UNIVERSIDAD CATÓLICA DE CUENCA

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

FACULTAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE INGENIERÍA DE SOFTWARE

Desarrollo de un aplicativo web para la gestión de laboratorios, aulas de realidad virtual y otros espacios educativos

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SOFTWARE

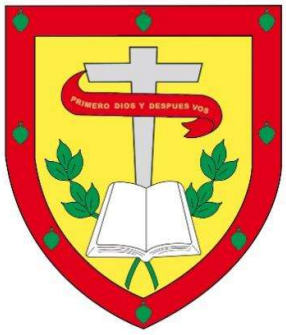
AUTOR: ADRIÁN ORLANDO ÁLVAREZ DOS SANTOS

DIRECTOR: LEOPOLDO PAUTA AYABACA

CUENCA - ECUADOR

2026

DIOS, PATRIA, CULTURA Y DESARROLLO



UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

FACULTAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE INGENIERÍA DE SOFTWARE

Desarrollo de un aplicativo web para la gestión de laboratorios, aulas de realidad virtual y otros espacios educativos

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SOFTWARE

AUTOR: ADRIÁN ORLANDO ÁLVAREZ DOS SANTOS

DIRECTOR: LEOPOLDO PAUTA AYABACA

CUENCA - ECUADOR

2026

DIOS, PATRIA, CULTURA Y DESARROLLO

Universidad Católica de Cuenca	**DECLARATORIA DE AUTORÍA Y RESPONSABILIDAD**

Adrián Orlando Álvarez Dos Santos portador(a) de la cédula de ciudadanía Nº **0151547916**. Declaro ser el autor de la obra: "**Desarrollo de un aplicativo web para la gestión de laboratorios, aulas de realidad virtual y otros espacios educativos**", sobre la cual me hago responsable sobre las opiniones, versiones e ideas expresadas. Declaro que la misma ha sido elaborada respetando los derechos de propiedad intelectual de terceros y eximo a la Universidad Católica de Cuenca sobre cualquier reclamación que pudiera existir al respecto. Declaro finalmente que mi obra ha sido realizada cumpliendo con todos los requisitos legales, éticos y bioéticos de investigación, que la misma no incumple con la normativa nacional e internacional en el área específica de investigación, sobre la que también me responsabilizo y eximo a la Universidad Católica de Cuenca de toda reclamación al respecto.

Cuenca, 23 de marzo de 2026

F: .....

Adrián Orlando Álvarez Dos Santos
C.I. **0151547916**

Universidad Católica de Cuenca	**CERTIFICADO DEL TUTOR**

Yo, Leopoldo Pauta Ayabaca, certifico que el presente trabajo de investigación / artículo científico, con el título "**Desarrollo de un aplicativo web para la gestión de laboratorios, aulas de realidad virtual y otros espacios educativos**" fue desarrollado por Adrián Orlando Álvarez Dos Santos, con número de cédula 0151547916 bajo mi supervisión.

F: .....

Leopoldo Pauta Ayabaca
C.I. 0101995843

Dedicatoria

A mis padres, por acompañarme en este camino que tomó su tiempo, pero que hoy finalmente culmina.

El proceso de formación profesional no siempre sigue los tiempos que uno planifica. Hubo momentos de pausa, de replantear prioridades, de enfrentar obstáculos que no estaban en el plan original. Sin embargo, a pesar de las demoras y los desvíos, hoy puedo decir que se logró.

Nada de esto hubiera sido posible sin su apoyo constante. Gracias por la paciencia cuando los plazos se extendían, por la confianza cuando las dudas aparecían, y por el respaldo silencioso pero firme que me permitió seguir adelante. Ustedes nunca dejaron de creer en que este momento llegaría, incluso cuando yo mismo dudaba.

Este título no es solo mío. Es el resultado de años de esfuerzo compartido, de sacrificios que hicieron para que yo pudiera estudiar, de madrugadas y desvelos que también fueron suyos. Cada paso que di en esta carrera lo di porque ustedes me dieron la base para hacerlo.

Hoy, al cerrar este capítulo académico, quiero que sepan que su inversión de tiempo, recursos y fe en mí no fue en vano. Lo logramos.

A ustedes, con profundo agradecimiento y respeto, les dedico este trabajo.

Agradecimientos

Al culminar este trabajo de titulación, quiero expresar mi agradecimiento a todas las personas e instituciones que hicieron posible llegar a esta instancia.

En primer lugar, a mi tutor, Ing. Leopoldo Pauta Ayabaca, por su guía y orientación durante el desarrollo de este proyecto. Su dirección académica, sus observaciones técnicas y su disposición para resolver dudas fueron fundamentales para la culminación exitosa de este trabajo. Agradezco su paciencia y profesionalismo a lo largo de todo el proceso. A la Universidad Católica de Cuenca, institución que me abrió las puertas para formarme profesionalmente. El respaldo institucional, los recursos académicos y las facilidades brindadas fueron determinantes para poder completar mi carrera. Más allá de los conocimientos técnicos adquiridos, la universidad me proporcionó un entorno de aprendizaje que contribuyó a mi crecimiento tanto profesional como personal. El apoyo institucional fue clave para poder culminar esta etapa, especialmente en los momentos donde el camino se hizo más complejo.

A los docentes de la carrera de Ingeniería de Software, quienes a lo largo de estos años compartieron sus conocimientos y experiencias. Cada clase, cada proyecto y cada evaluación contribuyeron a construir las competencias que hoy me permiten presentar este trabajo. A la Jefatura de Innovación y Emprendimiento de la Universidad Católica de Cuenca, por facilitar el acceso a las aulas de realidad virtual y proporcionar la información necesaria para el desarrollo de este sistema. Su colaboración fue esencial para entender las necesidades reales del proyecto y poder diseñar una solución que responda a los requerimientos institucionales.

A mis padres, por el respaldo incondicional durante todos estos años de formación universitaria. El camino hacia la titulación no fue lineal ni breve, pero su apoyo constante me permitió seguir adelante hasta llegar a este punto. Gracias por la paciencia, por la comprensión y por nunca dejar de confiar en que este momento llegaría.

Finalmente, agradezco a todas aquellas personas que de una u otra forma contribuyeron a este logro. Cada palabra de aliento, cada consejo y cada gesto de apoyo tuvo su impacto en este resultado.

A todos, mi más sincero agradecimiento.

Resumen

Se presenta el desarrollo de un sistema web integral para la gestión y agendamiento de aulas de realidad virtual en la Universidad Católica de Cuenca. El objetivo principal fue diseñar e implementar una solución tecnológica que optimice la reserva de espacios académicos especializados, mejorando la experiencia de docentes y administradores mediante una interfaz moderna, intuitiva y funcional.

La arquitectura del sistema se basa en un modelo cliente-servidor. El backend fue desarrollado en Python utilizando el framework Flask, implementando una API RESTful compuesta por 67 endpoints organizados en 10 módulos funcionales. La base de datos relacional PostgreSQL gestiona información de usuarios, aulas, reservas, capacitaciones y contenido dinámico.

El frontend, desarrollado en Flutter, ofrece una interfaz web responsiva y multiplataforma. El sistema incluye autenticación mediante JSON Web Tokens (JWT) con tres roles definidos: docentes, administradores y superadministradores. Además, se incorporó un módulo de capacitaciones que regula la habilitación de docentes para el uso de las aulas VR, asegurando que solo usuarios capacitados puedan realizar reservas.

Se integró también un servicio de procesamiento de imágenes para la gestión de contenido multimedia. Las pruebas funcionales confirmaron que el sistema cubre todo el ciclo de agendamiento: registro de usuarios, solicitud de capacitaciones, y gestión de reservas.

Finalmente, su arquitectura modular facilita el mantenimiento y la escalabilidad, mientras que el uso de servicios cloud garantiza disponibilidad y accesibilidad, contribuyendo a la transformación digital universitaria.

Palabras clave: *Sistema de agendamiento, realidad virtual, Flask, Flutter, API RESTful*.

Abstract

This paper presents the development of a comprehensive web-based system for managing and scheduling virtual reality classrooms at the Catholic University of Cuenca. The primary objective was to design and implement a technological solution that optimizes the booking of specialized academic spaces, enhancing the experience of faculty members and administrators through a modern, intuitive, and functional interface.

The system architecture is based on a client-server model. The backend was developed in Python using the Flask framework, implementing a RESTful API consisting of 67 endpoints organized into 10 functional modules. The PostgreSQL relational database manages information on users, classrooms, reservations, training sessions, and dynamic content.

The frontend, developed using Flutter, offers a responsive and cross-platform interface. The system includes authentication via JSON Web Tokens (JWT) with three defined roles: faculty members, administrators, and superadministrators. In addition, a training module has been incorporated to control faculty authorization for the use of VR classrooms, ensuring that only trained users can make reservations.

An image processing service was also integrated for multimedia content management. Functional testing confirmed that the system covers the entire scheduling cycle: user registration, training requests, and reservation management.

Finally, its modular architecture facilitates maintenance and scalability, while the use of cloud services ensures availability and accessibility, contributing to the university digital transformation.

Keywords: *Scheduling system, virtual reality, Flask, Flutter, RESTful API.*

Tabla de contenido

Declaratoria de autoría y responsabilidad……ii

Certificado del tutor ……iii

Dedicatoria …… iv

Agradecimientos ……...v

Resumen …… vi

Abstract …… vii

Tabla de contenido …….. viii

Lista de figuras …….. xii

Lista de tablas …….. xiv

CAPÍTULO 1 1

INTRODUCCIÓN 1

1.1. Planteamiento del problema 1

1.2. Justificación de la investigación 2

1.3. Objetivos de la investigación 2

1.3.1 Objetivo general 2

1.3.2 Objetivos específicos 3

1.4. Alcance del estudio 3

CAPÍTULO 2 5

MARCO TEÓRICO 5

2.1. Sistemas de gestión de recursos académicos 5

2.1.1 Definición y características 5

2.1.2 Importancia en instituciones educativas 5

2.2. Tecnologías de desarrollo web 6

2.2.1 Arquitectura cliente-servidor 6

2.2.2 APIs RESTful 7

2.2.3 Autenticación mediante JWT 7

2.3. Framework Flask para desarrollo backend 8

2.3.1 Características principales 8

2.3.2 Extensiones utilizadas 8

2.3.3 SQLAlchemy como ORM 9

2.4. Framework Flutter para desarrollo frontend 9

2.4.1 Flutter Web 11
2.5. Bases de datos relacionales 12
2.5.1 PostgreSQL 12
2.5.2 Diseño de esquemas 12
2.6. Servicios cloud para despliegue 13
2.6.1 Render 13
2.6.2 Neon PostgreSQL 13
2.6.3 Firebase 13
2.7. Metodologías ágiles de desarrollo 13
2.7.1 SCRUM 13
CAPÍTULO 3 15
METODOLOGÍA 15
3.1. Tipo de investigación 15
3.2. Metodología de desarrollo 15
3.3. Análisis de requerimientos 16
3.4. Diseño de la arquitectura 17
3.5. Diseño de la base de datos 18
3.6. Desarrollo del backend 18
3.7. Desarrollo del frontend 21
3.8. Integración y pruebas 22
3.9. Despliegue en producción 22
CAPÍTULO 4 24
DESARROLLO E IMPLEMENTACIÓN 24
4.1. Configuración del entorno de desarrollo 24
4.2. Implementación del backend 25
4.2.1 Estructura del proyecto 25
4.2.2 Modelos de datos 26
4.2.3 Sistema de autenticación 28
4.2.4 Decoradores de autorización 28
4.2.5 Endpoints implementados 28
4.2.6. Servicio de procesamiento de imágenes 34
4.3. Implementación del frontend 35
4.3.1 Estructura del proyecto Flutter 35

4.3.2 Modelos de datos 35
4.3.3 Servicios de API 39
4.3.4 Gestión de estado 41
4.3.5 Pantallas principales 43
4.4. Integración de servicios 49
4.4.1 Configuración de CORS 49
4.4.2 Flujo de comunicación 49
4.5. Configuración del despliegue 50
4.5.1 Configuración de Neon PostgreSQL 50
4.5.2 Configuración de Render 51
4.5.3 Configuración de Firebas 52
CAPÍTULO 5 55
RESULTADOS 55
5.1. Pruebas del sistema 55
5.1.1 Pruebas de endpoints (Backend) 55
5.1.2 Pruebas funcionales (frontend) 55
5.2. Capturas de pantalla del sistema 56
5.2.1 Landing page 56
5.2.2 Módulo de autenticación 58
5.2.3 Panel de docentes 59
5.2.4 Panel de administración 62
5.2.5 Módulo de capacitaciones 65
5.3. Métricas de rendimiento 67
CAPÍTULO 6 68
CONCLUSIONES Y RECOMENDACIONES 68
6.1. Conclusiones 68
6.2. Recomendaciones 69
6.3. Trabajos futuros 69
REFERENCIAS 70
ANEXOS 74
7.1. Modelos de datos del sistema 74
7.2. Modelo de Aulas 75
7.3. Modelo de Agendamientos 75

7.4. Sistema de autenticación y autorización 76
7.5. Funciones utilitarias 77
7.6. Servicio de procesamiento de imágenes..... 78
7.7. Configuración de la aplicación Flask 79
7.8. Configuración de la aplicación Flutter 80
7.9. Dependencias del proyecto..... 81
7.10. Autorización de publicación en el repositorio institucional 82

Lista de figuras

Fig. 1: Arquitectura general del sistema de agendamiento ITE VR 17
Fig. 2: Estructura de directorios del backend 18
Fig. 3: Diagrama Entidad-Relación de la base de datos 19
Fig. 4. Diagrama de flujo del proceso de agendamiento 20
Fig. 5: Dependencias del proyecto (pubspec.yaml) 25
Fig. 6: Modelos de SQLAlchemy 26
Fig. 7: Estructura del proyecto 27
Fig. 8: Modelo aula 29
Fig. 9: Funciones utilitarias de autenticación (app/utils.py) 30
Fig. 10: Decoradores para implementar el control de acceso basado en roles 31
Fig. 11: Endpoint de creación de agenda 32
Fig. 12: Endpoint de disponibilidad de aula 33
Fig. 13: Servicio de imágenes 34
Fig. 14: Estructura de directorios del proyecto Flutter 35
Fig. 13: Modelo de Usuario 36
Fig. 14: Modelo de Aula 37
Fig. 15: Modelo Booking 38
Fig. 16: Servicios de API 39
Fig. 17: API Client 40
Fig. 18: Theme Provider 41
Fig. 20: Configuracion del Provider 43
Fig. 21: Landing Page - Hero section con carrusel de imágenes 43
Fig. 22: Landing Page 44
Fig. 24: Login Page 45
Fig. 27: Dashboard Layout 47
Fig. 28: App Card 48
Fig. 29: App Modal 48
Fig. 30: CORS 49
Fig. 31: Diagrama de Comunicacion 49
Fig. 33: Configuración del despliegue en Render 51
Fig. 34: Configuracion de Firebase por CMD 52
Fig. 35: Configuracion Firebase 53
Fig. 36: Configuracion .env 53

Fig. 37: Landing page Carrusel 56
Fig. 38: Landing page Info 1 56
Fig. 39: Landing page Info 2 57
Fig. 40: Landing page Equipos 57
Fig. 41: Landing page Info 3 57
Fig. 42: Login Page 58
Fig. 43: Agendamiento de Aulas Docente 1 59
Fig. 45: Agendamiento de Aulas Docente 3 59
Fig. 46: Agendamiento de Aulas Docente 4 59
Fig. 47: Agendamiento de Aulas Docente 5 60
Fig. 48: Agendamiento de Aulas Docente 6 60
Fig. 49: Agendas Docente 60
Fig. 50: Datos Personales Docente 61
Fig. 51: Capacitaciones Docente 61
Fig. 52: Agendas Administrador 62
Fig. 53: Editar Agendas Administrador 62
Fig. 54: Datos Personales Administrador 62
Fig. 55: Docentes Capacitados Administrador 63
Fig. 56: Administradores Administrador 63
Fig. 57: Capacitaciones Administrador 63
Fig. 58: Capacitaciones Agendadas Administrador 63
Fig. 59: Capacitaciones Finalizadas Administrador 64
Fig. 60: Aulas Administrador 64
Fig. 61: Crear Aulas Administrador 64
Fig. 62: Landing Page Carrusel Editor Administrador 64
Fig. 63: Landing Page Slide Editor Administrador 65
Fig. 64: Módulo de Capacitaciones 65
Fig. 65: Modal Creación Docente 66
Fig. 66: Notificación del estado del Docente 66
Fig. 67: Módulo de Capacitaciones, Sección de Agendas 66
Fig. 68: Módulo de Capacitaciones, Agendar Capacitacion 66
Fig. 69: Módulo de Capacitaciones, Agendadas 67
Fig. 70: Módulo de Capacitaciones, Finalizadas 67

Lista de tablas

Tabla 1: Comparación de frameworks backend 9
Tabla 2: Comparación de frameworks frontend. 12
Tabla 3: Requerimientos funcionales 17
Tabla 4: Requerimientos no funcionales 17
Tabla 5: Herramientas utilizadas en el frontend, versión y propósito 24
Tabla 6: Estructura de la tabla usuarios 28
Tabla 7: Resumen de endpoints por módulo 28
Tabla 8: Resumen de modelos de datos del fronten 39
Tabla 9: Configuracion de Firebase 54
Tabla 10: Resultados de pruebas de endpoints principales 55
Tabla 11: Resultados de pruebas funcionales del frontend 56
Tabla 12: Métricas de rendimiento del frontend 67

CAPÍTULO 1

INTRODUCCIÓN

1.1. Planteamiento del problema

La integración de tecnologías inmersivas en el ámbito educativo ha experimentado un crecimiento significativo en las últimas décadas, posicionando a la realidad virtual como una herramienta pedagógica de alto impacto para la formación de profesionales en diversas áreas del conocimiento. La Jefatura de Innovación y Emprendimiento de la Universidad Católica de Cuenca, consciente de esta tendencia, ha invertido en la implementación de aulas equipadas con dispositivos de realidad virtual para enriquecer la experiencia de aprendizaje de sus estudiantes [1].

