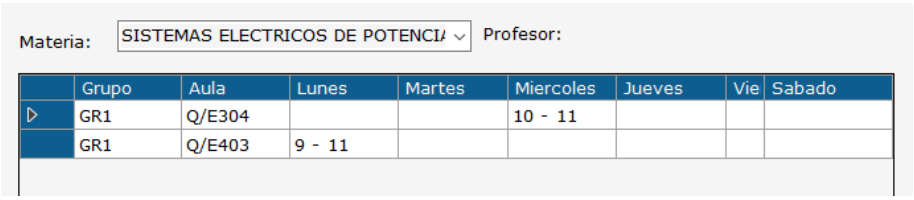


Figura 3.27. Visualización de cambio de horario y aula efectuado exitosamente

Posteriormente se procede a escoger el horario deseado, mismo que debe estar dentro del rango de tiempo seleccionado; para la presente prueba será el día jueves de 9:00 a 11:00. Cabe mencionar, que una vez seleccionado el día, la aplicación mostrará en pantalla únicamente las horas en las cuales el aula se encuentra disponible basándose en los datos proporcionados (Figura 3.28) .

Este proceso puede realizarse de un modo alterno, el cual consiste en seleccionar la fecha o periodo de reserva seguido del horario deseado, para finalmente seleccionar un aula de la lista de aulas disponibles según los parámetros elegidos.

Para verificar que la reserva se realizó exitosamente se muestra en pantalla el horario del aula para la semana de la fecha reservada, en donde se observan las horas seleccionadas resaltadas con color celeste (Figura 3.29).

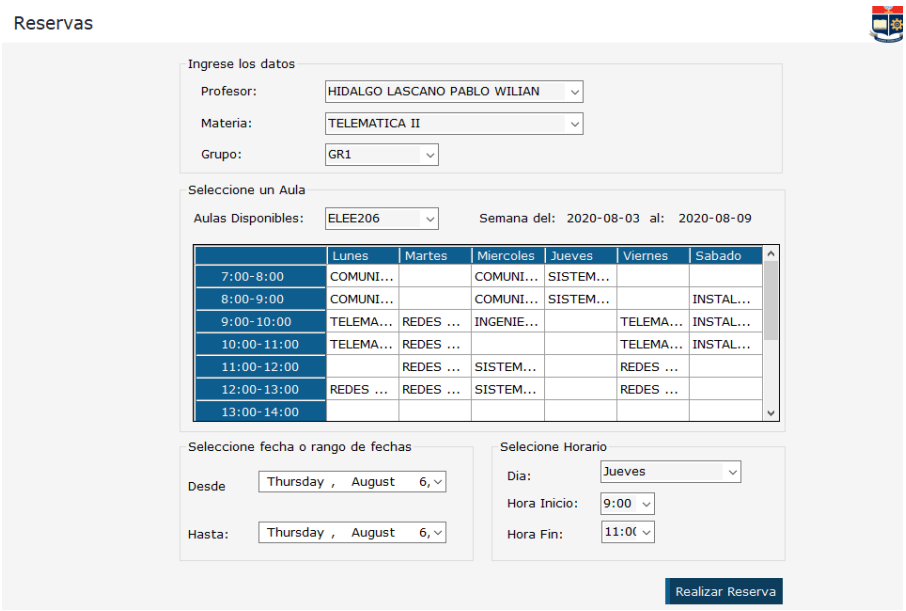


Figura 3.28. Selección de datos para la realización de la reserva

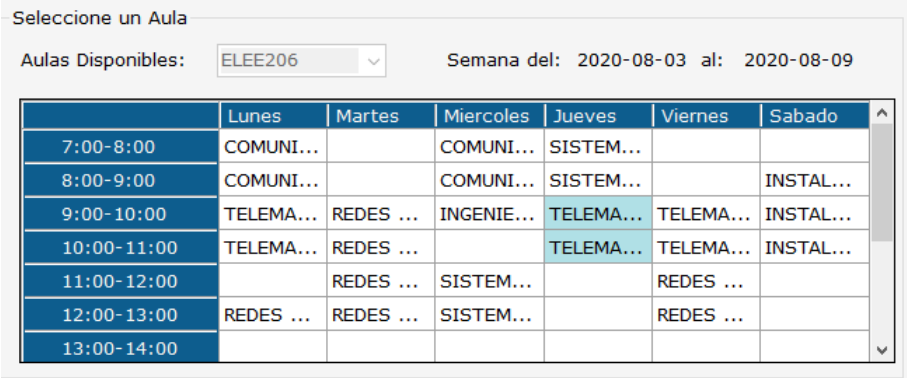


Figura 3.29. Visualización de reserva realizada exitosamente

3.2.2.3.4 Submódulo Solicitudes de Reserva

Para la comprobación de este submódulo, previamente se generó una solicitud de reserva desde la aplicación móvil y se comprobó que los datos estaban correctos al recuperarse en pantalla (Figura 3.30).

En orden de aprobar o denegar una solicitud, se debe dar clic en el botón atender, el cual redirigirá a la pantalla del submódulo Reservas, en donde se llenarán automáticamente todos los campos en base a los datos de la solicitud; si es que no existe ningún inconveniente con los datos de la reserva, como ocurrió con la presente prueba, se presiona el botón Realizar Reserva (Figura 3.31), caso contrario se mostrará un mensaje informando sobre la causa del conflicto y se retorna al submódulo Solicitudes de reserva para proceder a denegar la misma.

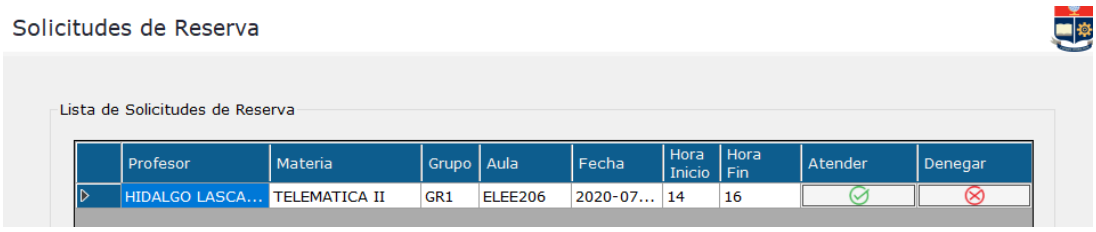


Figura 3.30. Lista de solicitudes pendientes

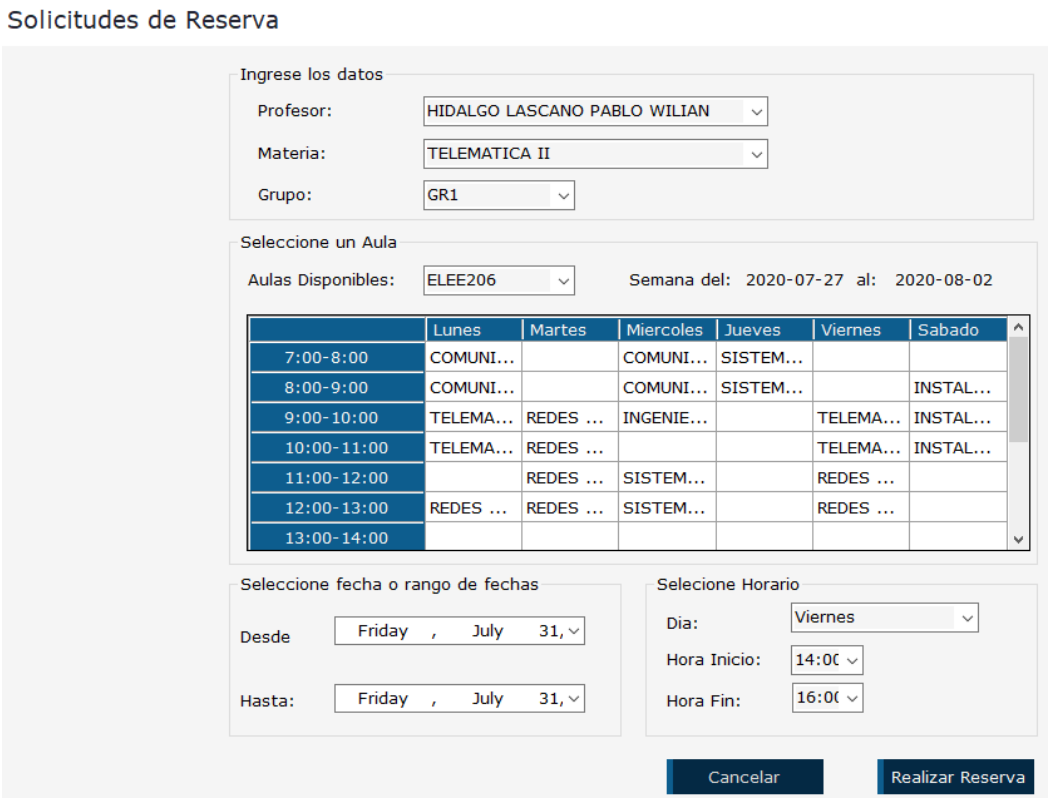


Figura 3.31. Solicitud de reserva correcta para ser aprobada

3.2.2.4 Módulo Dispositivos

3.2.2.4.1 Submódulo Agregar Dispositivos

Para la comprobación de este submódulo, se inicia con la verificación de la conexión del terminal biométrico que será asignado a un aula y posteriormente agregado a la base de datos; si esta conexión es exitosa, se recupera información del dispositivo y se muestra en pantalla, posteriormente se habilita el control donde se selecciona el aula a asignar como se observa en la Figura 3.32.

En este submódulo se verifica que la dirección IP sea correcta y que no se encuentre utilizada por otro dispositivo; adicionalmente se comprueba que las aulas presentadas no se encuentren asignadas a ningún terminal biométrico.

Figura 3.32. Recuperación de información del dispositivo posterior a la conexión con el mismo

Se puede verificar la correcta agregación del dispositivo a la base de datos de la Figura 3.8. En caso de existir alguna falla en la operación, el usuario será informado de la causa del error.

3.2.2.4.2 Submódulo Manejo de dispositivos

Para la comprobación de este submódulo, se pone a prueba a cada una de las funciones del mismo, documentando cada acción. Para ello los terminales biométricos ingresados al sistema deben encontrarse encendidos y en red.

- **Verificación de estado**

Una vez seleccionado el dispositivo sobre el cual se efectuará la acción, se debe presionar el botón correspondiente. Posteriormente se mostrará en pantalla toda la información referente al dispositivo como se observa en la Figura 3.33.

Si el estado de conexión del dispositivo es “Desconectado”, algunas funciones del submódulo serán restringidas, puesto que para efectuarlas se requiere que el dispositivo se encuentre conectado (Figura 3.34).

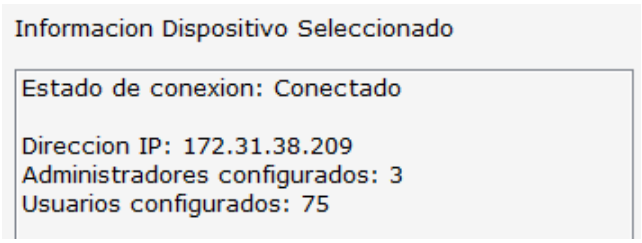


Figura 3.33. Información referente al dispositivo seleccionado

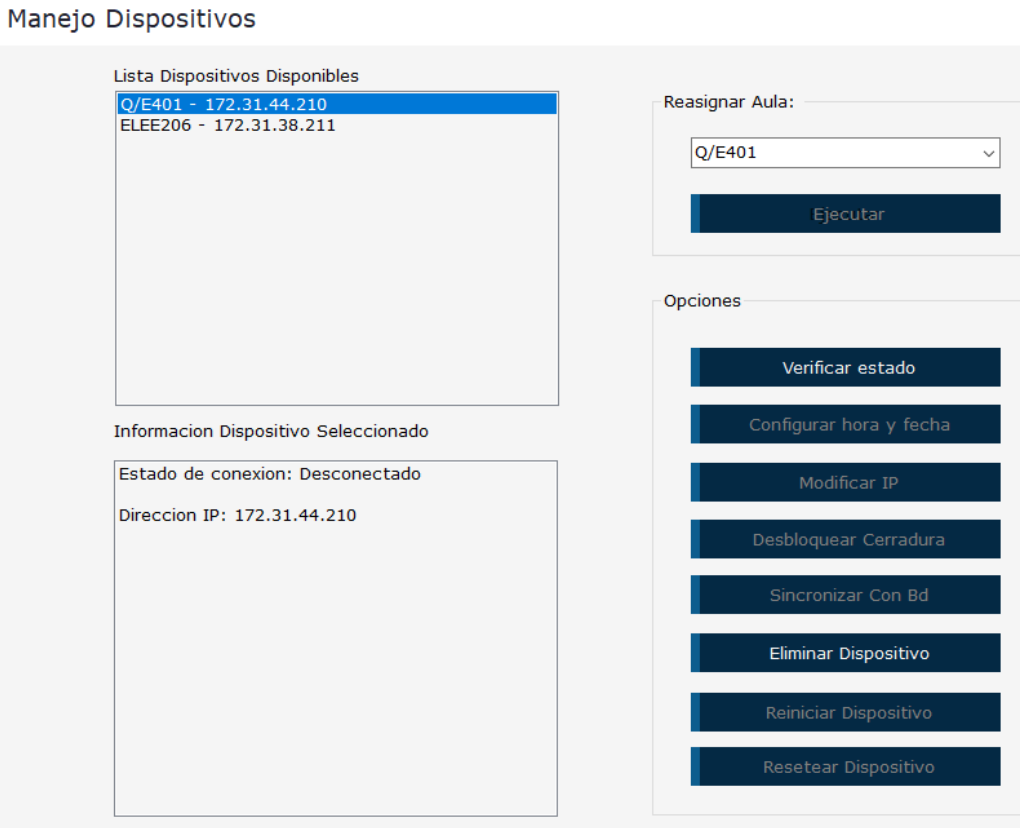


Figura 3.34. Restricción de funciones para dispositivo desconectado

- **Configuración de hora y fecha**

Una vez seleccionado el dispositivo sobre el cual se efectuará la acción, si este se encuentra conectado, se debe presionar el botón correspondiente. Seguido se mostrará en pantalla el mensaje de la Figura 3.35.

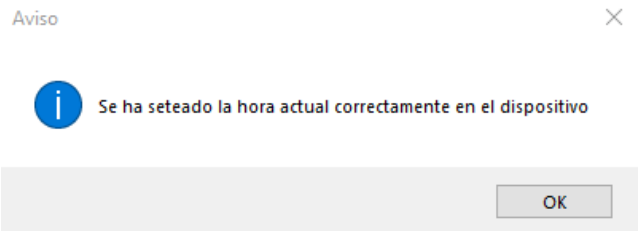


Figura 3.35. Mensaje informativo de acción efectuada con éxito

Para la presente prueba, el dispositivo se encontraba con fecha de 21 de Julio del 2020 y hora de 20:36 (Figura 3.36 (a)) las cuales diferían de la fecha y hora en las cuales se realizó la prueba. Al efectuar la acción, se configuró la fecha y hora del dispositivo con los datos correctos al 22 de julio del 2020 con hora 8:55 (Figura 3.36 (b)).

(a) (b)

Figura 3.36. Fotografía de la configuración Hora y Fecha capturada del dispositivo

- **Modificación de Dirección IP**

Una vez seleccionado el dispositivo sobre el cual se efectuará la acción, si éste se encuentra conectado, se debe presionar el botón correspondiente. Seguido se mostrará un cuadro diálogo en pantalla (Figura 3.37) solicitando la nueva dirección IP que se le asignará al dispositivo.

El sistema verifica que la dirección IP sea correcta y no se encuentre ocupada por otro dispositivo.

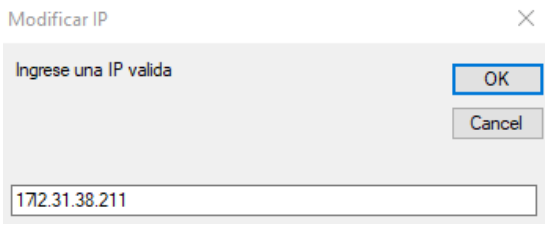


Figura 3.37. Cuadro de diálogo para ingresar la nueva dirección IP

Posteriormente se verifica el cambio realizado tanto en la base de datos (Figura 3.39) como en el dispositivo (Figura 3.38).

Figura 3.38. Verificación nueva dirección IP en el dispositivo

Results Messages

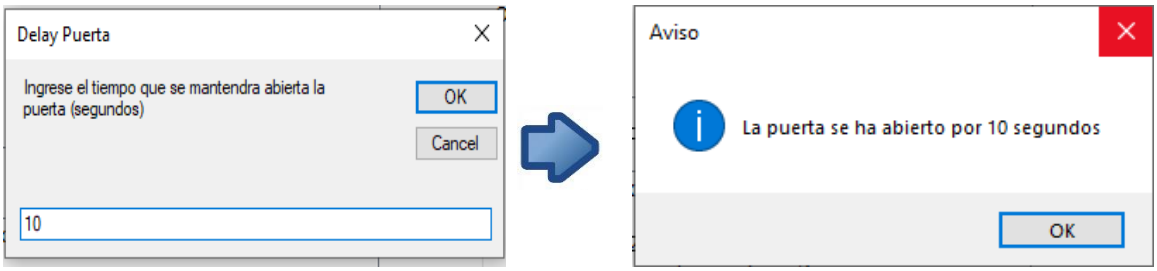
	idDisp	idAula	ip	mac	codigo	fabricante	tipo	serial
1	1	16	172.31.38.211	00:17:61:10:98:11	K50	ZKTeco Inc.	ZLM60_TFT	A2QO193260106

Figura 3.39. Verificación nueva dirección IP en la base de datos

- **Desbloqueo de cerradura**

Una vez seleccionado el dispositivo sobre el cual se efectuará la acción, si éste se encuentra conectado, se debe presionar el botón correspondiente. Seguido se mostrará un cuadro diálogo en pantalla (Figura 3.40 (a)) solicitando el tiempo en segundos que la cerradura se encontrará desbloqueada.