Actualmente, la institución cuenta con diversas aulas de realidad virtual equipados con dispositivos Meta Quest y otros equipos especializados, con capacidad para atender aproximadamente hasta 15 estudiantes por sesión. Se estima que cerca de 40 docentes de diversas facultades podrían requerir acceso a estos recursos para actividades académicas relacionadas con sus áreas de especialización.

Sin embargo, la gestión de estos recursos especializados presenta desafíos operativos significativos que afectan tanto la eficiencia administrativa como la experiencia de los usuarios finales.

El proceso actual de agendamiento de las aulas de realidad virtual se realiza de manera manual o semiautomatizada, lo que genera inconvenientes como: duplicidad de reservas, falta de visibilidad de la disponibilidad en tiempo real, dificultad para coordinar las capacitaciones requeridas previas al uso de los equipos, y ausencia de un registro histórico que permita analizar patrones de uso y optimizar la asignación de recursos [2].

La ausencia de un sistema digital integrado para la gestión de estos espacios afecta directamente la eficiencia administrativa y la experiencia de los docentes que requieren utilizar estas aulas para sus actividades académicas. Los procesos manuales consumen tiempo tanto del personal administrativo como de los docentes, quienes deben desplazarse físicamente o realizar múltiples comunicaciones para verificar disponibilidad y concretar una reserva [3].

Adicionalmente, el requisito de capacitación previa para el uso de equipos de realidad virtual añade una capa de complejidad al proceso. Los docentes deben demostrar que han recibido la formación necesaria antes de poder agendar el uso de las aulas, lo cual actualmente se gestiona mediante registros separados que dificultan la verificación automatizada de este requisito [4].

En este contexto surge la pregunta de investigación: ¿Cómo desarrollar un sistema web que permita gestionar de manera eficiente el agendamiento de aulas de realidad virtual, integrando el control de capacitaciones y proporcionando una experiencia de usuario moderna y accesible?

1.2. Justificación de la investigación

La transformación digital de los procesos administrativos en instituciones de educación superior representa una necesidad imperativa en el contexto actual. Los sistemas de gestión de recursos académicos no solo mejoran la eficiencia operativa, sino que también contribuyen a la satisfacción de la comunidad universitaria y a la optimización del uso de infraestructura especializada [5].

El desarrollo de un sistema web para el agendamiento de aulas de realidad virtual se justifica desde múltiples perspectivas:

Perspectiva tecnológica: La implementación de una arquitectura moderna basada en APIs RESTful, con un backend en Flask y un frontend en Flutter, permite crear una solución escalable, mantenible y adaptable a futuros requerimientos. La separación de responsabilidades entre cliente y servidor facilita el desarrollo independiente de cada componente y posibilita la futura expansión hacia aplicaciones móviles nativas [6].

Perspectiva operativa: La automatización del proceso de agendamiento elimina la duplicidad de reservas, reduce el tiempo dedicado a tareas administrativas y proporciona visibilidad en tiempo real de la disponibilidad de recursos. El módulo de capacitaciones integrado garantiza que solo usuarios habilitados puedan realizar reservas, cumpliendo con los protocolos de seguridad establecidos para el uso de equipos de realidad virtual [7].

Perspectiva académica: El sistema facilita la planificación de actividades académicas que requieren el uso de aulas especializadas, permitiendo a los docentes concentrarse en aspectos pedagógicos en lugar de procesos burocráticos. La disponibilidad de un calendario visual y la posibilidad de gestionar reservas desde cualquier dispositivo con acceso a internet mejora significativamente la experiencia del usuario [8].

Perspectiva institucional: La implementación de este sistema contribuye a los objetivos de modernización tecnológica de la Universidad Católica de Cuenca, alineándose con las tendencias de transformación digital en el sector educativo ecuatoriano. Además, el registro sistemático de datos de uso permite generar reportes y estadísticas que apoyan la toma de decisiones sobre inversión en infraestructura [9].

El proyecto también representa una oportunidad de aplicar conocimientos adquiridos durante la formación académica en el desarrollo de una solución real que beneficia directamente a la comunidad universitaria, demostrando las competencias profesionales requeridas para un Ingeniero en Software [10].

1.3. Objetivos de la investigación

1.3.1 Objetivo general

Desarrollar un sistema web para la gestión y agendamiento de aulas de realidad virtual en la Universidad Católica de Cuenca, utilizando Flutter como framework de frontend y Flask como framework de backend, que permita optimizar el proceso de reserva de espacios académicos especializados.

1.3.2 Objetivos específicos

1) Analizar los requerimientos funcionales y no funcionales del sistema mediante el levantamiento de información con los stakeholders involucrados en el proceso de agendamiento de aulas VR.

2) Diseñar la arquitectura del sistema aplicando el patrón cliente-servidor con una API RESTful, definiendo los modelos de datos, endpoints y flujos de información necesarios.

3) Implementar el backend del sistema utilizando Python con el framework Flask, desarrollando los módulos de autenticación, gestión de usuarios, aulas, agendamientos, capacitaciones y contenido dinámico.

4) Implementar el frontend del sistema utilizando Flutter Web, desarrollando las interfaces de usuario para la landing page, autenticación, panel de docentes y panel de administración.

5) Integrar los servicios de base de datos PostgreSQL, almacenamiento de imágenes en Firebase Storage y desplegar la aplicación en servicios cloud (Render, Neon, Firebase Hosting).

6) Validar el funcionamiento del sistema mediante pruebas funcionales que verifiquen el cumplimiento de los requerimientos establecidos.

1.4. Alcance del estudio

El presente trabajo de titulación establece los siguientes parámetros y límites: Alcance funcional:

- Sistema de registro y autenticación de usuarios con roles diferenciados (docente, administrador, super administrador).
- Módulo de gestión de capacitaciones para habilitar docentes en el uso de aulas VR.
- Módulo de agendamiento con calendario visual, verificación de disponibilidad y gestión de reservas.
- Panel de administración para gestión de usuarios, aulas y contenido de la landing page.
- Landing page pública con información del sistema y acceso a registro/login.
- Servicio de carga y procesamiento de imágenes para aulas y contenido multimedia.

Alcance tecnológico:

- Backend: Python 3.x, Flask 3.1.1, SQLAlchemy 2.0.41, Flask-Migrate 4.0.5
- Frontend: Flutter/Dart para web

- Base de datos: PostgreSQL (Neon)
- Almacenamiento: Firebase Storage
- Despliegue: Render (backend), Firebase Hosting (frontend)
- Autenticación: JWT con bcrypt para hash de contraseñas

Exclusiones:

- Aplicaciones móviles nativas (aunque la arquitectura las permite en el futuro)
- Integración con sistemas académicos externos (SGA, plataformas LMS)
- Módulo de facturación o pagos
- Notificaciones push o por correo electrónico (implementación básica incluida, pero no servicio de envío)
- Analítica avanzada con machine learning

Limitaciones:

- El sistema está diseñado para la gestión de aulas de realidad virtual específicamente, aunque la arquitectura permite adaptación a otros tipos de recursos.
- El despliegue en servicios gratuitos puede tener limitaciones de rendimiento y disponibilidad según los términos de servicio de cada proveedor.
- Las pruebas se realizan en un entorno controlado con datos de prueba, requiriendo validación adicional en producción con usuarios reales.

CAPÍTULO 2

MARCO TEÓRICO

2.1. Sistemas de gestión de recursos académicos

2.1.1 Definición y características

Los sistemas de gestión de recursos académicos son plataformas tecnológicas diseñadas para administrar, organizar y optimizar el uso de los diferentes recursos disponibles en instituciones educativas. Estos sistemas abarcan desde la gestión de espacios físicos como aulas y laboratorios, hasta la administración de equipos tecnológicos, materiales didácticos y recursos humanos [11].

Las características fundamentales de estos sistemas incluyen:

- Centralización de información: Todos los datos relacionados con los recursos se almacenan en una única plataforma, eliminando la dispersión de información y facilitando la consulta y actualización [12].
- Automatización de procesos: Las tareas repetitivas como verificación de disponibilidad, generación de reportes y notificaciones se ejecutan de manera automática, reduciendo la carga administrativa [13].
- Accesibilidad: Los usuarios autorizados pueden acceder al sistema desde diferentes dispositivos y ubicaciones, facilitando la gestión remota [14].
- Trazabilidad: El registro de todas las operaciones permite auditar el uso de recursos y analizar patrones históricos [15].

2.1.2 Importancia en instituciones educativas

La implementación de sistemas de gestión de recursos en el contexto educativo responde a la necesidad de optimizar el uso de infraestructura que frecuentemente representa inversiones significativas. En el caso específico de aulas de realidad virtual, donde los equipos tienen costos elevados y requieren condiciones especiales de uso, la gestión eficiente se vuelve crítica [16].

Estudios recientes demuestran que las instituciones que implementan sistemas digitales de gestión de recursos experimentan mejoras significativas en la tasa de utilización de infraestructura, reducción de conflictos de programación y mayor satisfacción de usuarios [17].

2.1.3 Trabajos relacionados y antecedentes

En el ámbito de sistemas de gestión de recursos académicos, diversas instituciones han implementado soluciones tecnológicas para optimizar la administración de espacios especializados. A continuation, se presentan algunos antecedentes relevantes:

- Sistemas de reserva de laboratorios universitarios: Universidades como la Universidad Politécnica de Madrid y la Universidad de Sao Paulo han

desarrollado sistemas web propios para la gestión de laboratorios de cómputo y espacios especializados. Estos sistemas típicamente incluyen autentificación institucional, calendarios de disponibilidad y notificaciones por correo electrónico. Sin embargo, la mayoría carece de módulos específicos para gestión de capacitaciones previas al uso de equipos relacionados [16].

- Plataformas comerciales de gestión de espacios: Soluciones como EM5 Campus (por Accruent) y Archibus ofrecen funcionalidades completas para la gestión de espacios en instituciones educativas. No obstante, estas plataformas presentan costos de licenciamiento elevados y requieren personalización significativa para adaptarse a flujos de trabajo específico como el control de capacitaciones para equipos VR [17].

- Soluciones basadas en herramientas genéricas: Muchas Instituciones recurren a combinaciones de Google Calendar, Microsoft Bookings o formularios en línea para gestionar reservas de espacios. Si bien estas soluciones son económicas, carecen de validación automática de requisitos previos (como capacitaciones), generan fragmentación de la información y no proporcionan visibilidad centralizada del uso de recursos [18].

El presente trabajo se diferencia de las soluciones existentes al integrará en una única plataforma: el proceso de capacitación como requisito previo para el agendamiento, la gestión de roles diferenciados (docente, administrador, super administrador), y una interfaz moderna desarrollada con tecnologías actuales (Flutter y Flask) que facilita tanto el uso como el mantenimiento del sistema.

2.2. Tecnologías de desarrollo web

2.2.1 Arquitectura cliente-servidor

La arquitectura cliente-servidor es un modelo de diseño de software que separa las funciones del sistema en dos componentes principales: el cliente, que representa la interfaz de usuario y realiza solicitudes, y el servidor, que procesa estas solicitudes y gestiona los datos [18].

En el contexto del presente proyecto, esta arquitectura se implementa mediante:

- Cliente (Frontend): Aplicación web desarrollada en Flutter que se ejecuta en el navegador del usuario, responsable de la presentación visual y la interacción.

- Servidor (Backend): API RESTful desarrollada en Flask que procesa las solicitudes del cliente, aplica la lógica de negocio y gestiona la persistencia de datos [19].

Las ventajas de esta arquitectura incluyen:

1. Separación de responsabilidades: Cada componente tiene funciones claramente definidas, facilitando el desarrollo y mantenimiento independiente.

2. Escalabilidad: El servidor puede atender múltiples clientes simultáneamente y puede escalarse horizontalmente según la demanda.

3. Reutilización: La API puede ser consumida por diferentes tipos de clientes (web, móvil, desktop) sin modificaciones [20].

2.2.2 APIs RESTful

REST (Representational State Transfer) es un estilo arquitectónico para el diseño de servicios web que define un conjunto de restricciones y principios para la comunicación entre sistemas. Una API RESTful es aquella que cumple con estos principios [21]. Los principios fundamentales de REST incluyen:

- Interfaz uniforme: Uso consistente de métodos HTTP (GET, POST, PUT, DELETE) para las operaciones CRUD.

- Sin estado (Stateless): Cada solicitud contiene toda la información necesaria para ser procesada, sin depender de contexto almacenado en el servidor.

- Recursos identificables: Cada recurso tiene una URI única que lo identifica.

- Representaciones: Los recursos pueden tener múltiples representaciones (JSON, XML) [22].

En el sistema desarrollado, los endpoints siguen convenciones REST estándar:

GET	/api/aulas	→ Listar aulas
GET	/api/aulas/{id}	→ Obtener aula específica
POST	/api/aulas	→ Crear nueva aula
PUT	/api/aulas/{id}	→ Actualizar aula
DELETE	/api/aulas/{id}	→ Eliminar aula

2.2.3 Autenticación mediante JWT

JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define un formato compacto y autocontenido para transmitir información de manera segura entre partes como un objeto JSON. Esta información puede ser verificada y confiable porque está firmada digitalmente [23]. Un JWT consta de tres partes separadas por puntos:

1. Header: Contiene el tipo de token y el algoritmo de firma utilizado.

2. Payload: Contiene las claims (declaraciones) sobre el usuario y metadatos adicionales.

3. Signature: Firma digital que verifica la integridad del token [24].

El flujo de autenticación implementado en el sistema sigue estos pasos:

1. El usuario envía sus credenciales (email y contraseña) al endpoint de login.
2. El servidor verifica las credenciales contra la base de datos.
3. Si son válidas, genera un JWT con la información del usuario y un tiempo de expiración.
4. El cliente almacena el token y lo incluye en el header Authorization de las solicitudes subsecuentes.
5. El servidor verifica el token en cada solicitud protegida [25].