Una vez ingresado este dato, se muestra en pantalla el mensaje de que la operación fue exitosa (Figura 3.40 (b)).



(a) (b)

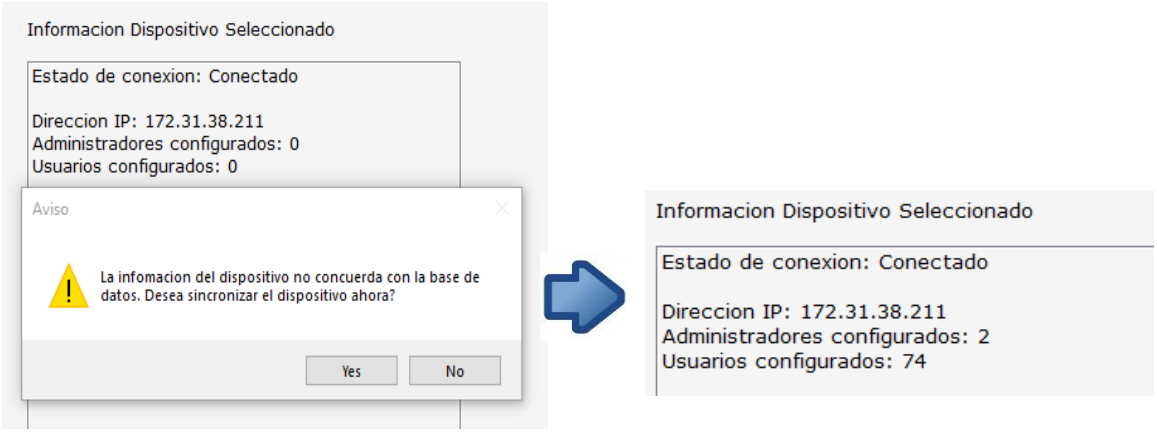
Figura 3.40. Proceso para desbloquear cerradura

- **Sincronización con base de datos**

Una vez seleccionado el dispositivo, se puede observar en pantalla la información correspondiente al mismo.

Al presionar el botón para efectuar la acción, se mostrará un mensaje en pantalla. Si los datos presentes en el dispositivo no concuerdan con los de la base de datos, se le informará de esto al usuario administrador y adicionalmente se le preguntará si desea sincronizar el dispositivo (Figura 3.41 (a)). Caso contrario se informa al usuario administrador que el dispositivo se encuentra sincronizado.

Posteriormente se verifica que la acción se efectuó con éxito, mediante la revisión de la información en el dispositivo (Figura 3.41 (b)).



(a) (b)

Figura 3.41. Proceso de sincronización del dispositivo con la base de datos

- **Eliminar dispositivo**

Una vez seleccionado el dispositivo sobre el cual se efectuará la acción, se debe presionar el botón correspondiente. Seguido se mostrará un mensaje informando al usuario administrador que la operación se efectuó correctamente (Figura 3.42 (b)), esto quiere decir que el dispositivo fue removido por completo del sistema.

Para verificar que la acción fue exitosa, se documenta la tabla de dispositivos en la base de datos antes de la acción (Figura 3.42 (a)) y después de la misma (Figura 3.42 (c)) .

- **Reiniciar el dispositivo**

Una vez seleccionado el dispositivo sobre el cual se efectuará la acción, se debe presionar el botón correspondiente. Seguido se mostrará un mensaje informando al usuario administrador que la operación se efectuó con éxito (Figura 3.43 (a)).

Se puede corroborar en el dispositivo que la acción se realizó correctamente puesto que en la pantalla de éste se carga el inicio del dispositivo (Figura 3.43 (b)).

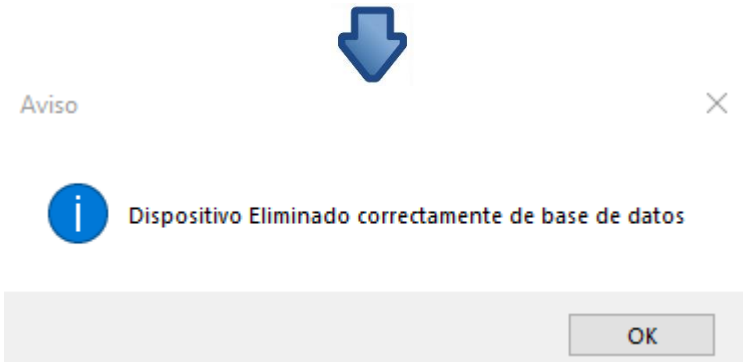
- **Resetear el dispositivo**

Una vez seleccionado el dispositivo sobre el cual se efectuará la acción, se debe presionar el botón correspondiente. Seguido se mostrará un mensaje informando al usuario administrador que la operación se realizó correctamente (Figura 3.44 (b)), es decir se restableció la información almacenada en el dispositivo.

Figura 3.42. Proceso para eliminar un dispositivo

idDisp	idAula	ip	mac	codigo	fabricante	tipo	serial
4	2	172.31.38.211	00:17:61:10:98:11	K50	ZKTeco Inc.	ZLM60_TFT	A2QO193260106
5	3	172.31.44.210	00:17:61:AE:37:A6	K20	ZKTeco Inc.	JZ4725_TFT	A8LN183960236

(a)



(b)

	idDisp	idAula	ip	mac	codigo	fabricante	tipo	serial
1	5	3	172.31.44.210	00:17:61:AE:37:A6	K20	ZKTeco Inc.	JZ4725_TFT	A8LN183960236

(c)

(a) (b)

Figura 3.43. Proceso para reiniciar un dispositivo

Para corroborar que la acción fue exitosa, se documenta la información en el dispositivo antes de la acción (Figura 3.44 (a)) y después de la misma (Figura 3.44 (c)).

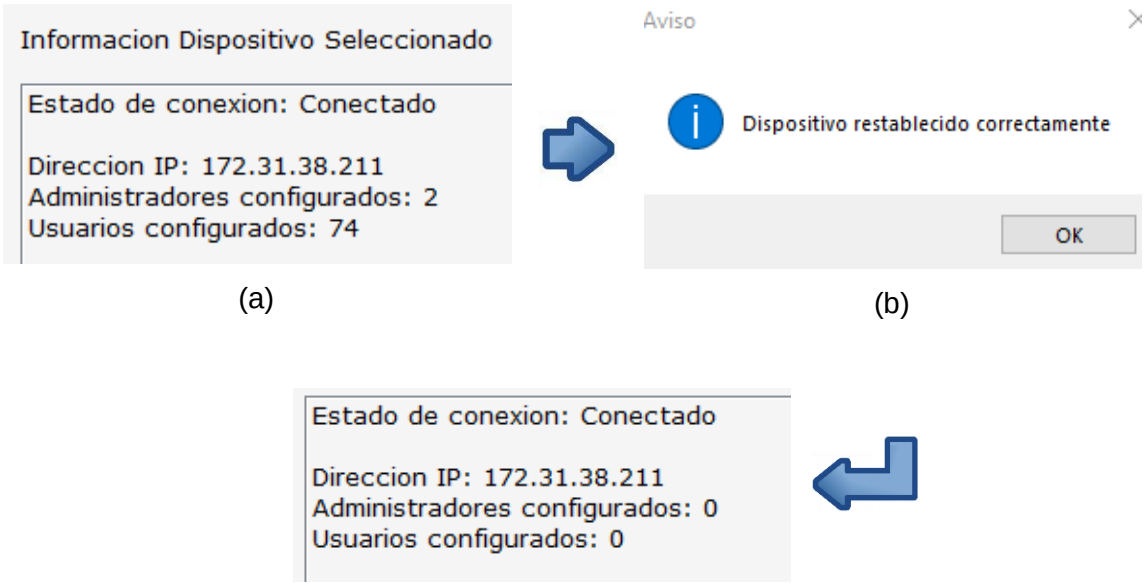


Figura 3.44. Proceso para restablecer un dispositivo

Cabe mencionar que en caso de no poder ejecutarse cualquiera de las funciones, o de que exista un error al momento de la ejecución, el sistema se encargará de informar de la falla al usuario administrador.

3.2.2.5 Módulo Reportes

Para la realización de las pruebas dentro del presente módulo, durante un periodo de tiempo de dos meses, se tomaron registros de huella de 3 docentes en sus respectivos horarios (Figura 3.45).

De igual manera, dentro de este periodo, se provocaron inasistencias arbitrarias a cada uno de ellos, con el fin de poder generar reportes y estadísticas con una muestra de datos significativa y lo más próxima a un escenario real.

En caso de presentarse fallas en cualquiera de los submódulos, se informará al usuario sobre la causa del error.

Figura 3.45. Generación de registro de asistencia

3.2.2.5.1 Reportes de Asistencia

Para la comprobación de este submódulo, una vez seleccionado el docente, se eligió el periodo de tiempo sobre el cual se quiere obtener el reporte. Opcionalmente se puede filtrar los registros obtenidos de la base de datos por materia y grupo.

Los registros que se encontraron dentro del intervalo de tiempo seleccionado son mostrados en pantalla, así como también las estadísticas correspondientes (Figura 3.46).

La información mostrada en pantalla puede ser exportada a un documento Excel o PDF según se requiera, mediante los botones respectivos.