2.3. Framework Flask para desarrollo backend

2.3.1 Características principales

Flask es un microframework de Python para el desarrollo de aplicaciones web. Se denomina "micro" porque mantiene un núcleo simple pero extensible, permitiendo a los desarrolladores añadir las funcionalidades que necesitan mediante extensiones [26].

Las características que motivaron su selección para este proyecto incluyen:

- Simplicidad: Sintaxis clara y curva de aprendizaje moderada.
- Flexibilidad: No impone una estructura de proyecto específica.
- Extensibilidad: Amplio ecosistema de extensiones para funcionalidades adicionales.
- Documentación: Documentación oficial completa y comunidad activa.
- Rendimiento: Adecuado para aplicaciones de tamaño pequeño a mediano [27].

2.3.2 Extensiones utilizadas

El proyecto utiliza las siguientes extensiones de Flask: Flask-SQLAlchemy (v3.1.1): Integración de SQLAlchemy con Flask, proporcionando un ORM (Object-Relational Mapping) que permite interactuar con la base de datos utilizando objetos Python en lugar de SQL directo [28].

Flask-Migrate (v4.0.5): Gestión de migraciones de base de datos utilizando Alembic, permitiendo versionar y aplicar cambios al esquema de manera controlada [29].

Flask-Bcrypt (v1.0.1): Implementación de funciones de hashing bcrypt para el almacenamiento seguro de contraseñas [30].

Flask-CORS (v6.0.0): Habilitación de Cross-Origin Resource Sharing, necesario para que el frontend pueda comunicarse con el backend desde diferentes dominios [31].

2.3.3 SQLAlchemy como ORM

SQLAlchemy es una biblioteca de Python que proporciona un conjunto completo de herramientas para trabajar con bases de datos relacionales. Su componente ORM permite mapear clases de Python a tablas de la base de datos, abstrayendo las operaciones SQL [32].

Beneficios del uso de SQLAlchemy:

- Abstracción de base de datos: El código puede funcionar con diferentes motores de base de datos con cambios mínimos.
- Seguridad: Prevención automática de inyección SQL.
- Productividad: Reducción del código boilerplate para operaciones CRUD.
- Relaciones: Manejo declarativo de relaciones entre tablas [33].

Característica	Flask	Django	FastAPI	Express.js
Lenguaje	Python	Python	Python	JavaScript
Tipo	Microframework	Full-stack	Microframework	Microframe-work
Curva de aprendizaje	Baja	Media-Alta	Media	Baja
Rendimiento	Bueno	Bueno	Excelente	Muy bueno
ORM incluido	No (SQLAl-chemy)	Si (Django ORM)	No (SQLAlchemy)	No
Documentación au-tomática	No	No	Si (OpenAPI)	No
Comunidad	Grande	Muy grande	Creciente	Muy grande
Flexibilidad	Alta	Media	Alta	Alta
Selección	**Si**	No	No	No

Tabla 1: Comparación de frameworks backend

2.4. Framework Flutter para desarrollo frontend

2.4.1. Arquitectura y características de Flutter

Flutter es un framework de código abierto desarrollado por Google para la creación de aplicaciones multiplataforma compiladas de forma nativa a partir de una única base de código. Lanzado oficialmente en diciembre de 2018, Flutter ha experimentado un crecimiento exponencial en su adopción, posicionándose como una de las tecnologías más populares para el desarrollo de interfaces de usuario modernas [48].

La arquitectura de Flutter se fundamenta en tres capas principales que trabajan de manera coordinada para renderizar interfaces de usuario de alto rendimiento. En la capa superior

se encuentra el Framework, escrito completamente en Dart, que proporciona un rico conjunto de bibliotecas para la construcción de interfaces. Esta capa incluye los widgets de Material Design y Cupertino, así como las herramientas de animación, gestos y renderizado. La capa intermedia corresponde al Engine, desarrollado principalmente en C++, que implementa las primitivas de bajo nivel necesarias para el renderizado gráfico utilizando Skia, el manejo de texto, la red y los plugins de plataforma. Finalmente, la capa de Embedder permite la integración con las plataformas específicas como Android, iOS, Web y Desktop [49].

El concepto central de Flutter radica en su sistema de widgets, donde "todo es un widget". Los widgets son descripciones inmutables de parte de la interfaz de usuario que se componen jerárquicamente para formar el árbol de widgets (Widget Tree). Cuando el estado de un widget cambia, Flutter reconstruye eficientemente solo las partes afectadas mediante un proceso de reconciliación que compara el árbol anterior con el nuevo, aplicando únicamente las diferencias necesarias al árbol de elementos (Element Tree) y posteriormente al árbol de renderizado (Render Tree) [50].

Una de las características más distintivas de Flutter es el Hot Reload, que permite a los desarrolladores ver los cambios en el código reflejados instantáneamente en la aplicación en ejecución sin perder el estado actual. Esta funcionalidad acelera significativamente el ciclo de desarrollo, permitiendo la experimentación rápida con diseños y la corrección inmediata de errores visuales. El Hot Restart, por su parte, reinicia completamente la aplicación cuando los cambios afectan el estado global [48].

La justificación para seleccionar Flutter en el presente proyecto se fundamenta en múltiples factores técnicos y estratégicos. En primer lugar, la capacidad de desarrollar una aplicación web responsiva y moderna con un único código base reduce significativamente el tiempo y los recursos necesarios para el desarrollo.

En segundo lugar, el rendimiento cercano al nativo que ofrece Flutter, incluso en su versión web, garantiza una experiencia de usuario fluida. En tercer lugar, el amplio ecosistema de paquetes disponibles en pub.dev facilita la integración de funcionalidades complejas como mapas, calendarios y procesamiento de imágenes. Finalmente, la documentación exhaustiva y la comunidad activa proporcionan soporte constante durante el desarrollo [51].

Para la gestión del estado de la aplicación se implementó el patrón Provider, un enfoque recomendado oficialmente por el equipo de Flutter para aplicaciones de complejidad media. Provider permite la inyección de dependencias y la propagación de cambios de estado a través del árbol de widgets de manera eficiente y declarativa.

En el sistema desarrollado, se utilizan dos providers principales: ThemeProvider para gestionar el tema visual (claro/oscuro) y LocaleProvider para manejar la internacionalización (español/inglés). Ambos providers persisten las preferencias del usuario utilizando SharedPreferences, garantizando que la configuración se mantenga entre sesiones [52].

2.4.1 Flutter Web

Flutter Web representa la extensión del framework Flutter para ejecutar aplicaciones en navegadores web modernos. Introducido como versión estable en marzo de 2021, Flutter Web permite compilar aplicaciones Dart a JavaScript estándar que puede ejecutarse en cualquier navegador compatible [53].

Flutter Web ofrece dos modos de renderizado distintos que pueden seleccionarse según los requisitos del proyecto. El modo HTML utiliza elementos del DOM combinados con Canvas para renderizar la interfaz, resultando en archivos de menor tamaño y mejor compatibilidad con navegadores antiguos. El modo CanvasKit emplea WebAssembly y WebGL para renderizar la interfaz de manera idéntica a las versiones móviles, ofreciendo mayor fidelidad visual y mejor rendimiento en animaciones complejas, aunque con un tamaño de bundle inicial mayor [54].

Las consideraciones de rendimiento para Flutter Web incluyen la optimización del tamaño del bundle inicial, la carga diferida de recursos y la implementación de estrategias de caché efectivas. El tiempo de carga inicial puede verse afectado por el tamaño del motor CanvasKit (aproximadamente 2.5 MB comprimido), aunque este se almacena en caché del navegador para visitas subsecuentes. Para aplicaciones que priorizan el tiempo de carga sobre la fidelidad visual, el modo HTML ofrece una alternativa más ligera [53].

La compatibilidad con navegadores de Flutter Web abarca las versiones modernas de Chrome, Firefox, Safari y Edge. El framework implementa detección automática de capacidades del navegador para seleccionar el modo de renderizado más apropiado. Sin embargo, existen limitaciones conocidas relacionadas con el acceso a APIs nativas del sistema operativo, la manipulación directa del DOM y la integración con bibliotecas JavaScript externas, las cuales requieren el uso de interoperabilidad específica [54].

Para el sistema de agendamiento ITE VR, se seleccionó Flutter Web debido a su capacidad de proporcionar una experiencia de usuario consistente y moderna sin requerir instalación de software adicional por parte de los usuarios finales. La aplicación se compila utilizando el modo CanvasKit para garantizar la máxima fidelidad visual en el renderizado de elementos como el calendario semanal y las tarjetas de información de aulas [55].

Característica	**Flutter**	**React**	**Angular**	**Vue.js**
Lenguaje	Dart	JavaScript	TypeScript	JavaScript
Tipo	Framework UI	Biblioteca	Framework	Framework
Multiplataforma	Web, Mobile, Desktop	Web (React Native)	Web	Web
Rendimiento Web	Muy bueno	Excelente	Muy bueno	Muy bueno
Curva de aprendi-zaje	Media	Media	Alta	Baja
Hot Reload	Si	Si	No	Si

Ecosistema	Grande (pub.dev)	Muy grande (npm)	Grande (npm)	Grande (npm)
UI consistente	Si (Material/Cupertino)	Depende de librerias	Angular Material	Vuetify
Selección	**Si**	No	No	No

Tabla 2: Comparación de frameworks frontend.

2.5. Bases de datos relacionales

2.5.1 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto, reconocido por su robustez, extensibilidad y cumplimiento de estándares SQL. Es ampliamente utilizado en aplicaciones de producción debido a su confiabilidad y conjunto de características avanzadas [34].

Características relevantes para el proyecto:

- Tipos de datos JSON: Soporte nativo para almacenar y consultar datos JSON, utilizado para campos como días disponibles en el modelo Aula.
- Integridad referencial: Soporte completo para claves foráneas y restricciones.
- Extensibilidad: Capacidad de definir tipos de datos personalizados y funciones.
- Transacciones ACID: Garantía de atomicidad, consistencia, aislamiento y durabilidad [35].

2.5.2 Diseño de esquemas

El diseño del esquema de base de datos sigue principios de normalización para evitar redundancia y garantizar integridad de datos. El modelo relacional implementado incluye las siguientes entidades principales:

- Entidades de ubicación: sedes, facultades, carreras (jerarquía geográfica/académica)
- Entidades de usuarios: usuarios (unificada con roles)
- Entidades de recursos: aulas
- Entidades de operación: agendas, asistencias, capacitaciones
- Entidades de contenido: carousel_slides, features, about_section, contact_info
- Entidades auxiliares: uploads, notificaciones [36]

2.6. Servicios cloud para despliegue

2.6.1 Render

Render es una plataforma cloud que simplifica el despliegue de aplicaciones web, ofreciendo servicios de hosting para aplicaciones backend, bases de datos y sitios estáticos. Su capa gratuita permite desplegar aplicaciones Python/Flask con las siguientes características:

- Despliegue automático desde repositorios Git
- SSL/HTTPS automático
- Escalado automático
- Variables de entorno seguras [37]

2.6.2 Neon PostgreSQL

Neon es un servicio de PostgreSQL serverless que ofrece una capa gratuita generosa para desarrollo y proyectos pequeños. Características relevantes:

- PostgreSQL compatible al 100%
- Branching de bases de datos para desarrollo
- Escalado automático
- Conexión segura SSL [38]

2.6.3 Firebase

Firebase es una plataforma de desarrollo de Google que proporciona múltiples servicios. En este proyecto se utilizan:

- Firebase Hosting: Alojamiento de aplicaciones web estáticas con CDN global.
- Firebase Storage: Almacenamiento de archivos (imágenes) con reglas de seguridad configurables [39].

2.7. Metodologías ágiles de desarrollo

2.7.1 SCRUM

SCRUM es un marco de trabajo ágil para la gestión y desarrollo de proyectos de software. Se basa en ciclos iterativos llamados sprints, que típicamente duran entre 1 y 4 semanas [40]. Elementos de SCRUM aplicados en el proyecto:

- Product Backlog: Lista priorizada de funcionalidades a desarrollar.
- Sprint Planning: Planificación de las tareas a completar en cada iteración.
- Daily Standups: Reuniones breves de seguimiento (adaptadas al contexto del proyecto).

- Sprint Review: Demostración de funcionalidades completadas.
- Sprint Retrospective: Análisis de mejoras para siguientes iteraciones [41].

La aplicación de SCRUM permitió:

1. Entregas incrementales de funcionalidad
2. Adaptación a cambios en los requerimientos
3. Visibilidad del progreso del proyecto
4. Mejora continua del proceso de desarrollo [42]

CAPÍTULO 3

METODOLOGÍA

3.1. Tipo de investigación

El presente trabajo se enmarca en una investigación de tipo aplicada con enfoque tecnológico, orientada al desarrollo de una solución de software que resuelve una problemática específica identificada en el contexto universitario. Se combina la investigación documental para la fundamentación teórica con el desarrollo experimental para la implementación del sistema [43].

La metodología empleada integra:

- Investigación documental: Revisión de literatura sobre sistemas de gestión de recursos, tecnologías de desarrollo web, y mejores prácticas de arquitectura de software.
- Investigación de campo: Levantamiento de requerimientos mediante entrevistas con stakeholders (administrativos, docentes, personal técnico de las aulas VR).
- Desarrollo experimental: Implementación iterativa del sistema con ciclos de desarrollo, prueba y refinamiento [44].

3.2. Metodología de desarrollo

Se adoptó la metodología SCRUM adaptada al contexto de un proyecto de titulación individual. Los sprints se definieron con duración de dos semanas, organizando el desarrollo en las siguientes fases:

Sprint 1-2: Análisis y Diseño

- Levantamiento de requerimientos
- Diseño de arquitectura
- Diseño de base de datos
- Configuración del entorno de desarrollo

Sprint 3-4: Backend Core

- Implementación de modelos
- Sistema de autenticación
- Endpoints de usuarios y aulas

Sprint 5-6: Backend Completo

- Endpoints de agendamientos

- Módulo de capacitaciones
- Landing page API
- Servicio de imágenes

Sprint 7-8: Frontend

- Configuración del proyecto Flutter Web y arquitectura por características
- Implementación de Landing Page con carrusel y secciones dinámicas
- Sistema de temas (claro/oscuro) e internacionalización (ES/EN)
- Desarrollo del módulo de autenticación y servicios de API
- Panel de docentes: selector de aulas, calendario semanal, gestión de reservas
- Panel de administración: CRUD de aulas, gestión de capacitaciones
- Editor de contenido de landing page para administradores
- Widgets reutilizables y componentes de UI

Sprint 9-10: Integración y Despliegue

- Integración frontend-backend
- Configuración de servicios cloud
- Pruebas de integración
- Despliegue en producción [45]

3.3. Análisis de requerimientos

A continuación, se presentan los requerimientos funcionales y no funcionales.