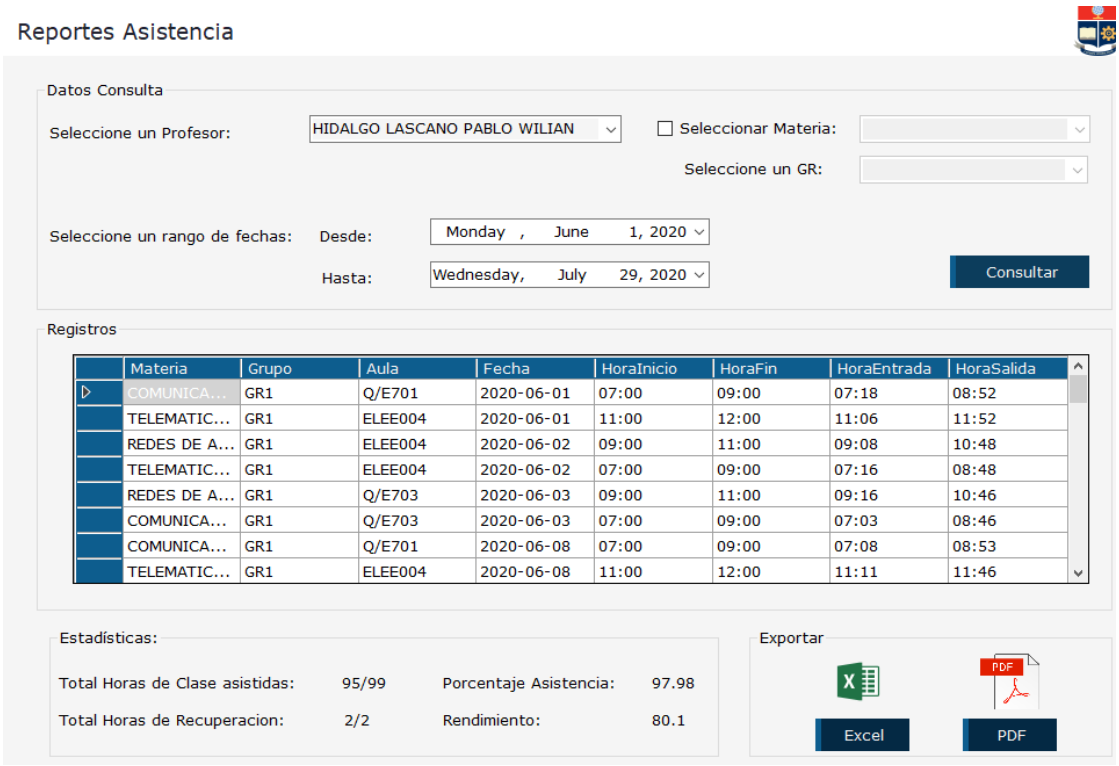


Figura 3.46. Registros obtenidos mediante consulta

En la Figura 3.47 y la Figura 3.48 se puede observar parte del documento Excel generado a partir de la aplicación. El documento estará compuesto por el encabezado, la fecha en la que se genera el reporte, los datos de consulta, los registros realizados, y las estadísticas. Cabe mencionar que los registros resaltados en color celeste corresponden a registros realizado en horarios de reserva.

En la Figura 3.49 se puede observar el documento PDF generado a partir de la aplicación, mismo que está compuesto de la misma información que el archivo Excel.

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA Y ELECTRONICA

Periodo: 2020-06-01 - 2020-07-29 Fecha: 2020-07-29

REGISTRO DE ASISTENCIA

Profesor: 1801537448 - HIDALGO LASCANO PABLO WILIAN

Materia	Grupo	Aula	Fecha	HoraInicio	HoraFin	HoraEntrada	HoraSalida
COMUNICACION	GR1	Q/E701	2020-06-01	07:00	09:00	07:18	08:52
TELEMATICA II	GR1	ELEE004	2020-06-01	11:00	12:00	11:06	11:52

Figura 3.47. Encabezado documento

COMUNICACIOI	GR1	Q/E703	2020-07-15	07:00	09:00	07:19	08:53
REDES DE AREA	GR1	ELEE004	2020-07-16	07:00	09:00	07:03	08:59
COMUNICACIOI	GR1	Q/E701	2020-07-20	07:00	09:00	07:02	08:49
TELEMATICA II	GR1	ELEE004	2020-07-20	11:00	12:00	11:19	11:47
REDES DE AREA	GR1	ELEE004	2020-07-21	09:00	11:00	09:04	10:53
TELEMATICA II	GR1	ELEE004	2020-07-21	07:00	09:00	07:01	08:58
REDES DE AREA	GR1	Q/E703	2020-07-22	09:00	11:00	09:19	10:55
COMUNICACIOI	GR1	Q/E703	2020-07-22	07:00	09:00	07:14	08:46
COMUNICACIOI	GR1	Q/E701	2020-07-27	07:00	09:00	07:09	08:51
TELEMATICA II	GR1	ELEE004	2020-07-27	11:00	12:00	11:12	11:45
REDES DE AREA	GR1	ELEE004	2020-07-28	09:00	11:00	09:12	10:56
TELEMATICA II	GR1	ELEE004	2020-07-28	07:00	09:00	07:15	08:50
REDES DE AREA	GR1	Q/E703	2020-07-29	09:00	11:00	09:03	10:50
COMUNICACIOI	GR1	Q/E703	2020-07-29	07:00	09:00	07:14	08:45

ESTADÍSTICAS:

Total Horas de Clase Asistidas:	**Materia**	**Grupo**	**Asistidas**	**Obligatorias**
	REDES DE AREA EXTENDI	GR1	34	36
	COMUNICACION DIGITAL	GR1	36	36
	TELEMATICA II	GR1	25	27
		TOTAL:	95	99

Total Horas de Recuperación:	**Materia**	**Grupo**	**Asistidas**	**Obligatorias**
	REDES DE AREA EXTENDI	GR1	2	2
		TOTAL:	2	2

Porcentaje Asistencia:	97.98
Rendimiento:	80.1

Figura 3.48. Registros y estadísticas

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA Y ELECTRONICA

Periodo: 2020-06-01 - 2020-07-29 Fecha: 2020-07-29

REGISTRO DE ASISTENCIA

Profesor: 1801537448 - HIDALGO LASCANO PABLO WILIAN

Materia	Grupo	Aula	Fecha	HoraInicio	HoraFin	HoraEntrada	HoraSalida
COMUNICACION I	GR1	Q/E701	2020-06-01	07:00	09:00	07:18	08:52
TELEMATICA II	GR1	ELEE004	2020-06-01	11:00	12:00	11:06	11:52
REDES DE AREA E.	GR1	ELEE004	2020-06-02	09:00	11:00	09:08	10:48
TELEMATICA II	GR1	ELEE004	2020-06-02	07:00	09:00	07:16	08:48
REDES DE AREA E.	GR1	Q/E703	2020-06-03	09:00	11:00	09:16	10:46
COMUNICACION I	GR1	Q/E703	2020-06-03	07:00	09:00	07:03	08:46
COMUNICACION I	GR1	Q/E701	2020-06-08	07:00	09:00	07:08	08:53
TELEMATICA II	GR1	ELEE004	2020-06-08	11:00	12:00	11:11	11:46
REDES DE AREA E.	GR1	ELEE004	2020-06-09	09:00	11:00	09:18	10:46
TELEMATICA II	GR1	ELEE004	2020-06-09	07:00	09:00	07:09	08:56
REDES DE AREA E.	GR1	Q/E703	2020-06-10	09:00	11:00	09:12	10:49
COMUNICACION I	GR1	Q/E703	2020-06-10	07:00	09:00	07:03	08:45
COMUNICACION I	GR1	Q/E701	2020-06-15	07:00	09:00	07:15	08:54
TELEMATICA II	GR1	ELEE004	2020-06-15	11:00	12:00	11:13	11:57
REDES DE AREA E.	GR1	ELEE004	2020-06-16	09:00	11:00	09:02	10:57
TELEMATICA II	GR1	ELEE004	2020-06-16	07:00	09:00	07:12	08:52
REDES DE AREA E.	GR1	Q/E703	2020-06-17	09:00	11:00	09:05	10:48
COMUNICACION I	GR1	Q/E703	2020-06-17	07:00	09:00	07:04	08:45
COMUNICACION I	GR1	Q/E701	2020-06-22	07:00	09:00	07:07	08:50
TELEMATICA II	GR1	ELEE004	2020-06-22	11:00	12:00	11:12	11:57
REDES DE AREA E.	GR1	ELEE004	2020-06-23	09:00	11:00	09:04	10:50
TELEMATICA II	GR1	ELEE004	2020-06-23	07:00	09:00	07:05	08:47
REDES DE AREA E.	GR1	Q/E703	2020-06-24	09:00	11:00	09:17	10:57
COMUNICACION I	GR1	Q/E703	2020-06-24	07:00	09:00	07:02	08:46
COMUNICACION I	GR1	Q/E701	2020-06-29	07:00	09:00	07:11	08:48
TELEMATICA II	GR1	ELEE004	2020-06-29	11:00	12:00	11:03	11:55
REDES DE AREA E.	GR1	ELEE004	2020-06-30	09:00	11:00	09:11	10:55
TELEMATICA II	GR1	ELEE004	2020-06-30	07:00	09:00	07:18	08:57
REDES DE AREA E.	GR1	Q/E703	2020-07-01	09:00	11:00	09:12	10:45
COMUNICACION I	GR1	Q/E703	2020-07-01	07:00	09:00	07:17	08:52
COMUNICACION I	GR1	Q/E701	2020-07-06	07:00	09:00	07:17	08:49
TELEMATICA II	GR1	ELEE004	2020-07-06	11:00	12:00	11:15	11:50
REDES DE AREA E.	GR1	ELEE004	2020-07-07	09:00	11:00	09:19	10:57
TELEMATICA II	GR1	ELEE004	2020-07-07	07:00	09:00	07:11	08:50
REDES DE AREA E.	GR1	Q/E703	2020-07-08	09:00	11:00	09:14	10:57
COMUNICACION I	GR1	Q/E703	2020-07-08	07:00	09:00	07:11	08:48
COMUNICACION I	GR1	Q/E701	2020-07-13	07:00	09:00	07:10	08:45
TELEMATICA II	GR1	ELEE004	2020-07-13	11:00	12:00	11:18	11:45
REDES DE AREA E.	GR1	Q/E703	2020-07-15	09:00	11:00	09:05	10:48
COMUNICACION I	GR1	Q/E703	2020-07-15	07:00	09:00	07:19	08:53
REDES DE AREA E.	GR1	ELEE004	2020-07-16	07:00	09:00	07:03	08:59
COMUNICACION I	GR1	Q/E701	2020-07-20	07:00	09:00	07:02	08:49
TELEMATICA II	GR1	ELEE004	2020-07-20	11:00	12:00	11:19	11:47
REDES DE AREA E.	GR1	ELEE004	2020-07-21	09:00	11:00	09:04	10:53
TELEMATICA II	GR1	ELEE004	2020-07-21	07:00	09:00	07:01	08:58
REDES DE AREA E.	GR1	Q/E703	2020-07-22	09:00	11:00	09:19	10:55
COMUNICACION I	GR1	Q/E703	2020-07-22	07:00	09:00	07:14	08:46
COMUNICACION I	GR1	Q/E701	2020-07-27	07:00	09:00	07:09	08:51
TELEMATICA II	GR1	ELEE004	2020-07-27	11:00	12:00	11:12	11:45
REDES DE AREA E.	GR1	ELEE004	2020-07-28	09:00	11:00	09:12	10:56
TELEMATICA II	GR1	ELEE004	2020-07-28	07:00	09:00	07:15	08:50
REDES DE AREA E.	GR1	Q/E703	2020-07-29	09:00	11:00	09:03	10:50
COMUNICACION I	GR1	Q/E703	2020-07-29	07:00	09:00	07:14	08:45