ID	Requerimiento	Prioridad
RF01	El sistema debe permitir el registro de usuarios con correo institucional	Alta
RF02	El sistema debe autenticar usuarios mediante email y contraseña	Alta
RF03	El sistema debe diferenciar roles: docente, admin, superAdmin	Alta
RF04	Los docentes deben poder solicitar capacitación	Alta
RF05	Los administradores deben poder aprobar/rechazar capacitaciones	Alta
RF06	Solo usuarios capacitados pueden crear agendamientos	Alta
RF07	El sistema debe mostrar disponibilidad de aulas en tiempo real	Alta
RF08	Los usuarios deben poder crear, modificar y cancelar reservas	Alta
RF09	El sistema debe validar conflictos de horario	Alta
RF10	Los administradores deben poder gestionar aulas	Media

RF11	El sistema debe permitir subir imágenes de aulas	Media
RF12	El sistema debe mostrar un calendario visual de reservas	Media
RF13	La landing page debe ser configurable dinámicamente	Baja
RF14	El sistema debe generar estadísticas de uso	Baja

Tabla 3: Requerimientos funcionales

ID	Requerimiento	Criterio
RNF01	Tiempo de respuesta	< 2 segundos para operaciones comunes
RNF02	Disponibilidad	99% uptime (limitado por servicios gratuitos)
RNF03	Seguridad	Contraseñas hasheadas, tokens JWT, HTTPS
RNF04	Usabilidad	Interfaz intuitiva, responsive
RNF05	Mantenibilidad	Código modular, documentado
RNF06	Escalabilidad	Arquitectura que permita crecimiento

Tabla 4: Requerimientos no funcionales

3.4. Diseño de la arquitectura

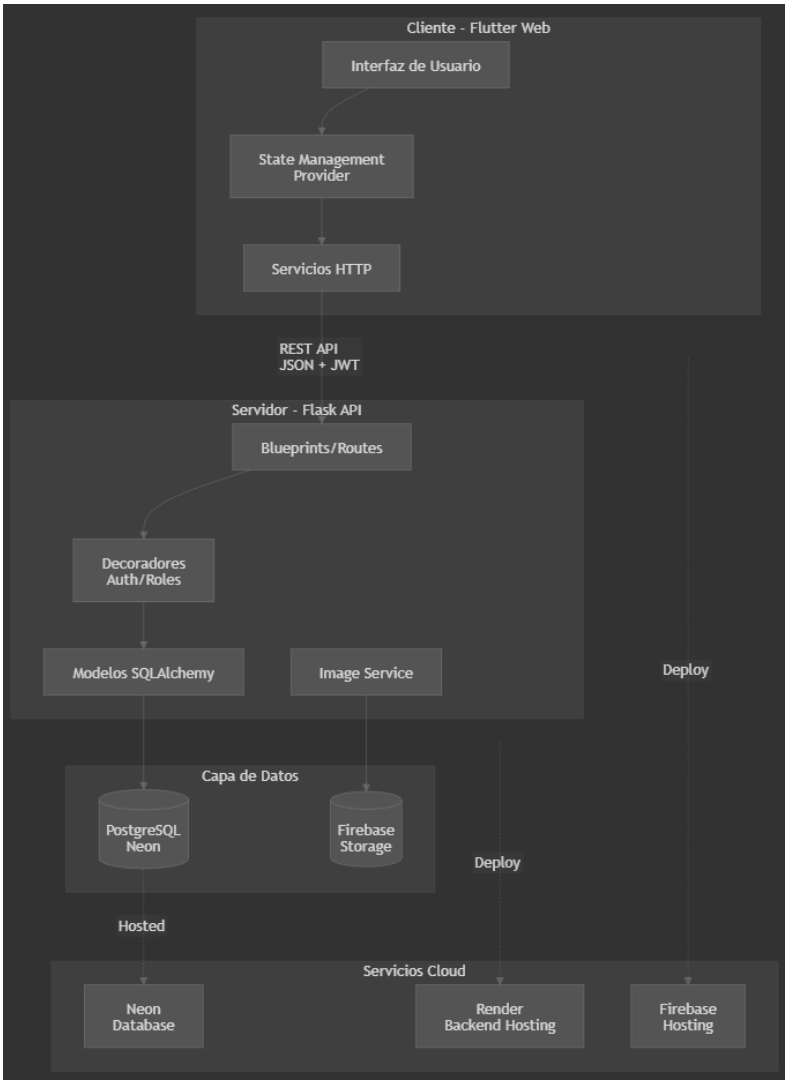


Fig. 1: Arquitectura general del sistema de agendamiento ITE VR

Como se observa en la figura 1, la arquitectura del sistema sigue el patrón de tres capas.

3.5. Diseño de la base de datos

El diseño de la base de datos se realizó siguiendo el proceso de normalización hasta la tercera forma normal (3NF) para garantizar la integridad de los datos y evitar redundancia

Diagrama entidad-relación de la base de datos. Las entidades principales y sus relaciones:

1. Sede → Facultad → Carrera: Jerarquía de ubicación académica (1:N)
2. Carrera → Usuario: Un usuario puede pertenecer a una carrera (N:1)
3. Usuario → Agenda: Un usuario puede tener múltiples agendas (1:N)
4. Aula → Agenda: Un aula puede tener múltiples agendas (1:N)
5. Usuario → Capacitación: Un usuario puede tener múltiples solicitudes de capacitación (1:N)
6. Usuario → Aula (técnico): Un administrador puede ser técnico de múltiples aulas (1:N)

La figura 3 en la siguiente página muestra el diagrama entidad-relación de las bases de datos. En la figura 4 en la página 19 se puede observar el diagrama de flujo del proceso de agendamiento.

3.6. Desarrollo del backend

El desarrollo del backend siguió una estructura modular organizada de la siguiente manera:

```
app/
├── __init__.py          # Factory de la aplicación Flask
├── models.py            # Modelos SQLAlchemy
├── decorators.py        # Decoradores de autenticación
├── utils.py             # Funciones utilitarias
├── routes/
│   ├── __init__.py
│   ├── auth.py          # Autenticación (7 endpoints)
│   ├── aulas.py         # Gestión de aulas (8 endpoints)
│   ├── agendas.py       # Agendamientos (9 endpoints)
│   ├── capacitaciones.py# Capacitaciones (7 endpoints)
│   ├── admins.py        # Gestión de admins (6 endpoints)
│   ├── docentes.py      # Gestión de docentes (5 endpoints)
│   ├── landing.py       # Landing page (14 endpoints)
│   ├── uploads.py       # Subida de archivos (3 endpoints)
│   ├── stats.py         # Estadísticas (4 endpoints)
│   └── filtros.py       # Filtros públicos (4 endpoints)
├── services/
│   ├── __init__.py
│   └── image_service.py # Procesamiento de imágenes
└── static/
    └── uploads/         # Almacenamiento de imágenes
        ├── aulas/
        ├── carousel/
        ├── about/
        └── profiles/
```

Fig. 2: Estructura de directorios del backend

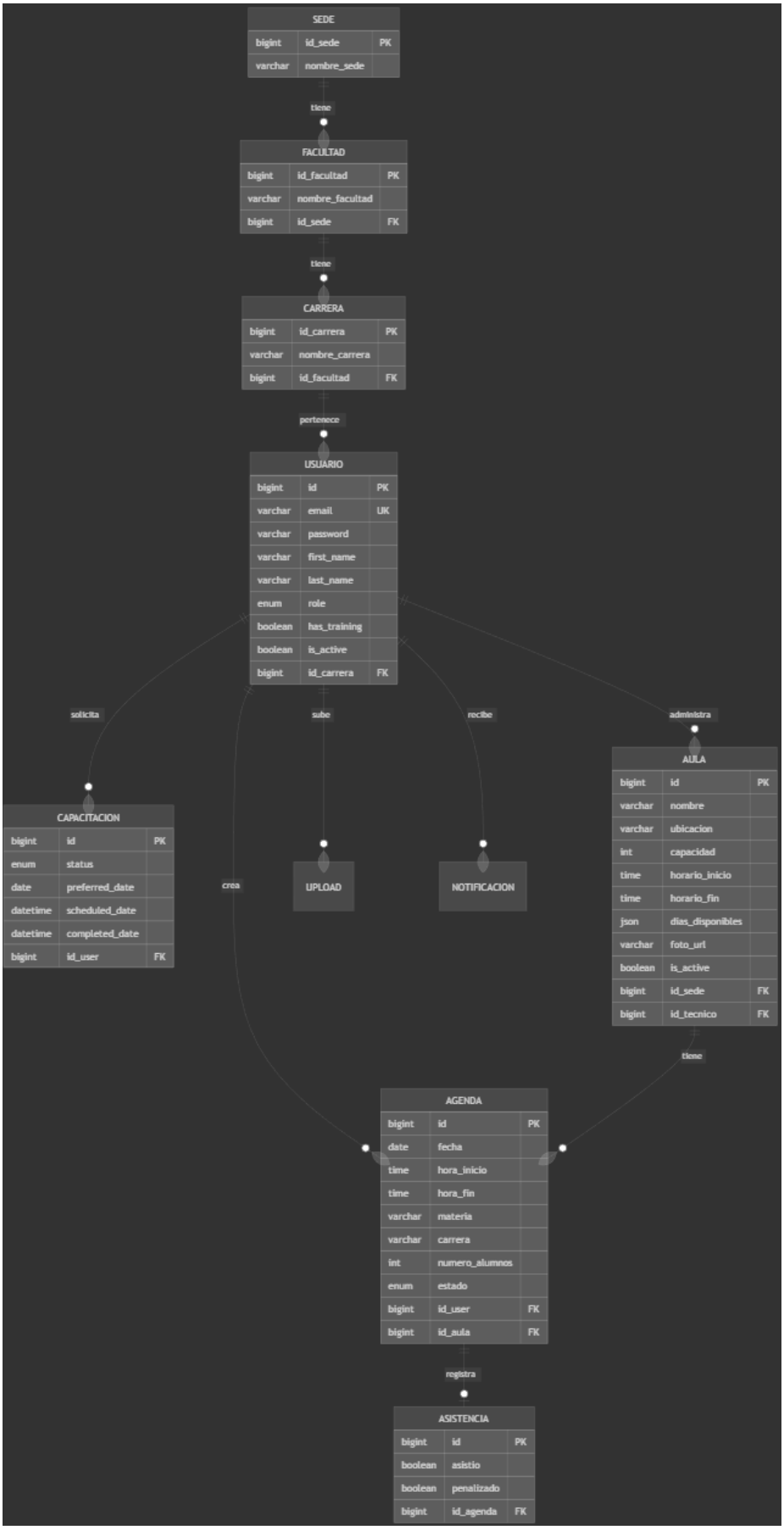


Fig. 3: Diagrama Entidad-Relación de la base de datos

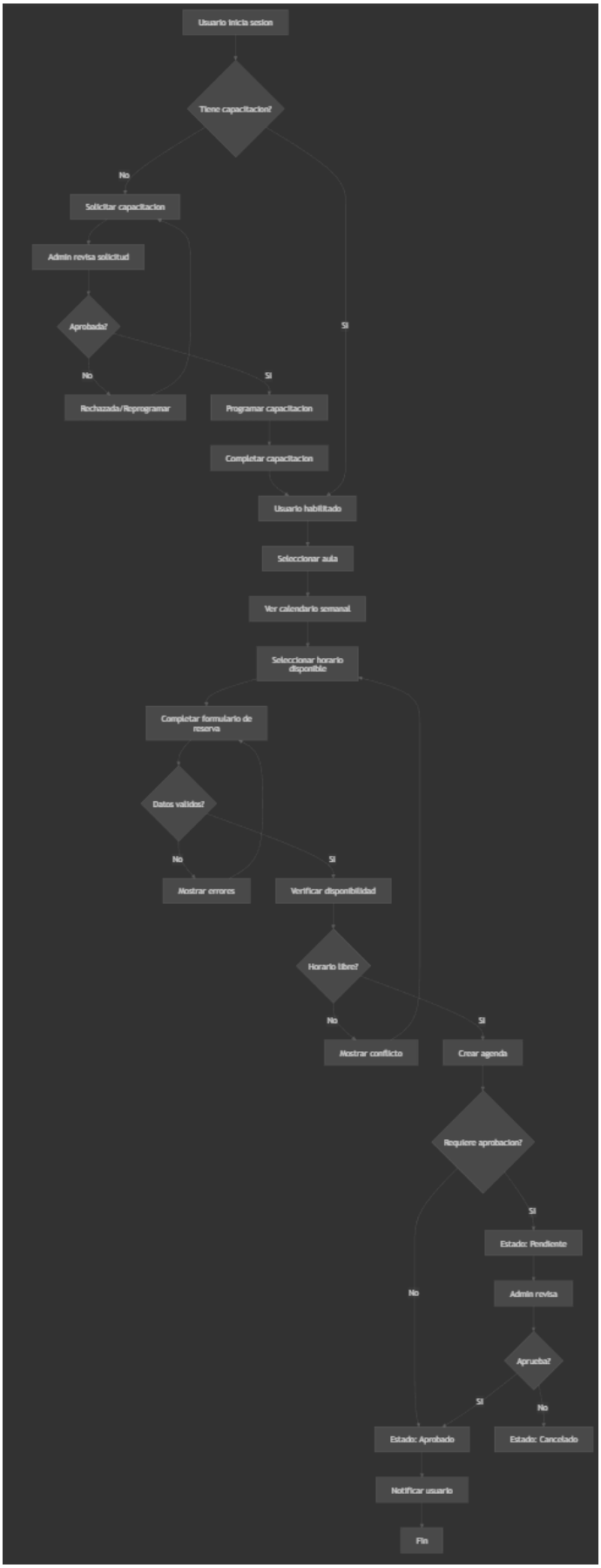


Fig. 4. Diagrama de flujo del proceso de agendamiento

3.7. Desarrollo del frontend

El desarrollo del frontend se organizó en dos sprints principales siguiendo la metodología SCRUM adaptada al proyecto:

Sprint 7: Arquitectura base y módulos de presentación. Durante este sprint se estableció la estructura fundamental del proyecto Flutter y se desarrollaron los componentes principales de la interfaz de usuario:

- Configuración del proyecto Flutter con soporte para web y las dependencias necesarias (Provider, HTTP, SharedPreferences, CarouselSlider, TableCalendar, FlutterMap)
- Implementación del sistema de temas (claro/oscuro) con persistencia de preferencias
- Implementación del sistema de internacionalización (español/inglés) con cambio dinámico de idioma
- Desarrollo de la Landing Page con secciones Hero (carrusel), Features, About, What We Do, VR Equipment y Footer
- Desarrollo del módulo de autenticación con formularios de login validados
- Creación de widgets reutilizables: AppButton, AppCard, AppModal, AppTextField, ThemedLogo

Sprint 8: Módulos funcionales y paneles de usuario. En este sprint se desarrollaron los módulos específicos para docentes y administradores:

- Implementación del DashboardLayout compartido con sidebar responsive
- Desarrollo del panel de docentes con vistas de Agendar, Agendas y Perfil
- Desarrollo del panel de administración con vistas adicionales de Gestión de Aulas, Capacitaciones y Editor de Landing Page
- Implementación del calendario semanal interactivo con selección de horarios
- Implementación del selector de aulas con vista de tarjetas y filtros
- Desarrollo de modales para creación, edición y cancelación de reservas
- Integración del selector de ubicación con mapas (flutter_map)
- Implementación del selector de iconos para personalización de características

Herramientas utilizadas:

- IDE: Visual Studio Code con extensiones Flutter y Dart

- Flutter SDK versión 3.6.0+
- Dart SDK versión 3.6.0+
- Navegadores de prueba: Google Chrome (principal), Microsoft Edge
- Control de versiones: Git con repositorio en GitHub
- Herramientas de diseño de referencia: Figma para el prototipo visual

3.8. Integración y pruebas

Las pruebas del sistema se organizaron en los siguientes niveles:

- Pruebas unitarias: Verificación de funciones individuales como validación de email, generación de tokens, y lógica de disponibilidad de horarios.
- Pruebas de integración: Verificación de la comunicación entre módulos del backend y entre frontend y backend.
- Pruebas funcionales: Verificación de casos de uso completos desde la perspectiva del usuario.
- Pruebas de aceptación: Validación con usuarios finales para confirmar que el sistema cumple con los requerimientos [46].

3.9. Despliegue en producción

El despliegue se realizó en servicios cloud gratuitos siguiendo esta estrategia:

1. **Base de datos (Neon):**
    a. Creación de proyecto y base de datos
    b. Configuración de conexión SSLEjecución de migraciones
2. **Backend (Render):**
    a. Conexión con repositorio Git
    b. Configuración de variables de entorno
    c. Despliegue automático
3. **Frontend (Firebase Hosting):**
    a. Build de producción de Flutter Web
    b. Configuración de Firebase CLI
    c. Despliegue del build
4. **Almacenamiento (Firebase Storage):**

a. Configuración de bucketReglas de seguridad

b. Integración con backend [47]

CAPÍTULO 4

DESARROLLO E IMPLEMENTACIÓN

4.1. Configuración del entorno de desarrollo

El entorno de desarrollo se configuró con las siguientes herramientas: Backend:

- Python 3.11+
- Virtual environment (venv)
- Visual Studio Code con extensiones Python
- Postman para pruebas de API

Herramientas del frontend:

Herramienta	Versión	Propósito
Flutter SDK	3.6.0+	Framework de desarrollo
Dart SDK	3.6.0+	Lenguaje de programación
Visual Studio Code	1.85+	IDE de desarrollo
Flutter Extension	3.80+	Soporte Flutter para VS Code
Dart Extension	3.80+	Soporte Dart para VS Code
Google Chrome	120+	Navegador principal de pruebas
Flutter DevTools	Integrado	Herramientas de depuración

Tabla 5: Herramientas utilizadas en el frontend, versión y propósito

```
dependencies/
├── Core/
│   └── flutter                          # sdk: flutter
├── UI Components/
│   ├── cupertino_icons                  # ^1.0.8
│   ├── carousel_slider                  # ^5.0.0
│   ├── font_awesome_flutter             # ^10.8.0
│   ├── table_calendar                   # ^3.1.3
│   └── lucide_icons_flutter             # ^1.1.7
├── State Management/
│   └── provider                         # ^6.1.2
├── Network & Storage/
│   ├── http                             # ^1.4.0
│   ├── shared_preferences               # ^2.3.5
│   └── cached_network_image             # ^3.4.1
├── Maps & Location/
│   ├── flutter_map                      # ^6.1.0
│   └── latlong2                         # ^0.9.0
├── Image Picker/
│   └── image_picker                     # ^1.0.7
└── Internationalization/
    ├── flutter_localizations            # sdk: flutter
    └── intl                             # any
```

Fig. 5: Dependencias del proyecto (pubspec.yaml)

4.2. Implementación del backend

4.2.1 Estructura del proyecto

La aplicación Flask sigue el patrón Application Factory, que permite crear instancias de la aplicación con diferentes configuraciones. En la figura 7, en la siguiente página, se puede observar la estructura del proyecto.

4.2.2 Modelos de datos

Los modelos de SQLAlchemy definen la estructura de la base de datos. A continuación, se presentan los modelos principales:

```python
class Usuario(db.Model):
    """
    Tabla unificada de usuarios (docentes, admins, superAdmins)
    Reemplaza las antiguas tablas 'usuarios' y 'administradores'
    """
    __tablename__ = 'usuarios'

    id = db.Column(mysql.BIGINT(unsigned=True), primary_key=True, autoincrement=True)
    email = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    first_name = db.Column(db.String(100), nullable=False)
    last_name = db.Column(db.String(100), nullable=False)
    phone = db.Column(db.String(20))

    # Rol del usuario: teacher (docente), admin, superAdmin
    role = db.Column(db.Enum('teacher', 'admin', 'superAdmin', name='user_role'),
                     default='teacher')

    # Información académica (para docentes)
    faculty = db.Column(db.String(255))
    position = db.Column(db.String(255))
    id_carrera = db.Column(mysql.BIGINT(unsigned=True),
                           db.ForeignKey('carreras.id_carrera'))

    # Estado de capacitación
    has_training = db.Column(db.Boolean, default=False)
    training_date = db.Column(db.Date)

    # Estado de la cuenta
    is_active = db.Column(db.Boolean, default=True)

    # Timestamps
    created_at = db.Column(db.DateTime, server_default=db.func.current_timestamp())
    updated_at = db.Column(db.DateTime, server_default=db.func.current_timestamp(),
                           onupdate=db.func.current_timestamp())

    # Relaciones
    agendas = db.relationship('Agenda', backref='usuario', lazy=True,
                              foreign_keys='Agenda.id_user')
    capacitaciones = db.relationship('Capacitacion', backref='usuario', lazy=True)

    def to_dict(self):
        """Serializa el usuario a diccionario"""
        return {
            'id': self.id,
            'email': self.email,
            'first_name': self.first_name,
            'last_name': self.last_name,
            'phone': self.phone,
            'role': self.role,
            'has_training': self.has_training,
            'is_active': self.is_active,
            'created_at': self.created_at.isoformat() if self.created_at else None
        }
```

Fig. 6: Modelos de SQLAlchemy

```
"""
Aplicación Flask - Sistema de Agendamiento ITE VR
"""
from flask import Flask, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_cors import CORS
from flask_bcrypt import Bcrypt
from flask_migrate import Migrate
from dotenv import load_dotenv
import os

# Inicializar extensiones
db = SQLAlchemy()
bcrypt = Bcrypt()
migrate = Migrate()


def create_app():
    """Factory function para crear la aplicación Flask"""
    load_dotenv()

    app = Flask(  name  )

    # Configuración CORS
    CORS(app, resources={
        r"/api/*": {
            "origins": "*",
            "methods": ["GET", "POST", "PUT", "DELETE", "OPTIONS"],
            "allow_headers": ["Content-Type", "Authorization"]
        }
    })

    # Configuración de la aplicación
    app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv('DATABASE_URL')
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    app.config['SECRET_KEY'] = os.getenv('SECRET_KEY')
    app.config['JWT_SECRET_KEY'] = os.getenv('JWT_SECRET_KEY')

    # Configuración de uploads
    app.config['MAX_CONTENT_LENGTH'] = 5 * 1024 * 1024         # 5MB max
    app.config['UPLOAD_FOLDER'] = os.path.join(app.root_path, 'static', 'uploads')

    # Inicializar extensiones con la app
    db.init_app(app)
    bcrypt.init_app(app)
    migrate.init_app(app, db)

    # Crear carpetas de uploads si no existen
    upload_folders = ['aulas', 'carousel', 'about', 'profiles']
    for folder in upload_folders:
        folder_path = os.path.join(app.config['UPLOAD_FOLDER'], folder)
        os.makedirs(folder_path, exist_ok=True)

    # Registro de blueprints
    from app.routes.auth import auth_bp
    app.register_blueprint(auth_bp, url_prefix='/api/auth')

    from app.routes.aulas import aulas_bp
    app.register_blueprint(aulas_bp, url_prefix='/api/aulas')

    from app.routes.agendas import agendas_bp
    app.register_blueprint(agendas_bp, url_prefix='/api/agendas')

    # ... (registro de demás blueprints)

    # Endpoint de health check
    @app.route('/api/health')
    def health_check():
        try:
            db.session.execute(db.text('SELECT 1'))
            db_status = 'connected'
        except Exception as e:
            db_status = f'error: {str(e)}'

        return jsonify({
            'status': 'healthy' if db_status == 'connected' else 'unhealthy',
            'database': db status
        }), 200 if db_status == 'connected' else 500

    # Manejadores de errores globales
    @app.errorhandler(404)
    def not_found(error):
        return jsonify({
            'error': 'Recurso no encontrado',
            'code': 'NOT_FOUND'
        }), 404

    return app
```

Fig. 7: Estructura del proyecto

Modelo Usuario (Tabla unificada):

Campo	Tipo	Restricciones	Descripción
id	BIGINT	PK, AUTO_INCREMENT	Identificador único
email	VARCHAR(100)	UNIQUE, NOT NULL	Correo institucional
password	VARCHAR(255)	NOT NULL	Hash bcrypt de contraseña
first_name	VARCHAR(100)	NOT NULL	Nombres
last_name	VARCHAR(100)	NOT NULL	Apellidos
phone	VARCHAR(20)	NULL	Teléfono de contacto
role	ENUM	DEFAULT 'teacher'	Rol del usuario
has_training	BOOLEAN	DEFAULT FALSE	Estado de capacitación
is_active	BOOLEAN	DEFAULT TRUE	Estado de la cuenta

Tabla 6: Estructura de la tabla usuarios

La figura 8 en la siguiente página muestra el código del modelo aula.

4.2.3 Sistema de autenticación

El sistema de autenticación implementa el flujo JWT con tokens de acceso y refresco se muestra en la figura 9 de la página 28, donde se puede apreciar el código de las funciones utilitarias de autenticación (app/utils.py).

4.2.4 Decoradores de autorización

Los decoradores implementan el control de acceso basado en roles. En la figura 10 de la página 29 se muestra el código para los decoradores que implementan el control de acceso.

4.2.5 Endpoints implementados

Módulo	Ruta Base	Endpoints	Descripción
Auth	/api/auth	8	Registro, login, logout, perfil
Aulas	/api/aulas	8	CRUD de aulas, disponibilidad
Agendas	/api/agendas	9	CRUD de reservas, calendario
Capacitaciones	/api/capacitaciones	7	Solicitudes y gestion
Admins	/api/admins	6	Gestión de administradores
Docentes	/api/docentes	5	Gestión de docents
Landing	/api/landing	16	Contenido dinámico
Uploads	/api/uploads	3	Subida de archivos
Stats	/api/stats	4	Estadísticas
Filtros	/api/filtros	4	Sedes, facultades, carreras
Total		67	

Tabla 7: Resumen de endpoints por módulo

Los códigos para los ednpoints de creación de agenda y de disponibilidad de aula se encuentran indicados en las figuras 11 y 12, en las páginas 30 y 31 respectivamente.

```
class Aula(db.Model):
    """
    Aulas de VR/Laboratorios disponibles para agendamiento
    """
    __tablename__ = 'aulas'

    id = db.Column(mysql.BIGINT(unsigned=True), primary_key=True, autoincrement=True)
    nombre = db.Column(db.String(100), nullable=False)
    ubicacion = db.Column(db.String(100), nullable=False)
    capacidad = db.Column(db.Integer, nullable=False, default=15)

    # Horarios de operación
    horario_inicio = db.Column(db.Time, default=time(7, 0))
    horario_fin = db.Column(db.Time, default=time(16, 0))
    almuerzo_inicio = db.Column(db.Time, default=time(12, 0))
    almuerzo_fin = db.Column(db.Time, default=time(13, 0))

    # Días disponibles (JSON array: [1,2,3,4,5] = Lun-Vie)
    dias_disponibles = db.Column(db.JSON, default=[1, 2, 3, 4, 5])

    # Multimedia y descripción
    foto_url = db.Column(db.String(500))
    descripcion = db.Column(db.Text)

    # Ubicación geográfica
    latitud = db.Column(db.Numeric(10, 8))
    longitud = db.Column(db.Numeric(11, 8))

    # Estado (soft delete)
    is_active = db.Column(db.Boolean, default=True)

    # Relaciones
    id_sede = db.Column(mysql.BIGINT(unsigned=True), db.ForeignKey('sedes.id_sede'))
    id_facultad = db.Column(mysql.BIGINT(unsigned=True),
                            db.ForeignKey('facultades.id facultad'))
    id_tecnico = db.Column(mysql.BIGINT(unsigned=True), db.ForeignKey('usuarios.id'))

Modelo Agenda:
class Agenda(db.Model):
    """Agendamientos/Reservas de aulas"""
    __tablename__ = 'agendas'

    id = db.Column(mysql.BIGINT(unsigned=True), primary_key=True, autoincrement=True)

    # Fecha y hora
    fecha = db.Column(db.Date, nullable=False)
    hora_inicio = db.Column(db.Time, nullable=False)
    hora_fin = db.Column(db.Time, nullable=False)

    # Relaciones
    id_user = db.Column(mysql.BIGINT(unsigned=True),
                        db.ForeignKey('usuarios.id'), nullable=False)
    id_aula = db.Column(mysql.BIGINT(unsigned=True),
                        db.ForeignKey('aulas.id'), nullable=False)

    # Información de la clase
    docente_nombre = db.Column(db.String(100))
    materia = db.Column(db.String(100), nullable=False)
    ciclo = db.Column(db.String(20), nullable=False)
    carrera = db.Column(db.String(100), nullable=False)
    paralelo = db.Column(db.String(10), nullable=False)
    grupo = db.Column(db.String(50), default='Grupo único')
    numero_alumnos = db.Column(db.Integer, nullable=False)

    # Tipo y estado
    tipo = db.Column(db.Enum('regular', 'lunch', 'blocked', name='tipo_agenda'),
                     default='regular')
    capacitacion = db.Column(db.Boolean, nullable=False, default=False)
    estado = db.Column(db.Enum('pendiente', 'aprobado', 'cancelado', 'penalizado',
                               name='estado_agenda'), default='pendiente')

    # Cancelación
    cancellation_reason = db.Column(db.Text)
    cancelled_by = db.Column(mysql.BIGINT(unsigned=True), db.ForeignKey('usuarios.id'))
    cancelled_at = db.Column(db.DateTime)
```

Fig. 8: Modelo aula

```python
def validate_email(email: str) -> Tuple[bool, str]:
    """
    Valida que el email sea válido y del dominio institucional.
    """
    if not email:
        return False, "El correo es requerido"

    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if not re.match(pattern, email):
        return False, "Formato de correo inválido"

    if not email.endswith('@ucacue.edu.ec'):
        return False, "Solo se permiten correos institucionales (@ucacue.edu.ec)"

    return True, ""

def generate_token(user_id: int, role: str, expires_hours: int = 6) -> str:
    """
    Genera un token JWT para el usuario.
    """
    payload = {
        'user_id': user_id,
        'role': role,
        'exp': datetime.utcnow() + timedelta(hours=expires_hours),
        'iat': datetime.utcnow()
    }

    return jwt.encode(
        payload,
        os.getenv('JWT_SECRET_KEY'),
        algorithm='HS256'
    )

def generate_refresh_token(user_id: int) -> str:
    """
    Genera un token de refresco con mayor duración (30 días).
    """
    payload = {
        'user_id': user_id,
        'type': 'refresh',
        'exp': datetime.utcnow() + timedelta(days=30),
        'iat': datetime.utcnow()
    }

    return jwt.encode(
        payload,
        os.getenv('JWT_SECRET_KEY'),
        algorithm='HS256'
    )

Endpoint de login (app/routes/auth.py):
@auth_bp.route('/login', methods=['POST'])
def login():
    """
    Iniciar sesión para usuarios (docentes).
    """
    data = request.get_json()

    if not data:
        return jsonify({'error': 'Datos requeridos'}), 400

    email = data.get('email', '').strip().lower()
    password = data.get('password', '')

    if not email or not password:
        return jsonify({'error': 'Email y contraseña son requeridos'}), 400

    # Buscar usuario
    usuario = Usuario.query.filter_by(email=email).first()

    if not usuario:
        return jsonify({'error': 'Credenciales inválidas'}), 401

    if not usuario.is_active:
        return jsonify({'error': 'Cuenta desactivada. Contacte al administrador'}), 401

    # Verificar contraseña
    if not bcrypt.check_password_hash(usuario.password, password):
        return jsonify({'error': 'Credenciales inválidas'}), 401

    # Generar tokens
    token = generate_token(usuario.id, usuario.role)
    refresh_token = generate_refresh_token(usuario.id)

    return jsonify({
        'mensaje': 'Login exitoso',
        'token': token,
        'refresh_token': refresh_token,
        'user': usuario.to_dict()
    }), 200
```