ESTADÍSTICAS:

Total Horas de Clase Asistidas:	Materia	Grupo	Asistidas	Obligatorias
	REDES DE AREA EXTENDIDA	GR1	34	36
	COMUNICACION DIGITAL	GR1	36	36
	TELEMATICA II	GR1	25	27
		TOTAL:	95	99

Total Horas de Recuperación:	Materia	Grupo	Asistidas	Obligatorias
	REDES DE AREA EXTENDIDA	GR1	2	2
		TOTAL:	2	2

Porcentaje Asistencia: 97.98

Rendimiento: 80.1

Figura 3.49. Reporte en formato PDF

3.2.2.5.2 Reportes de Inasistencia

Para la comprobación de este submódulo, se genera un reporte una vez seleccionado el docente y el periodo de tiempo (Figura 3.51).

Adicionalmente, se utiliza el filtro por materia y grupo para verificar su correcto funcionamiento (Figura 3.50).

De igual forma que en el submódulo anterior, se puede generar un documento Excel o PDF que tendrá el mismo formato.

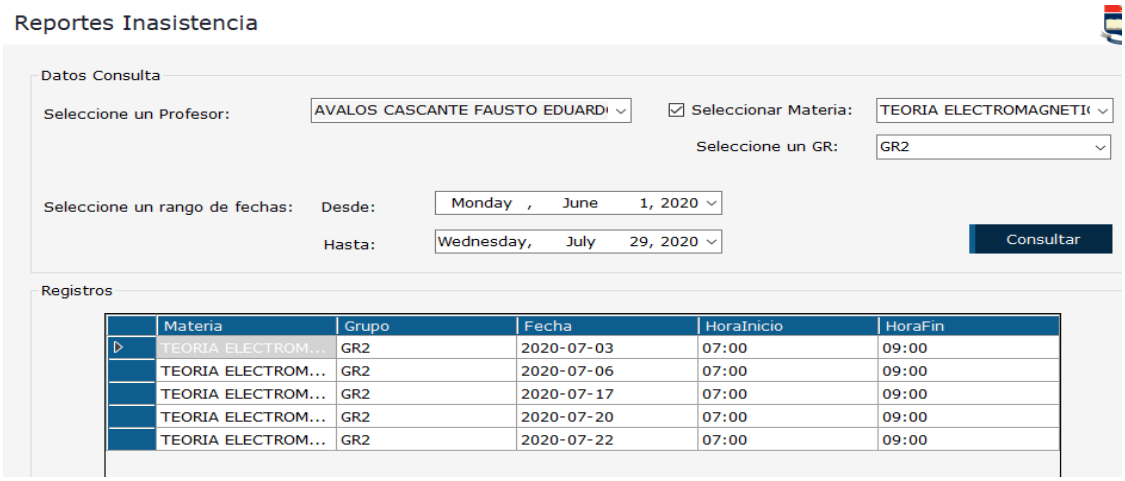


Figura 3.50. Inasistencias obtenidas mediante consulta con filtro activado

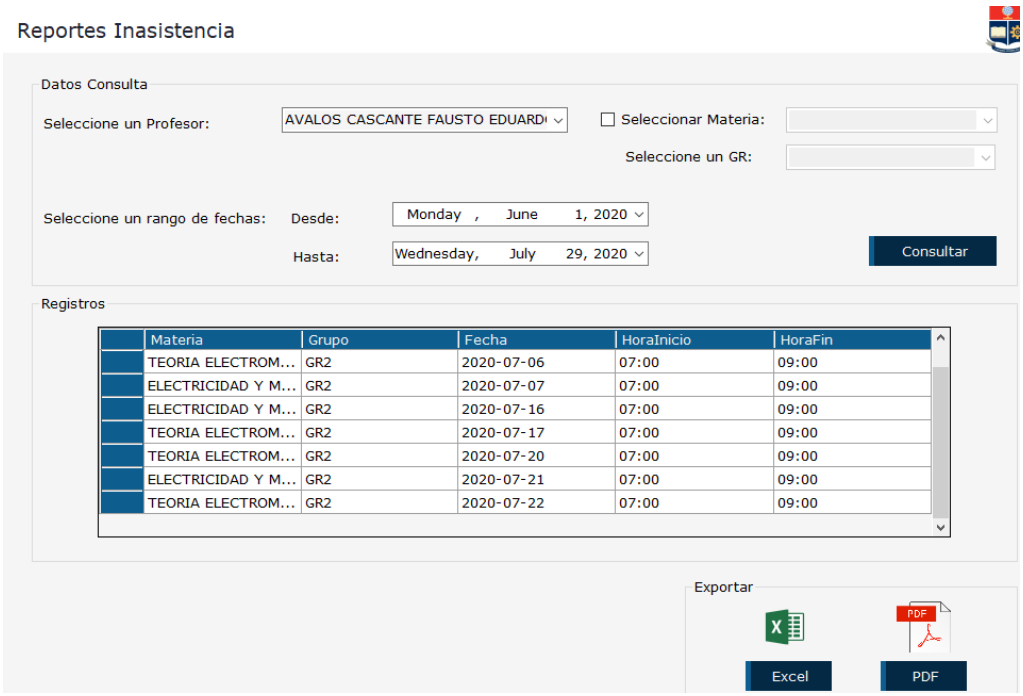


Figura 3.51. Inasistencias obtenidas mediante consulta

3.2.2.5.3 Reportes de Aulas

Para la comprobación de este submódulo, una vez seleccionada el aula y el periodo de tiempo sobre el cual se quiere obtener el reporte, se ejecuta la consulta.