Fig. 9: Funciones utilitarias de autenticación (app/utils.py)

```python
def token_required(f):
    """
    Decorador que requiere un token JWT válido.
    El usuario autenticado se almacena en g.current_user
    """
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None

        # Obtener token del header Authorization
        auth_header = request.headers.get('Authorization')
        if auth_header:
            try:
                parts = auth_header.split()
                if len(parts) == 2 and parts[0].lower() == 'bearer':
                    token = parts[1]
            except Exception:
                pass

        if not token:
            return jsonify({
                'error': 'Token de autenticación requerido',
                'code': 'TOKEN_MISSING'
            }), 401

        try:
            payload = jwt.decode(
                token,
                os.getenv('JWT_SECRET_KEY'),
                algorithms=['HS256']
            )

            user_id = payload.get('user_id')
            current_user = Usuario.query.get(user_id)

            if not current_user or not current_user.is_active:
                return jsonify({'error': 'Usuario no válido'}), 401

            g.current_user = current_user

        except jwt.ExpiredSignatureError:
            return jsonify({'error': 'Token expirado'}), 401
        except jwt.InvalidTokenError:
            return jsonify({'error': 'Token inválido'}), 401

        return f(*args, **kwargs)

    return decorated


def admin_required(f):
    """
    Decorador que requiere rol de admin o superAdmin.
    Debe usarse después de @token_required
    """
    @wraps(f)
    def decorated(*args, **kwargs):
        if g.current_user.role not in ['admin', 'superAdmin']:
            return jsonify({
                'error': 'Acceso denegado. Se requiere rol de administrador',
                'code': 'ADMIN REQUIRED'
            }), 403

        return f(*args, **kwargs)

    return decorated


def training_required(f):
    """
    Decorador que requiere que el usuario haya completado la capacitación.
    Los admins están exentos de este requisito.
    """
    @wraps(f)
    def decorated(*args, **kwargs):
        # Los admins no necesitan capacitación
        if g.current_user.role in ['admin', 'superAdmin']:
            return f(*args, **kwargs)

        if not g.current_user.has_training:
            return jsonify({
                'error': 'Se requiere completar la capacitación',
                'code': 'TRAINING REQUIRED'
            }), 403

        return f(*args, **kwargs)

    return decorated
```

Fig. 10: Decoradores para implementar el control de acceso basado en roles

```python
@agendas_bp.route('', methods=['POST'])
@token_required
@training required
def crear_agenda():
    """
    Crear una nueva agenda/reserva.
    Requiere que el usuario haya completado la capacitación.
    """
    data = request.get_json()
    user = g.current_user

    # Validar campos requeridos
    required_fields = ['id_aula', 'fecha', 'hora_inicio', 'materia',
                       'carrera', 'ciclo', 'paralelo', 'numero_alumnos']
    for field in required fields:
        if field not in data:
            return jsonify({'error': f'Campo requerido: {field}'}), 400

    # Parsear fecha y hora
    fecha = parse_date(data['fecha'])
    if not fecha:
        return jsonify({'error': 'Formato de fecha inválido'}), 400

    hora_inicio = time(data['hora_inicio'], 0)
    hora_fin = time(data.get('hora_fin', data['hora_inicio'] + 1), 0)

    # Verificar que el aula existe y está activa
    aula = Aula.query.get(data['id_aula'])
    if not aula or not aula.is_active:
        return jsonify({'error': 'Aula no disponible'}), 400

    # Verificar capacidad
    if data['numero_alumnos'] > aula.capacidad:
        return jsonify({
            'error': f'Excede la capacidad del aula ({aula.capacidad})'
        }), 400

    # Verificar disponibilidad del día
    dia semana = fecha.isoweekday()
    if dia_semana not in (aula.dias_disponibles or [1, 2, 3, 4, 5]):
        return jsonify({'error': 'El aula no está disponible este día'}), 400

    # Verificar conflicto de horario
    conflicto = Agenda.query.filter by(
        id_aula=data['id_aula'],
        fecha=fecha,
        hora_inicio=hora_inicio
    ).filter(Agenda.estado.in (['pendiente', 'aprobado'])).first()

    if conflicto:
        return jsonify({'error': 'El horario ya está ocupado'}), 409

    # Crear agenda
    nueva_agenda = Agenda(
        fecha=fecha,
        hora inicio=hora inicio,
        hora_fin=hora_fin,
        id_user=user.id,
        id_aula=data['id_aula'],
        docente nombre=f"{user.first name} {user.last name}",
        materia=data['materia'],
        ciclo=data['ciclo'],
        carrera=data['carrera'],
        paralelo=data['paralelo'],
        grupo=data.get('grupo', 'Grupo único'),
        numero_alumnos=data['numero_alumnos'],
        estado='pendiente'
    )

    try:
        db.session.add(nueva_agenda)
        db.session.commit()

        return jsonify({
            'mensaje': 'Agendamiento creado correctamente',
            'agenda': nueva agenda.to dict()
        }), 201

    except Exception as e:
        db.session.rollback()
        return jsonify({'error': str(e)}), 500
```

Fig. 11: Endpoint de creación de agenda

```python
@aulas_bp.route('/<int:id>/disponibilidad', methods=['GET'])
def obtener_disponibilidad(id):
    """
    Obtener horarios disponibles de un aula para una fecha.
    """
    aula = Aula.query.get_or_404(id)

    fecha_str = request.args.get('fecha')
    if not fecha_str:
        return jsonify({'error': 'Se requiere el parámetro fecha'}), 400

    fecha = parse_date(fecha_str)
    if not fecha:
        return jsonify({'error': 'Formato de fecha inválido'}), 400

    # Verificar día disponible
    dia_semana = fecha.isoweekday()
    if dia_semana not in (aula.dias_disponibles or [1, 2, 3, 4, 5]):
        return jsonify({
            'disponibles': [],
            'mensaje': 'El aula no está disponible en este día'
        }), 200

    # Obtener horas ocupadas
    agendas_dia = Agenda.query.filter_by(
        id_aula=id,
        fecha=fecha
    ).filter(Agenda.estado.in_(['pendiente', 'aprobado'])).all()

    horas_ocupadas = [a.hora_inicio for a in agendas_dia]

    # Calcular horas disponibles
    disponibles = get_available_hours(
        aula.horario_inicio or time(7, 0),
        aula.horario_fin or time(16, 0),
        aula.almuerzo_inicio or time(12, 0),
        aula.almuerzo_fin or time(13, 0),
        horas_ocupadas
    )

    return jsonify({
        'fecha': fecha_str,
        'aula': aula.nombre,
        'disponibles': disponibles,
        'ocupadas': [h.strftime('%H:%M') for h in horas_ocupadas]
    }), 200
```

Fig. 12: Endpoint de disponibilidad de aula

4.2.6. Servicio de procesamiento de imágenes

El servicio de imágenes maneja la validación, redimensionamiento y almacenamiento de archivos:

```python
class ImageService:
    """Servicio para procesamiento y almacenamiento de imágenes"""

    ALLOWED_EXTENSIONS = {'jpg', 'jpeg', 'png', 'webp', 'gif'}
    MAX_FILE_SIZE = 5 * 1024 * 1024  # 5MB

    RESIZE_CONFIG = {
        'aula': (800, 600),
        'carousel': (1920, 1080),
        'about': (1200, 800),
        'profile': (400, 400)
    }

    @staticmethod
    def save_image(file: FileStorage, folder: str,
                   resize: Optional[Tuple[int, int]] = None) -> Tuple[Optional[str], Optional[str]]:
        """
        Guarda y procesa una imagen.

        Returns:
            Tuple[str, None]: (ruta relativa, None) si éxito
            Tuple[None, str]: (None, mensaje error) si error
        """
        # Validar archivo
        if not file or not file.filename:
            return None, "No se proporcionó ningún archivo"

        if not ImageService.allowed_file(file.filename):
            return None, "Formato no permitido"

        # Verificar tamaño
        file.seek(0, 2)
        file_size = file.tell()
        file.seek(0)

        if file_size > ImageService.MAX_FILE_SIZE:
            return None, "El archivo excede el tamaño máximo de 5MB"

        # Generar nombre único
        stored_name = f"{uuid.uuid4()}.jpg"
        upload_path = os.path.join(UPLOAD_FOLDER, folder)
        os.makedirs(upload_path, exist_ok=True)
        filepath = os.path.join(upload_path, stored_name)

        # Determinar tamaño de redimensionamiento
        if resize is None:
            resize = ImageService.RESIZE_CONFIG.get(folder, (800, 600))

        try:
            # Procesar con PIL
            img = Image.open(file)

            # Convertir a RGB
            if img.mode in ('RGBA', 'P', 'LA'):
                background = Image.new('RGB', img.size, (255, 255, 255))
                if img.mode == 'RGBA':
                    background.paste(img, mask=img.split()[3])
                else:
                    background.paste(img)
                img = background
            elif img.mode != 'RGB':
                img = img.convert('RGB')

            # Redimensionar
            img.thumbnail(resize, Image.Resampling.LANCZOS)

            # Guardar como JPEG optimizado
            img.save(filepath, 'JPEG', quality=85, optimize=True)

            return f"/static/uploads/{folder}/{stored_name}", None

        except Exception as e:
            return None, f"Error al procesar la imagen: {str(e)}"
```

Fig. 13: Servicio de imágenes

4.3. Implementación del frontend

4.3.1 Estructura del proyecto Flutter

El proyecto Flutter sigue una arquitectura modular organizada por características (feature-based architecture), que facilita la escalabilidad y el mantenimiento del código:

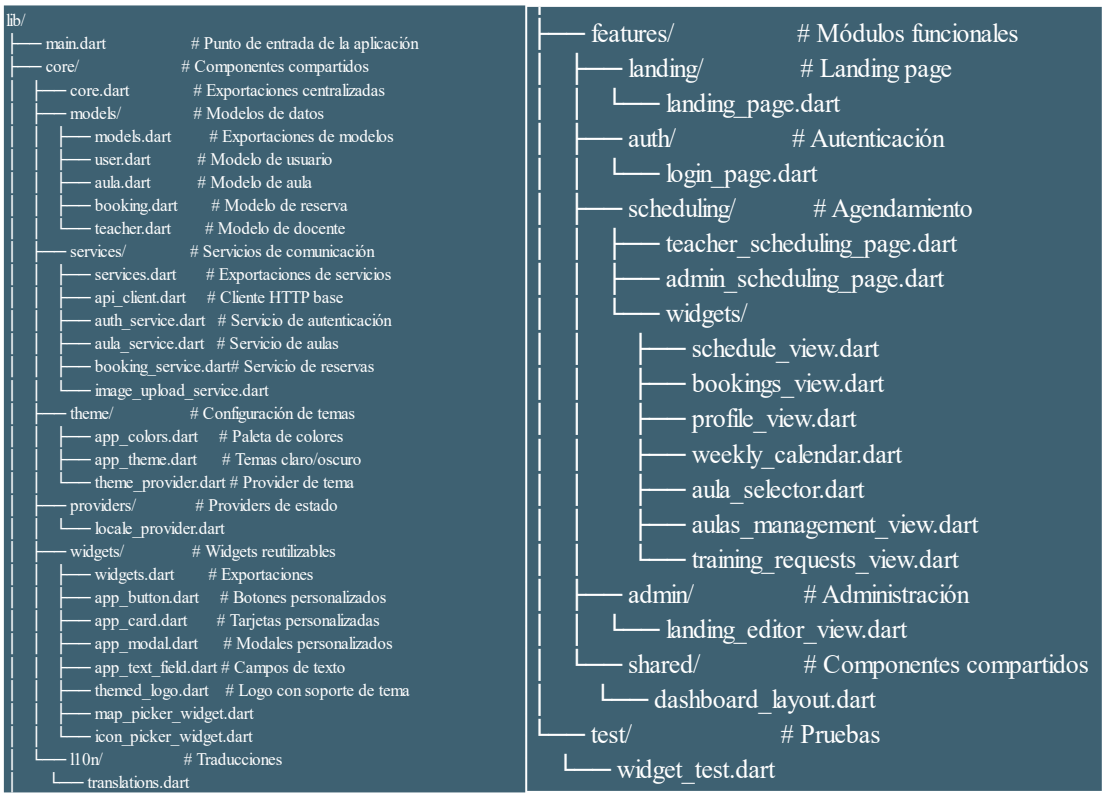


Fig. 14: Estructura de directorios del proyecto Flutter

La organización del proyecto sigue los siguientes principios:

- core/: Contiene todos los elementos compartidos entre módulos, incluyendo modelos de datos, servicios de comunicación con la API, configuración de temas y widgets reutilizables.
- features/: Agrupa el código por funcionalidad de negocio. Cada feature contiene sus propias pantallas y widgets específicos.
- Separación de responsabilidades: Los modelos definen la estructura de datos, los servicios manejan la comunicación con el backend, los providers gestionan el estado global, y los widgets construyen la interfaz.

4.3.2 Modelos de datos

Los modelos de datos en Dart implementan la serialización JSON para la comunicación con la API REST. Se utilizan clases inmutables con el patrón copyWith para facilitar actualizaciones de estado:

Modelo Usuario:

```
/// Modelo de Usuario base
class User {
  final String id;
  final String firstName;
  final String lastName;
  final String email;
  final String? phone;
  final UserRole role;
  final DateTime? createdAt;
  const User({
    required this.id,
    required this.firstName,
    required this.lastName,
    required this.email,
    this.phone,
    required this.role,
    this.createdAt,
  });
  String get fullName => '$firstName $lastName';
  String get initials => '${firstName.isNotEmpty ? firstName[0] : ''}${lastName.isNotEmpty ? lastName[0] : ''}'.toUpperCase();
  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'] as String,
      firstName: json['firstName'] as String,
      lastName: json['lastName'] as String,
      email: json['email'] as String,
      phone: json['phone'] as String?,
      role: UserRole.values.firstWhere(
        (e) => e.name == json['role'],
        orElse: () => UserRole.teacher,
      ),
      createdAt: json['createdAt'] != null
          ? DateTime.parse(json['createdAt'] as String)
          : null,
    );
  }
  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'firstName': firstName,
      'lastName': lastName,
      'email': email,
      'phone': phone,
      'role': role.name,
      'createdAt': createdAt?.toIso8601String(),
    };
  }
}
/// Roles de usuario disponibles
enum UserRole {
  teacher,
  admin,
  superAdmin,
}
```

Fig. 13: Modelo de Usuario

Modelo aula:

```
/// Modelo de Aula VR
class Aula {
 final String id;
 final String name;
 final String location;
 final int capacity;
 final String schedule; // Horario de operación
 final String? lunchBreak; // Horario de almuerzo
 final List<int> availableDays; // Días disponibles (1=Lun...7=Dom)
 final String? imageUrl;
 final double? latitude;
 final double? longitude;
 final bool isActive;
 final String? description;
 final String? technicianId;
 final String? technicianName;
 final String? technicianEmail;
 final String? technicianPhone;
 const Aula({
 required this.id,
 required this.name,
 required this.location,
 required this.capacity,
 required this.schedule,
 this.lunchBreak,
 this.availableDays = const [1, 2, 3, 4, 5],
 this.imageUrl,
 this.latitude,
 this.longitude,
 this.isActive = true,
 this.description,
 this.technicianId,
 this.technicianName,
 this.technicianEmail,
 this.technicianPhone,
 });
 /// Indica si tiene ubicación GPS configurada
 bool get hasGpsLocation => latitude != null && longitude != null;

 /// Indica si tiene técnico asignado
 bool get hasTechnician => technicianId != null && technicianName != null;
 factory Aula.fromJson(Map<String, dynamic> json) {
 return Aula(
 id: json['id'] as String,
 name: json['name'] as String,
 location: json['location'] as String,
 capacity: json['capacity'] as int,
 schedule: json['schedule'] as String,
 lunchBreak: json['lunchBreak'] as String?,
 availableDays: (json['availableDays'] as List<dynamic>?)
 ?.map((e) => e as int).toList() ?? [1, 2, 3, 4, 5],
 imageUrl: json['imageUrl'] as String?,
 latitude: json['latitude'] as double?,
 longitude: json['longitude'] as double?,
 isActive: json['isActive'] as bool? ?? true,
 description: json['description'] as String?,
 technicianId: json['technicianId'] as String?,
 technicianName: json['technicianName'] as String?,
 technicianEmail: json['technicianEmail'] as String?,
 technicianPhone: json['technicianPhone'] as String?,
 );
 }
}
```

Fig. 14: Modelo de Aula

Modelo Booking (Reserva/Agendamiento):

```
/// Tipos de reserva
enum BookingType { regular, lunch, blocked }
/// Estados de reserva
enum BookingStatus { active, cancelled, completed }
/// Modelo de Reserva/Agendamiento
class Booking {
 final String id;
 final String teacherId;
 final String teacherName;
 final String aulaId;
 final String aulaName;
 final String subject;
 final String career;
 final String? parallel;
 final String? cycle;
 final int? numStudents;
 final String group;
 final List<String> schedule;
 final DateTime date;
 final int startHour;
 final int endHour;
 final String? notes;
 final BookingType type;
 final BookingStatus status;
 final String? cancellationReason;
 final String? cancelledBy;
 final DateTime? cancelledAt;
 const Booking({
 required this.id,
 required this.teacherId,
 required this.teacherName,
 required this.aulaId,
 required this.aulaName,
 required this.subject,
 required this.career,
 this.parallel,
 this.cycle,
 this.numStudents,
 required this.group,
 required this.schedule,
 required this.date,
 this.startHour = 7,
 this.endHour = 8,
 this.notes,
 this.type = BookingType.regular,
 this.status = BookingStatus.active,
 this.cancellationReason,
 this.cancelledBy,
 this.cancelledAt,
 });
 bool get isActive => status == BookingStatus.active;
 bool get isCancelled => status == BookingStatus.cancelled;
 /// Marca como cancelada con motivo
 Booking cancel({required String reason, required String cancelledByUser}) =>
 copyWith(
 status: BookingStatus.cancelled,
 cancellationReason: reason,
 cancelledBy: cancelledByUser,
 cancelledAt: DateTime.now(),
 );
}
```

Fig. 15: Modelo Booking

Modelo	Campos principales	Propósito
User	id, email, firstName, lastName, role	Información del usuario autenticado
Aula	id, name, location, capacity, schedule	Información de aulas VR
Booking	id, teacherId, aulaId, date, startHour	Reservas de aulas
Teacher	extends User + hasTraining, trainingDate	Docentes con estado de capacitación

Tabla 8: Resumen de modelos de datos del frontend

4.3.3 Servicios de API

El servicio de comunicación con la API REST se implementa mediante una clase

```
/// Cliente HTTP base para comunicación con el backend
class ApiClient {
 static const String _baseUrl = 'http://localhost:5000/api';

 final http.Client _client;
 String? _authToken;

 ApiClient({http.Client? client}) : _client = client ?? http.Client();

 /// Configura el token de autenticación
 void setAuthToken(String? token) {
 _authToken = token;
 }

 /// Headers comunes para todas las peticiones
 Map<String, String> get _headers => {
 'Content-Type': 'application/json',
 'Accept': 'application/json',
 if (_authToken != null) 'Authorization': 'Bearer $_authToken',
 };

 /// GET request
 Future<ApiResponse<T>> get<T>(
 String endpoint, {
 T Function(dynamic json)? fromJson,
 Map<String, String>? queryParams,
 }) async {
 try {
 var uri = Uri.parse('$_baseUrl$endpoint');
 if (queryParams != null && queryParams.isNotEmpty) {
 uri = uri.replace(queryParameters: queryParams);
 }
 final response = await _client.get(uri, headers: _headers);
 return _handleResponse(response, fromJson);
 } catch (e) {
 return ApiResponse.error('Error de conexión: $e');
 }
 }
```

Fig. 16: Servicios de API

ApiClient que centraliza todas las peticiones HTTP:

```
/// POST request
Future<ApiResponse<T>> post<T>(
String endpoint, {
Map<String, dynamic>? body,
T Function(dynamic json)? fromJson,
}) async {
try {
final uri = Uri.parse('$_baseUrl$endpoint');
final response = await _client.post(
uri,
headers: _headers,
body: body != null ? jsonEncode(body) : null,
);
return _handleResponse(response, fromJson);
} catch (e) {
return ApiResponse.error('Error de conexión: $e');
}
}
/// Procesa la respuesta HTTP
ApiResponse<T> _handleResponse<T>(
http.Response response,
T Function(dynamic json)? fromJson,
) {
final statusCode = response.statusCode;

if (statusCode >= 200 && statusCode < 300) {
if (response.body.isEmpty) {
return ApiResponse.success(null as T);
}
final json = jsonDecode(response.body);
if (fromJson != null) {
return ApiResponse.success(fromJson(json));
}
return ApiResponse.success(json as T);
}

String errorMessage;
try {
final json = jsonDecode(response.body);
errorMessage = json['message'] ?? json['error'] ?? 'Error desconocido';
} catch (_) {
errorMessage = 'Error del servidor (código: $statusCode)';
}
return ApiResponse.error(errorMessage, statusCode: statusCode);
}
}
/// Respuesta genérica de la API
class ApiResponse<T> {
final T? data;
final String? error;
final int? statusCode;
final bool isSuccess;

factory ApiResponse.success(T data) => ApiResponse._(data: data, isSuccess: true);
factory ApiResponse.error(String message, {int? statusCode}) =>
ApiResponse._(error: message, statusCode: statusCode, isSuccess: false);
}
```

```
/// Servicio de autenticación
class AuthService {
 final ApiClient _api;
 static const String _tokenKey = 'auth_token';
 static const String _userKey = 'user_data';
 User? _currentUser;

 User? get currentUser => _currentUser;
 bool get isAuthenticated => _currentUser != null;
 bool get isAdmin => _currentUser?.role == UserRole.admin ||
 _currentUser?.role == UserRole.superAdmin;

 /// Login con email y contraseña
 Future<AuthResult> login({
 required String email,
 required String password,
 }) async {
 if (email.isEmpty || password.isEmpty) {
 return AuthResult.failure('Por favor complete todos los campos');
 }

 if (!_isValidEmail(email)) {
 return AuthResult.failure('El correo electrónico no es válido');
 }

 final response = await _api.post<Map<String, dynamic>>(
 '/auth/login',
 body: {'email': email, 'password': password},
 );

 if (response.isSuccess && response.data != null) {
 final data = response.data!;
 final token = data['token'] as String?;
 final userData = data['user'] as Map<String, dynamic>?;

 if (token != null && userData != null) {
 await _saveAuthData(token, userData);
 _api.setAuthToken(token);
 _currentUser = User.fromJson(userData);
 return AuthResult.success(_currentUser!);
 }
 }
 return AuthResult.failure(response.error ?? 'Error al iniciar sesión');
 }

 /// Cerrar sesión
 Future<void> logout() async {
 final prefs = await SharedPreferences.getInstance();
 await prefs.remove(_tokenKey);
 await prefs.remove(_userKey);
 _api.setAuthToken(null);
 _currentUser = null;
 }
}
```

Fig. 17: API Client

4.3.4 Gestión de estado

El sistema implementa el patrón Provider para la gestión de estado global. Se utilizan dos providers principales:

ThemeProvider - Gestión del tema visual:

```
/// Provider para gestión de tema (claro/oscuro)
class ThemeProvider extends ChangeNotifier {
 static const String _themeKey = 'ite-vr-theme';
 bool _isDarkMode = false;
 bool get isDarkMode => _isDarkMode;

 ThemeProvider() {
 _loadTheme();
 }

 Future<void> _loadTheme() async {
 try {
 final prefs = await SharedPreferences.getInstance();
 final savedTheme = prefs.getString(_themeKey);
 if (savedTheme != null) {
 _isDarkMode = savedTheme == 'dark';
 } else {
 // Detectar preferencia del sistema
 _isDarkMode = WidgetsBinding.instance
 .platformDispatcher.platformBrightness == Brightness.dark;
 }
 notifyListeners();
 } catch (e) {
 debugPrint('Error loading theme: $e');
 }
 }

 Future<void> toggleTheme() async {
 _isDarkMode = !_isDarkMode;
 notifyListeners();
 try {
 final prefs = await SharedPreferences.getInstance();
 await prefs.setString(_themeKey, _isDarkMode ? 'dark' :
'light');
 } catch (e) {
 debugPrint('Error saving theme: $e');
 }
 }
}
```

Fig. 18: Theme Provider

```
/// Provider para gestión de idioma
class LocaleProvider extends ChangeNotifier {
 static const String _localeKey = 'ite-vr-language';
 Locale _locale = const Locale('es');
 Locale get locale => _locale;

 String get languageCode => _locale.languageCode;
 bool get isSpanish => _locale.languageCode == 'es';
 bool get isEnglish => _locale.languageCode == 'en';

 Future<void> setLocale(Locale newLocale) async {
 if (_locale == newLocale) return;
 _locale = newLocale;
 notifyListeners();
 try {
 final prefs = await SharedPreferences.getInstance();
 await prefs.setString(_localeKey, newLocale.languageCode);
 } catch (e) {
 debugPrint('Error saving locale: $e');
 }
 }
}
Configuración de Providers en main.dart:
void main() {
 WidgetsFlutterBinding.ensureInitialized();
 runApp(const AppProviders());
}
class AppProviders extends StatelessWidget {
 const AppProviders({super.key});
 @override
 Widget build(BuildContext context) {
 return MultiProvider(
 providers: [
 ChangeNotifierProvider(create: (_) => ThemeProvider()),
 ChangeNotifierProvider(create: (_) => LocaleProvider()),
 ],
 child: const MyApp(),
 );
 }
}
```

Fig. 19: Locale Provider

LocaleProvider - Gestión de idioma:

Configuración de Providers en main.dart:

```
void main() {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(const AppProviders());
}
class AppProviders extends StatelessWidget {
  const AppProviders({super.key});
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => ThemeProvider()),
        ChangeNotifierProvider(create: (_) => LocaleProvider()),
      ],
      child: const MyApp(),
    );
  }
}
```

Fig. 20: Configuracion del Provider

4.3.5 Pantallas principales

Landing Page:

La Landing Page constituye el punto de entrada público del sistema, diseñada para presentar información sobre las aulas de realidad virtual y facilitar el acceso al sistema de agendamiento.

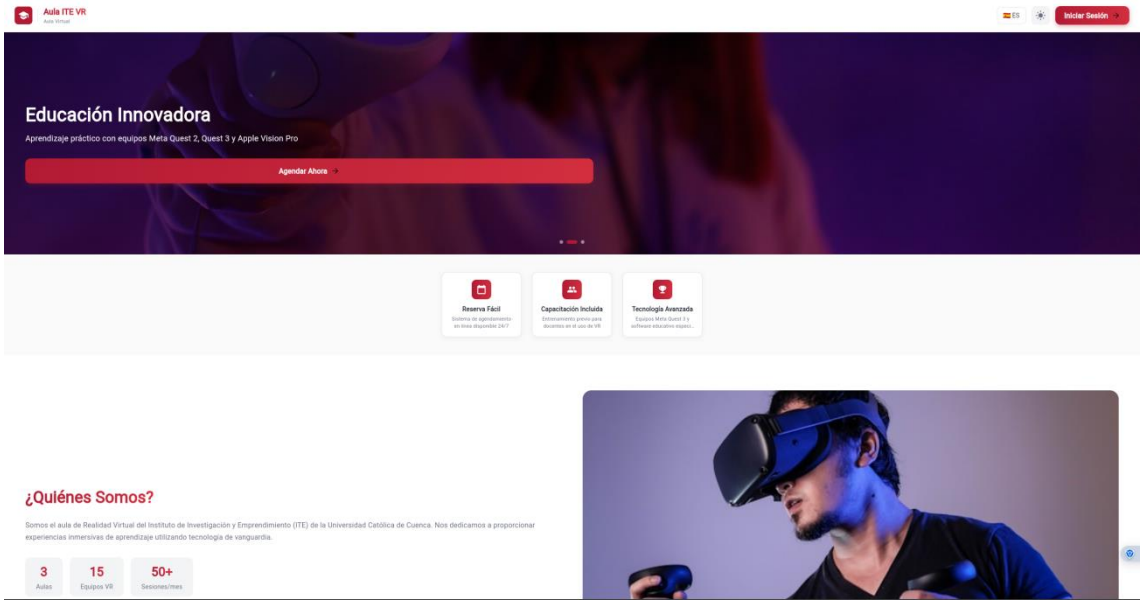


Fig. 21: Landing Page - Hero section con carrusel de imágenes

La Landing Page se estructura en las siguientes secciones:

- Hero Section: Carrusel automático con imágenes de alta calidad y llamada a la acción

- Features Section: Tarjetas compactas destacando beneficios clave (Agendamiento fácil, Capacitación incluida, Tecnología avanzada)
- About Section: Información institucional con estadísticas (3 laboratorios, 15 equipos VR, 50+ sesiones/mes)
- VR Equipment Section: Grid de tarjetas con los dispositivos disponibles (Meta Quest 3, Quest 2, Apple Vision Pro)
- CTA Section: Sección con gradiente para incentivar el registro
- Footer: Enlaces rápidos, redes sociales e información de contacto

```
class LandingPage extends StatefulWidget {
  @override
  Widget build(BuildContext context) {
    final isDark =
context.watch<ThemeProvider>().isDarkMode;
    final lang =
context.watch<LocaleProvider>().languageCode;
    final t = AppTranslations.of(lang);

    return Scaffold(
      body: SingleChildScrollView(
        child: Column(
          children: [
            _buildHeader(isDark, t),
            _buildHeroCarousel(isDark, t),
            _buildFeaturesSection(isDark, t),
            _buildAboutSection(isDark, t),
            _buildWhatWeDoSection(isDark, t),
            _buildVREquipmentSection(isDark, t),
            _buildCTASection(isDark, t),
            _buildFooter(isDark, t),
          ],
        ),
      ),
    );
  }
}
```

Fig. 22: Landing Page

Página de Login:

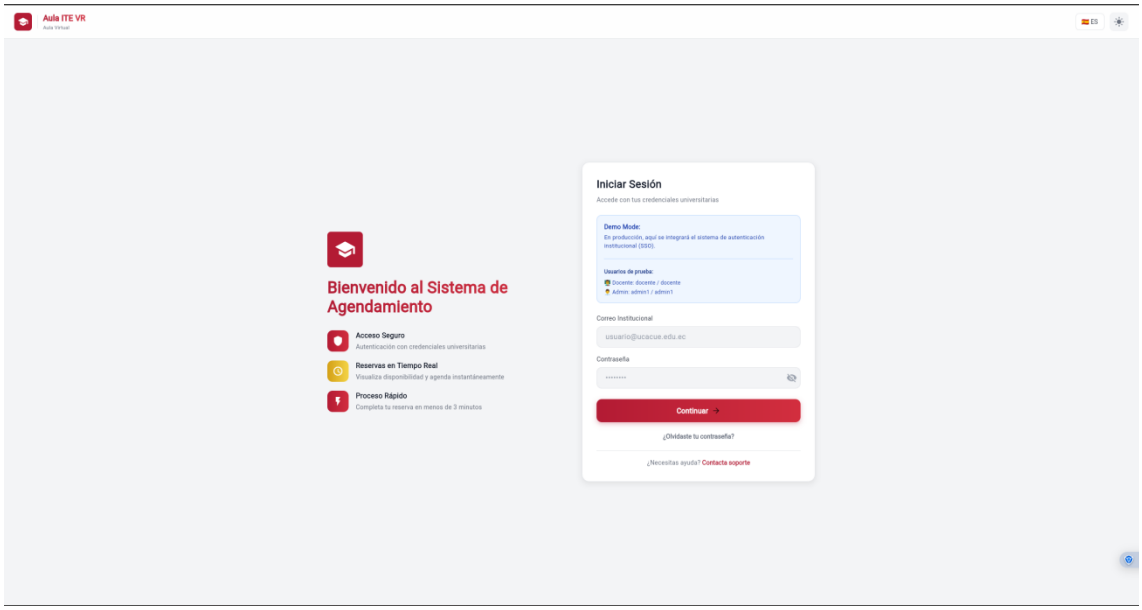


Fig. 23: Formulario de login con información de usuarios de prueba

La página de autenticación presenta un diseño de dos columnas en pantallas grandes: información del sistema a la izquierda y formulario de login a la derecha.

```
class LoginPage extends StatefulWidget {
  // Controladores de formulario
  final _usernameController = TextEditingController();
  final _passwordController = TextEditingController();
  String? _errorMessage;
  bool _isLoading = false;
  bool _obscurePassword = true;
  void _handleLogin() {
    setState(() { _errorMessage = null; _isLoading = true; });

    Future.delayed(const Duration(milliseconds: 500), () {
      final user = testUsers.firstWhere(
        (u) => u.username == _usernameController.text &&
            u.password == _passwordController.text,
        orElse: () => const TestUser(username: '', password: '', role: '', name: ''),
      );
      if (user.username.isNotEmpty) {
        if (user.role == 'teacher') {
          Navigator.pushReplacementNamed(context, '/teacher-scheduling');
        } else if (user.role == 'admin') {
          Navigator.pushReplacementNamed(context, '/admin-scheduling');
        }
      } else {
        setState(() {
          _errorMessage = 'Credenciales inválidas';
          _isLoading = false;
        });
      }
    });
  }
}
```