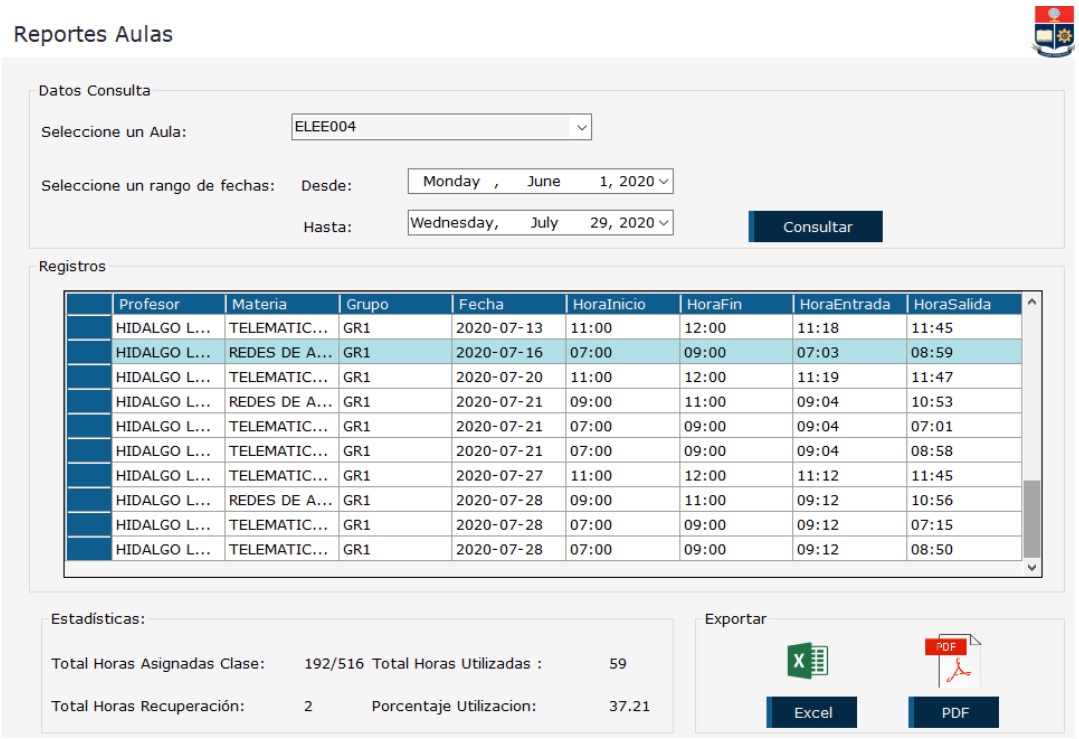


Figura 3.52. Registros en aula seleccionada obtenidos mediante consulta

Los registros que se encontraron dentro del intervalo de tiempo seleccionado son mostrados en pantalla, así como también las estadísticas correspondientes del aula (Figura 3.52). Los registros resaltados en color celeste corresponden a registros de asistencia realizados en horas de reserva.

De igual forma que en los submódulos anteriores, se puede exportar la información al formato deseado.

En base a los resultados obtenidos de las pruebas realizadas a cada uno de los submódulos, mismos que fueron positivos, se logra verificar la correcta ejecución de cada una de las funcionalidades implementadas en la aplicación, así como también la interacción exitosa entre los componentes que integran el sistema.

3.2.3 APLICACIÓN MÓVIL

En esta sección se probará la operación de todos los módulos que componen la aplicación móvil, esto adicionalmente permitirá que se verifique la correcta interacción de la aplicación con el sistema.

Para la realización de las pruebas, se inició sesión en la aplicación con las credenciales del docente “PABLO WILIAN HIDALGO LASCANO”.

3.2.3.1 Módulo Horarios de Aulas

Para la comprobación de este módulo, se selecciona el aula Q/E304, y una fecha arbitraria. Se escoge esta aula con el fin de demostrar la concordancia del resultado obtenido (Figura 3.54) con el horario presentado de la consulta para la misma aula, en el submódulo Mostrar Aulas de la aplicación de administración (Figura 3.19).

3.2.3.2 Módulo Registros de asistencia

Para la comprobación de este módulo, únicamente se ingresa a éste y se selecciona el mismo periodo de fechas que en el submódulo Reportes de Asistencia de la aplicación de administración, esto con el fin de demostrar la concordancia de resultados de la Figura 3.56 con los obtenidos en la Figura 3.46.

Opcionalmente se puede habilitar el filtro por materia y grupo al igual que en la aplicación de administración.

En caso de realizar selecciones erróneas, por ejemplo, en las fechas, la aplicación informará de los errores presentados.

Figura 3.53. Pantalla Horario de aulas

Figura 3.54. Horario de aula Q/E304

Figura 3.55. Pantalla Reportes de Asistencia

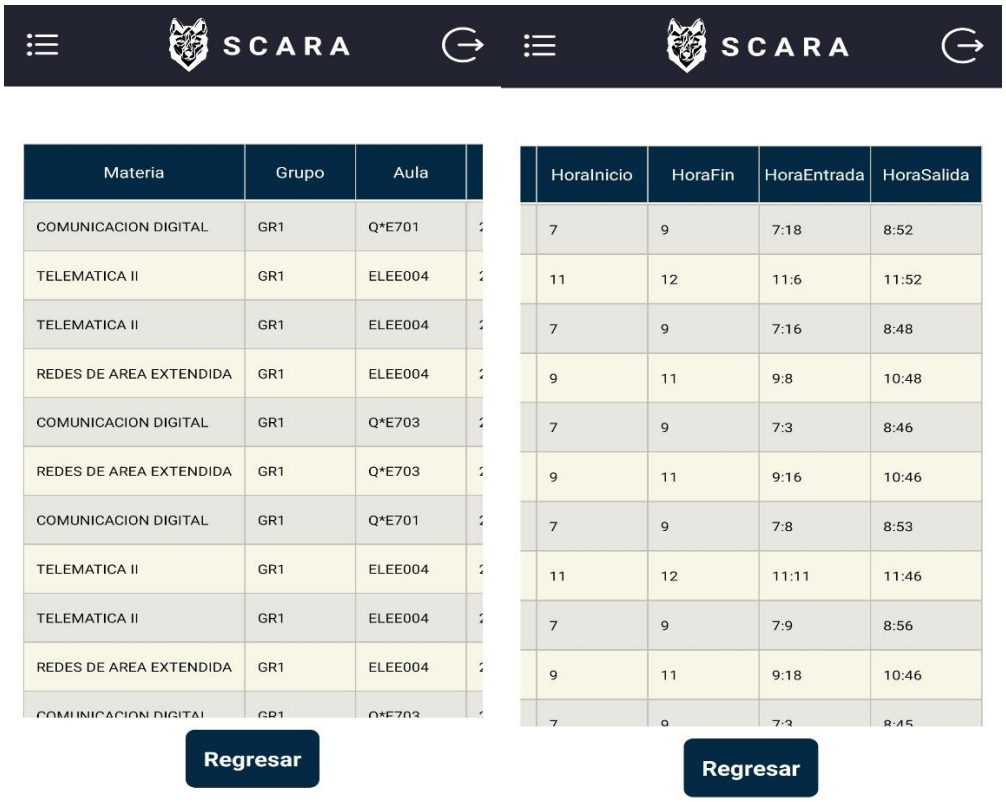


Figura 3.56. Registros de asistencia resultantes de consulta

3.2.3.3 Módulo Solicitud de reserva

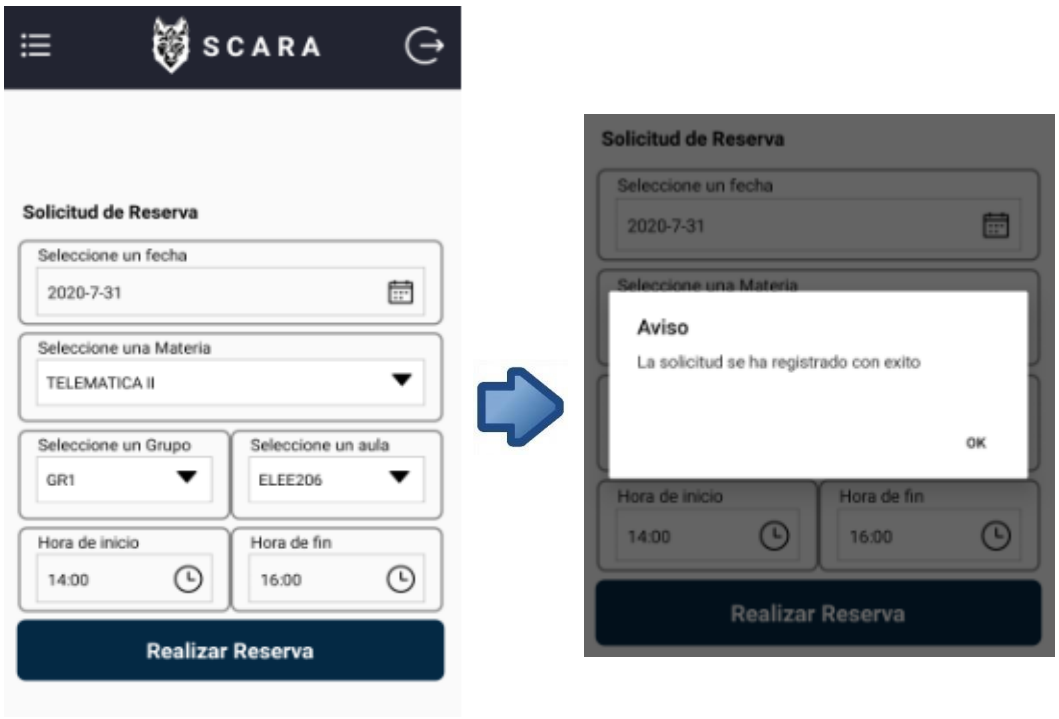
Para la comprobación de este módulo, se selecciona una fecha y una hora, conociendo previamente mediante el módulo Horarios de Aulas las horas libres del aula que se desea para esa semana. Seguidamente se selecciona la materia y el grupo para el cual se realizará la reserva (Figura 3.58 (a)). En caso de realizarse exitosamente la reserva se presenta un mensaje en pantalla, caso contrario se informa del error (Figura 3.58 (b)).

Cabe mencionar que, al iniciar sesión en la pantalla de inicio, se muestra al usuario las solicitudes que tiene a futuro y el estado de las mismas. En la Figura 3.57 se observa la pantalla de inicio antes de realizar la reserva.

Figura 3.57. Pantalla de inicio

Hasta que la solicitud sea atendida por el usuario administrador el estado de la misma permanecerá en pendiente como se muestra en la Figura 3.59 (a).

Una vez recibida la solicitud por el usuario administrador como se muestra en la Figura 3.30, ésta debe ser atendida (Figura 3.31). Si la solicitud es aprobada, se actualizará el estado de la misma en la pantalla inicio de la aplicación móvil (Figura 3.59 (b)).



(a) (b)

Figura 3.58. Proceso de generación solicitud de reserva

(a) (b)

Figura 3.59. Cambio de estado de la solicitud en la pantalla de inicio

En caso de existir algún error en los datos seleccionados, o que el horario seleccionado se encuentre ocupado, la aplicación informará al usuario sobre el mismo.

Con respecto a las pruebas realizadas a la aplicación móvil, se constata la correcta ejecución de las acciones que ésta realiza, así como también la exitosa integración de Firebase con la misma.

4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- Se dispone de un prototipo que cumple el objetivo de un control adecuado y optimizado sobre el acceso a aulas y el registro de asistencia a clases del personal docente. El prototipo se encuentra conformado por: Lectores biométricos de huella dactilar, cerraduras electromagnéticas, un servidor centralizado que aloja el servicio web, una aplicación para administración, una aplicación móvil, y bases de datos alojadas en Firebase y SQL server para el almacenamiento de información.
- Este trabajo de titulación fue realizado basándose en las necesidades particulares de la FIEE; mediante el mismo se logra una solución eficiente que agiliza y simplifica procesos para el personal docente y administrativo, puesto que se eliminan tareas que resultan tediosas y conllevan tiempo si son efectuadas de forma manual. Adicionalmente se disminuye el uso de recursos humanos y físicos que generalmente son utilizados para la realización de este proceso.
- El sistema proporciona un adecuado tratamiento de la información, incrementando la integridad, fiabilidad y disponibilidad de los datos, mediante la disminución de la probabilidad de existencia de errores humanos. De igual forma la FIEE, se beneficiaría con data histórica confiable, que puede ser obtenida de forma rápida y sencilla, lo cual facilitará la toma de decisiones en pro de la facultad.
- Mediante esta herramienta, la FIEE se adaptaría a la tendencia actual del uso de dispositivos tecnológicos tales como los sistemas biométricos en centros educativos, lo cual tiene como propósito elevar el nivel de seguridad de las instalaciones y brindar un ambiente de confianza para el personal y los estudiantes.
- Se realizó un análisis comparativo para la adquisición de los dispositivos requeridos en el sistema, donde se evaluaron las principales características técnicas y el costo de las distintas opciones, para poder escoger la opción que presentaba una mejor relación costo-beneficio. Por esta razón la implementación del prototipo a mayor escala dentro de la facultad representaría una propuesta económica, moderna y eficaz.
- La aplicación de administración permite al usuario administrador gestionar todo el sistema lo que comprende las siguientes actividades: El manejo de datos de cualquier usuario; la adición y control de terminales biométricos de manera remota, realizar cambios de horarios de clases, gestionar reservas de aulas de clases, generar reportes de registros de asistencia e inasistencia de docentes, así como

también reportes de utilización de aulas; adicionalmente permite actualizar la información de la base de datos para un nuevo semestre.

- La aplicación móvil posibilita al usuario revisar horarios de aulas y sus registros de asistencia mismos que son actualizados constantemente, además de realizar solicitudes de reserva de aulas, con lo cual se simplificaría este proceso, de manera que el usuario no necesitaría realizar personalmente dicho trámite.
- La implementación del servidor resulta útil no solo para realizar el montaje del servicio web, sino también para mantener un constante control de la comunicación con los terminales biométricos, además de servir como puente para el intercambio de información entre éstos y la aplicación de administración.
- Un servicio web permite la interoperabilidad entre distintas aplicaciones, además para el caso específico de un servicio SOAP, las aplicaciones en Windows permiten la serialización de XML y su operación contraria de manera automática, por lo cual un desarrollador no necesita preocuparse por dicha operación.
- El uso de React Native permitió un desarrollo eficiente en lo que a la aplicación móvil se refiere, debido a que permite construirlas sin conocimiento de lenguajes propios de cada una de sus plataformas actuales, como Java (Android) y Swift (IOs), lo que permitiría desarrollar la versión para los teléfonos de Apple en un futuro. Además, al desarrollar una aplicación nativa, permite un mejor desempeño de la misma tanto de manera funcional como visual.
- El lenguaje C# es muy versátil, en consecuencia, permite crear un sinfín de aplicaciones, además cuenta con una gran variedad de librerías, con lo cual se pudo crear a prácticamente todo el sistema implementado e interconectar cada una de sus piezas entre sí.
- Mediante el análisis comparativo realizado, se tomó la mejor opción basada en confiabilidad y costos para la selección del dispositivo a implementar en el sistema. Los lectores de huella dactilar presentan innumerables ventajas frente a otros dispositivos, tales como: Evitar la suplantación de identidad; la característica biométrica seleccionada es una de las más utilizadas y de menor rango de error, además es inherente al individuo lo cual elimina el riesgo de pérdida, sustracción, u olvido como en el caso de las tarjetas de identificación, tarjetas RFID, o contraseñas; no es necesario invertir en un mantenimiento técnico y software costosos para un buen funcionamiento. Adicionalmente, la tasa de falso rechazo (FAR) de los dispositivos adquiridos es de 0.0001%, que es diez veces menor a lo que se encuentra en el mercado, lo cual incrementa la confiabilidad del prototipo presentado.

- Mediante el uso de Firebase, se tienen varios beneficios, entre los que se puede mencionar: Datos actualizados a todos los clientes al instante (*Real-Time Database*), presenta su propio sistema de reglas de seguridad para la protección de la información (*Firebase Storage*), proporciona varios métodos de autenticación entre los cuales destaca el método de email/contraseño integrado.
- La automatización de reportes permite que se elimine la probabilidad de existencia de errores humanos en la generación de los mismos. Además permite el ahorro de recursos físicos y humanos utilizados, así como también agilita la realización del proceso.
- SCRUM destaca por tener una mejor retroalimentación del trabajo realizado, ya que gracias al trabajo ejecutado en plazos de tiempo fijo denominados Sprints se tiene una revisión constante que garantiza un producto de mayor calidad.
- El modo de reservar aulas con el prototipo implementado, permitirá tener un mayor control sobre las instalaciones de la facultad, adicionalmente este facilita al docente el proceso de reservas y brinda la posibilidad de una planificación certera y ordenada al momento de apartar un aula.

4.2 RECOMENDACIONES

- El presente trabajo de titulación contempla el desarrollo de una aplicación móvil en Android. Al desarrollar esta aplicación en React Native se facilita la implementación de la versión para IOs, por lo que se recomienda a futuro integrar esta versión al sistema.
- Para el desarrollo de aplicaciones con React Native se recomienda el uso de Expo, ya que es un entorno de desarrollo universal para aplicaciones en React, y esto simplifica la fase de pruebas puesto que al compartir un enlace o código QR se puede probar las modificaciones realizadas a la aplicación en tiempo real y en varios dispositivos.
- Se recomendaría usar la versión 6.x.x de React-Native-Firebase, debido a que la versión 5.x.x hace que aparezca una serie de advertencias en la consola en el proceso de desarrollo, además ya no cuenta con soporte de los desarrolladores.
- Es recomendable usar una sola versión de React Native para desarrollar una aplicación, debido a que si se intenta mejorar a una versión actual, algunas librerías podrían quedar inutilizables.
- Al desarrollar un software en C# con Firebase, no se recomienda usar la Cloud Firestore, debido a que la librería existente no funciona correctamente para agregar información a esta base de datos.

- En el servicio web no se implementaron medidas de seguridad debido a que esto no se contempló en el alcance del proyecto, es por ello que se debería incluir algún tipo de cifrado de información para evitar que esta puede ser vulnerada.
- Se recomienda añadir una función que registre todos los movimientos realizados en la sesión de un usuario en la aplicación de administración, debido a que esto brindaría mayor seguridad al sistema.
- Se puede incluir en los submódulos de Reportes, gráficas de las estadísticas presentadas al realizar una consulta. Puesto que la representación gráfica ayudaría a una mejor y más rápida interpretación de los resultados de la consulta.
- En la aplicación móvil se podría implementar un módulo en el cual el docente pueda solicitar permiso para ausentarse remotamente, e inmediatamente este pueda programar la recuperación de las horas de clase no dictadas.
- En la aplicación móvil se podría implementar un módulo en el cual el docente pueda justificar faltas remotamente, de manera que este pueda subir los documentos requeridos para explicar su ausencia.
- El prototipo se diseñó para funcionar en la Facultad de Ingeniería Eléctrica y Electrónica, sin embargo, se recomienda extenderlo al resto de facultades de la Escuela Politécnica Nacional.
- El prototipo se diseñó exclusivamente para aulas de clase, por esta razón se ha excluido todo lo relacionado con laboratorios. Se recomienda ampliar el alcance del prototipo para brindar un mayor control en lo que respecta a laboratorios.
- Es recomendable que en los futuros trabajos de titulación a realizar en la Facultad de Eléctrica y Electrónica se busque la forma de implementar e integrar sistemas de seguridad de distinto tipo como por ejemplo sistemas de video vigilancia, alarmas, entre otros, incluyendo el presente prototipo. Esto con el propósito de brindar una mayor seguridad al entorno y a su comunidad.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] U. Dahan, «Avoid data deletion,» 2009. [En línea]. Available: https://udidahan.com/2009/09/01/dont-delete-just-dont/.

[2] M. Trigas Gallego, «Metodología SCRUM,» 2019. [En línea]. Available: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf.

[3] D. Montaña, *Sistema de Identificación Mediante Huella Digital Para el Control de Accesos a la Universidad Libre en un Entorno Web,* Bogota D.C.: Universidad Libre, 2017.

[4] C. E. Alvez, M. G. Benedetto, G. R. Etchart, L. J. Luna, C. R. Leal, M. A. Fernández, G. L. Berón y S. R. Loggio, «Identificación de personas mediante Sistemas Biométricos. Estudio de factibilidad y su implementación en organismos estatales,» *Ciencia, Docencia y Tecnología Suplemento,* vol. 4, nº 4, pp. 48-71, 2014.

[5] A. Balsero, *Diseño E Implementación De Un Prototipo Para El Control De Acceso En La Sede De Ingeniería De La Universidad Distrital Francisco José De Caldas Mediante El Uso De Torniquetes Controlados Por Carnet Con Tecnologia Nfc Y Lector Biométrico De Huella Dactilar,* Bogota D.C.: Universidad Distrital Francisco José de Caldas, 2016.

[6] Y. Velasco y M. Villacrés, *Desarrollo Del Sistema Control Biométrico De Docentes De La Universidad Central Del Ecuador,* Quito: Universidad Central del Ecuador, 2012.

[7] C. Carrizales, *Sistema de Huella Digital Para el Registro de Asistencia de la D.C.B,* México, D.F: Universidad Nacional Autónoma de México, 2011.

[8] K. P. Tripathi, *A Comparative Study of Biometric Technologies with Reference to Human Interface,* vol. 14, Kolhapur, India: Bharati Vidyapeeth Deemed University, 2011, pp. 10-14.

[9] J.Aching y D. Rojas, *Reconocimiento Biométrico de Huellas Dactilares y su Implementación en DSP,* Lima, Perú: Universidad Nacional Mayor de San Marcos, 2005.

[10] ZKTeco, «ZKFinger Reader SDK Development Guide C# Vesion 2.0,» Septiembre 2016. [En línea]. [Último acceso: 15 abril 2019].

[11] ZKTeco, «ZKTeco Standalone SDK Development Manual Version 2.1 Rev.A.2,» 25 Abril 2018. [En línea]. [Último acceso: 12 Junio 2019].

[12] O. Vogel, I. Arnold, A. Chughtai y T. Kehrer, Software Architecture, Germany: Sprin 2011.

[13] Z. Qin, J. Xing y X. Zheng, Software Architecture, Germani: Springer, 2008.

[14] educativeio, «edpresso,» [En línea]. Available: https://www.educative.io/edpresso/w is-firebase.

[15] C. Esplin, «howtofirebase,» [En línea]. Available: https://howtofirebase.com/what-is-firebase-fcb8614ba442.

[16] VisualStudioCode, «Visual Studio Code,» 2020. [En línea]. Available: https://code.visualstudio.com/docs/editor/whyvscode.

[17] MicrosoftDocumentation, «Microsoft Documentation,» 25 05 2018. [En línea]. Availa https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2019.

[18] V. S. Code, «Visual Studio Code,» [En línea]. Available: https://code.visualstudio.com/docs/getstarted/userinterface.

[19] F. Zammetti, Practical React Native, PA, USA: APRESS, 2018.

[20] FacebookOpenSource, «React Native,» [En línea]. Available: https://reactnative.dev/?source=post_page-----6e8a2396eea1----------.

[21] A. N. Akshat Paul, React Native for Mobile Development, Haryana, India: Apress, 2(

[22] E. Cerami, Web Services Essentials, Californica, USA: O'Reilly.

[23] T. Satpathy, A Guide to the SCRUM Body Of Knowledge, Arizona, USA: VMEdu, 2(

[24] L. Hunt, «Coding Standards for .NET,» Marzo 2007. [En línea]. Available: http://www.myp11d.com/p11d/download/policies/csharp_coding_standards.pdf.

[25] S. Corporation, «secugen,» [En línea]. Available: https://secugen.com/products/hamster-plus/.

[26] Siasa, «Tecnología de Seguridad,» [En línea]. Available: https://www.siasa.com/producto.php?prod=0100041.

[27] Z. Latinoamérica, «ZTKeco Latioamérica,» [En línea]. Available: https://www.zktecolatinoamerica.com/zk9500.

[28] S. Corporation, «SecuGen,» [En línea]. Available: https://secugen.com/products/sdk

[29] Fingerspot, «Fingerspot Flexcode Sdk,» [En línea]. Available: https://www.flexcodesdk.com/.

[30] F. T. Co, «fanraysh.en,» [En línea]. Available: https://fanraysh.en.made-in-china.com/product/ujeEqKNUwwkB/China-Fingerprint-Reader-Scanner-with-USB-Cable-ZK9500-.html.

[31] Crossmatch, «Devportal.digitalpersona,» [En línea]. Available: https://devportal.digitalpersona.com/U.are.U%20SDK%20Developer%20Guide.pdf.

[32] Hikvision, «Hikvision,» [En línea]. Available: https://www.hikvision.com/es/Products/Access-Control/Terminal/DS-K1T8003.

[33] I. ZKTeco, «ZKTeco,» [En línea]. Available: https://www.cqnetcr.com/ftpdf/uploadpdf/5280.pdf.

[34] Z. Latinoamérica, «ZKTeco Latinoamérica,» [En línea]. Available: https://www.zktecolatinoamerica.com/k20.

[35] Z. Latinoamérica, «ZKTeco Latinoamérica,» [En línea]. Available: https://zktecolatinoamerica.com/2-uncategorised?limit=5&start=90.

[36] A. Leiva, *Tratamiento Digital de Huellas Dactilares,* Cataluña: Universidad Politécnic de Cataluña.

[37] A. Lindoso, *Contribución al Reconocimiento de Huellas Dactilares Mediante Técnica de Correlación y Arquitecturas Hardware para el Aumento de Prestaciones,* Madrid: Universidad Carlos III De Madrid, 2009.

[38] T.Sabhanayagam, D. V. P. Venkatesan y D. K. Senthamaraikannan, *A Comprehens Survey on Various Biometric Systems,* vol. 13, Tamilnadu, India.: Manonmaniam Sundaranar University, 2018, pp. 2276-2297.

[39] L. Guerreros, *Sistema de Control de Asistencia Estudiante-Docente para la Faculta Ingenieria de Sistemas de la Unica,* Ica, Perú: Universidad Nacional San Luis Gonz de Ica, 2013.

[40] A. Meier y M. Kaufmann, SQL & NoSQL Databases, Alemania: Springer Vieweg, 20

[41] IBMEducation, «IBM Cloud Learn Hub,» 6 August 2019. [En línea]. Available: https://www.ibm.com/cloud/learn/relational-databases.

[42] A. Kaur, «GeeksforGeeks,» 12 12 2018. [En línea]. Available: https://www.geeksforgeeks.org/need-for-dbms/.

[43] M. Radivojevic, D. Sarka, W. Durkin, C. Coté y M. Lah, Mastering SQL Server 2017 Birmingham, U.K.: Pack Publishing Ltd., 2019.

[44] B. Forta, Teach Yourself SQL, Indianapolis, USA: SAMS, 2004.

[45] IBMCloudEducation, «IBM Cloud Learn Hub,» 06 August 2019. [En línea]. Available https://www.ibm.com/cloud/learn/nosql-databases.

[46] M. Massé, REST API Design Rulebook, California, USA: O'Reilly Media, 2012.

[47] M. Rouse, «SearchWinDesarrollo,» 2017. [En línea]. Available: https://searchwindevelopment.techtarget.com/definition/C.

[48] J. Albahari y B. Albahari, C# 7.0 in a Nutshell, California, USA: O' Reilly Media, 201

[49] MicrosoftDocumentation, «Microsoft Documentation,» 19 03 2019. [En línea]. Availa https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019.

[50] G. Aroraa, Learn C# in 7 days, Birmingham, U.K.: Pack Publishing Ltd., 2017.

[51] S. Stefanov, JavaScript Patterns, California, USA: O' Reilly Media, 2010.

[52] E. Elliott, Programming JavaScript Applications, California, USA: O'Reilly Media, 20

6. ANEXOS