Fig. 24: Login Page

Dashboard del Docente:

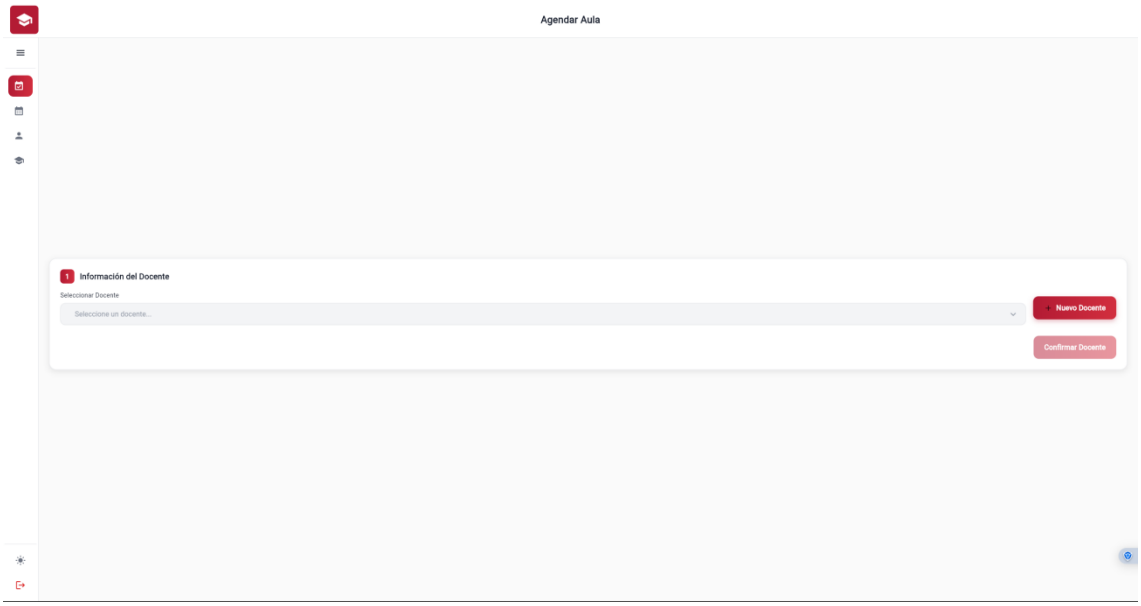


Fig. 25: Panel principal del docente con vista de agendamiento

El dashboard del docente proporciona acceso a tres módulos principales:

- Agendar: Selector de aulas y calendario semanal para crear reservas
- Agendas: Lista de reservas activas con opciones de edición y cancelación
- Perfil: Información personal y solicitud de capacitación

Dashboard del Administrador:

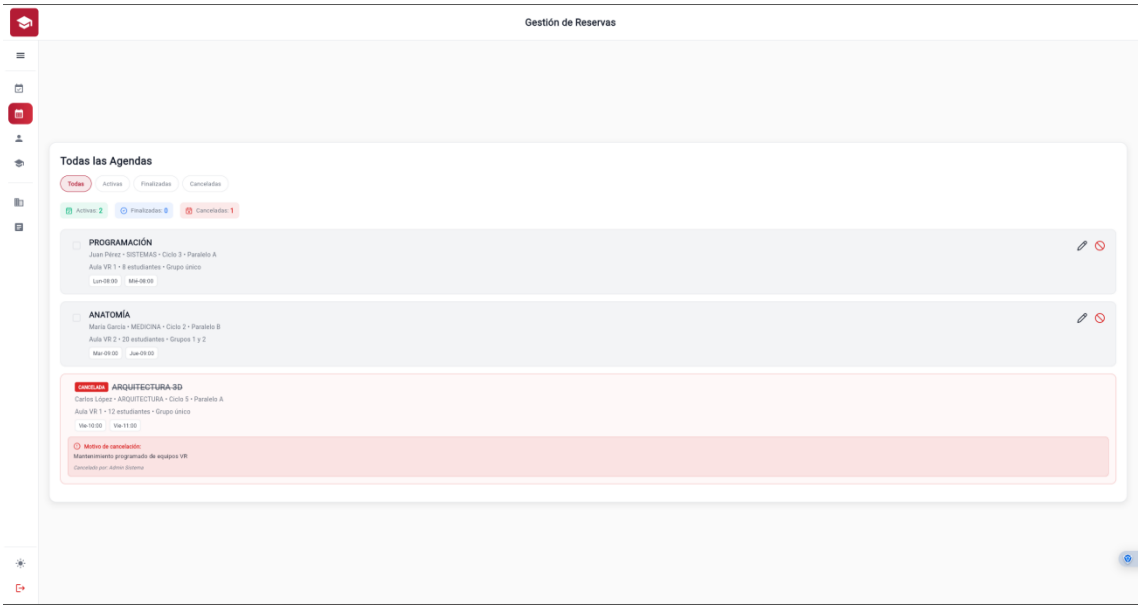


Fig. 26: Panel de administración con menú expandido

El panel de administración extiende el dashboard del docente con módulos adicionales:

- Gestión de Aulas: CRUD completo de aulas con selector de ubicación en mapa

- Capacitaciones: Gestión de solicitudes (pendientes, agendadas, capacitados)
- Editor Landing Page: Personalización de contenido del sitio público

```
class DashboardLayout extends StatefulWidget {
  final String pageTitle;
  final Widget child;
  final DashboardView currentView;
  final ValueChanged<DashboardView> onViewChange;
  final bool isAdmin;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Column(
        children: [
          _buildTopBar(isDark, t),
          Expanded(
            child: Stack(
              children: [
                Row(
                  children: [
                    AnimatedContainer(
                      duration: const Duration(milliseconds: 200),
                      width: isMobile ? 0 : (_sidebarExpanded ? 240 : 80),
                    ),
                    Expanded(
                      child: SingleChildScrollView(
                        padding: const EdgeInsets.all(24),
                        child: widget.child,
                      ),
                    ),
                  ],
                ),
                _buildSidebar(isDark, t, isMobile),
              ],
            ),
          ),
        ],
      ),
      bottomNavigationBar: isMobile ? _buildMobileNav(isDark, t) : null,
    );
  }
}
```

Fig. 27: Dashboard Layout

4.3.6. Widgets reutilizables

El sistema implementa un conjunto de widgets personalizados que mantienen consistencia visual:

AppCard - Tarjeta con soporte de temas:

```
class AppCard extends StatelessWidget {
  final Widget child;
  final EdgeInsetsGeometry? padding;
  final double borderRadius;
  final bool hasBorder;
  final bool hasShadow;
  final VoidCallback? onTap;
  final bool isSelected;
  @override
  Widget build(BuildContext context) {
    final isDark = Theme.of(context).brightness == Brightness.dark;

    return Container(
      decoration: BoxDecoration(
        color: isDark ? AppColors.darkCard : AppColors.lightCard,
        borderRadius: BorderRadius.circular(borderRadius),
        border: hasBorder || isSelected
            ? Border.all(
                color: isSelected
                    ? AppColors.ucRed
                    : (isDark ? AppColors.darkBorder : AppColors.lightBorder),
                width: isSelected ? 2 : 1,
              )
            : null,
        boxShadow: hasShadow ? [
          BoxShadow(
            color: Colors.black.withOpacity(isDark ? 0.3 : 0.08),
            blurRadius: 10,
            offset: const Offset(0, 4),
          ),
        ] : null,
      ),
      child: ClipRRect(
        borderRadius: BorderRadius.circular(borderRadius),
        child: Padding(
          padding: padding ?? const EdgeInsets.all(16),
          child: child,
        ),
      ),
    );
  }
}
```

Fig. 28: App Card

AppModal - Modal reutilizable:

```
class AppModal extends StatelessWidget {
  final String? title;
  final String? subtitle;
  final Widget child;
  final List<Widget>? actions;
  final double maxWidth;
  final bool showCloseButton;
  static Future<T?> show<T>({
    required BuildContext context,
    required Widget child,
    String? title,
    List<Widget>? actions,
    double maxWidth = 500,
  }) {
    return showDialog<T>(
      context: context,
      barrierColor: Colors.black.withOpacity(0.5),
      builder: (context) => AppModal(
        title: title,
        actions: actions,
        maxWidth: maxWidth,
        child: child,
      ),
    );
  }
```

Fig. 29: App Modal

4.4. Integración de servicios

La integración entre el frontend y el backend se realiza mediante llamadas HTTP a la API REST. Las solicitudes incluyen:

- Headers: Content-Type: application/json, Authorization: Bearer {token}
- Body: Datos en formato JSON
- Responses: Objetos JSON con la estructura definida

4.4.1 Configuración de CORS

Para permitir que el frontend (desplegado en Firebase Hosting) se comunique con el backend (desplegado en Render), se configuro CORS en el servidor Flask:

```
from flask_cors import CORS
CORS(app, resources={
    r"/api/*": {
        "origins": "*",
        "methods": ["GET", "POST", "PUT",
"DELETE", "OPTIONS"],
        "allow_headers": ["Content-Type",
"Authorization"]
    }
})
```

Fig. 30: CORS

4.4.2 Flujo de comunicación

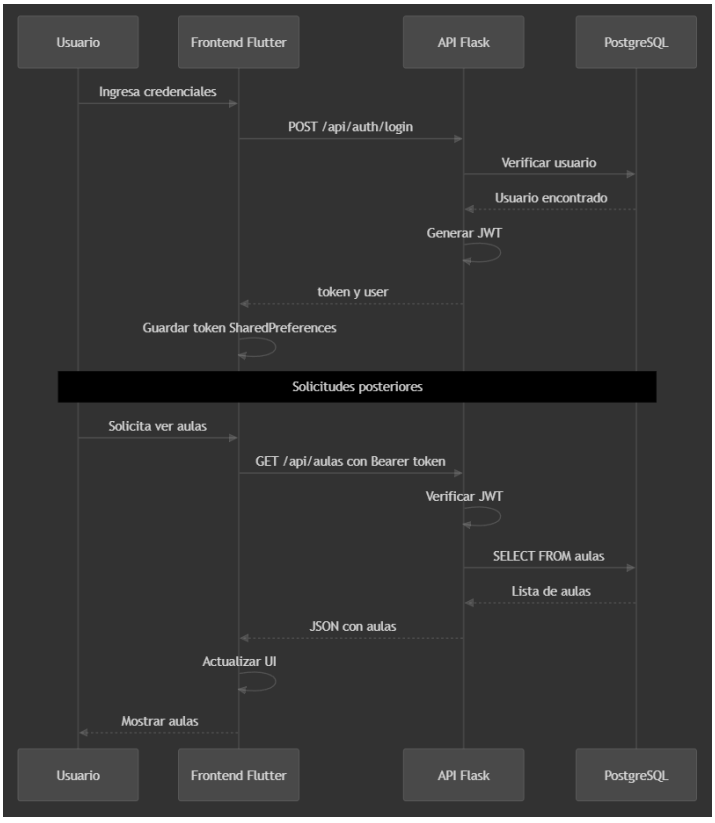


Fig. 31: Diagrama de Comunicacion

4.5. Configuración del despliegue

4.5.1 Configuración de Neon PostgreSQL

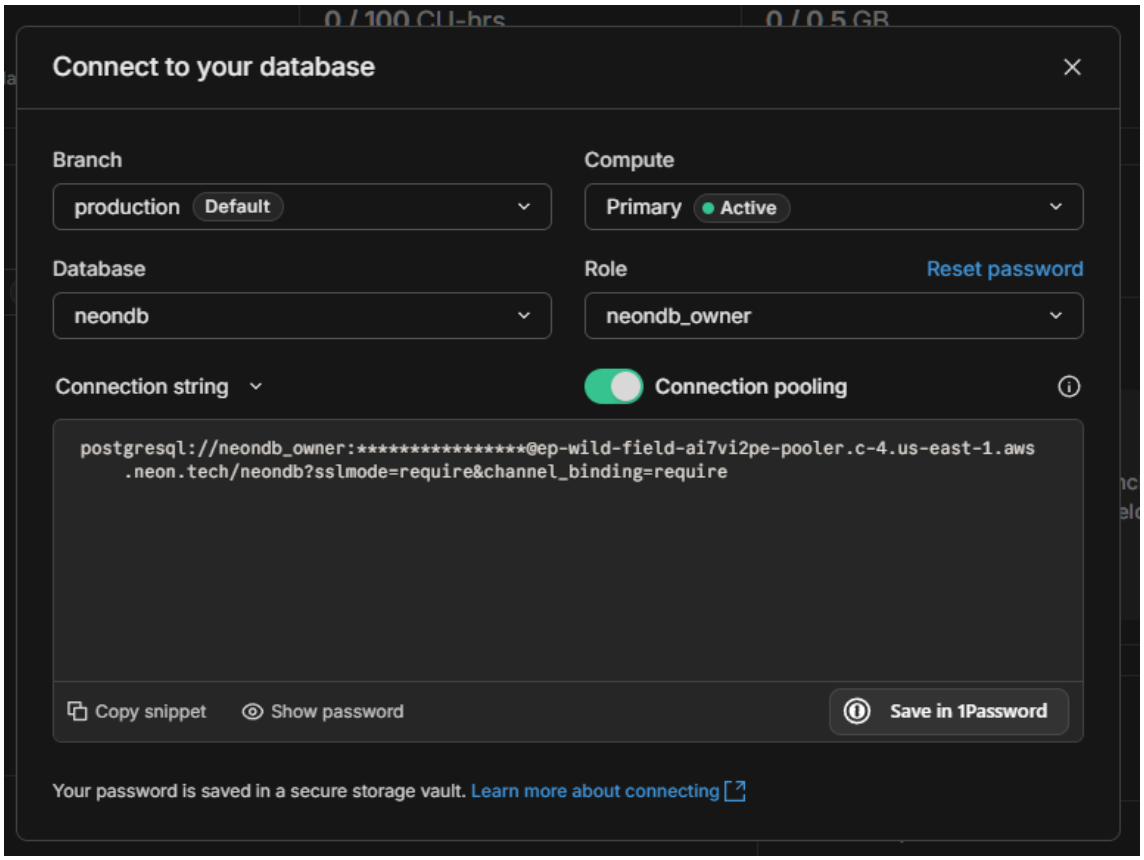


Fig. 32. Configuración del proyecto en Neon PostgreSQL

Pasos de configuración:

- Crear cuenta en neon.tech
- Crear nuevo proyecto
- Copiar connection string
- Configurar en variables de entorno del backend

4.5.2 Configuración de Render

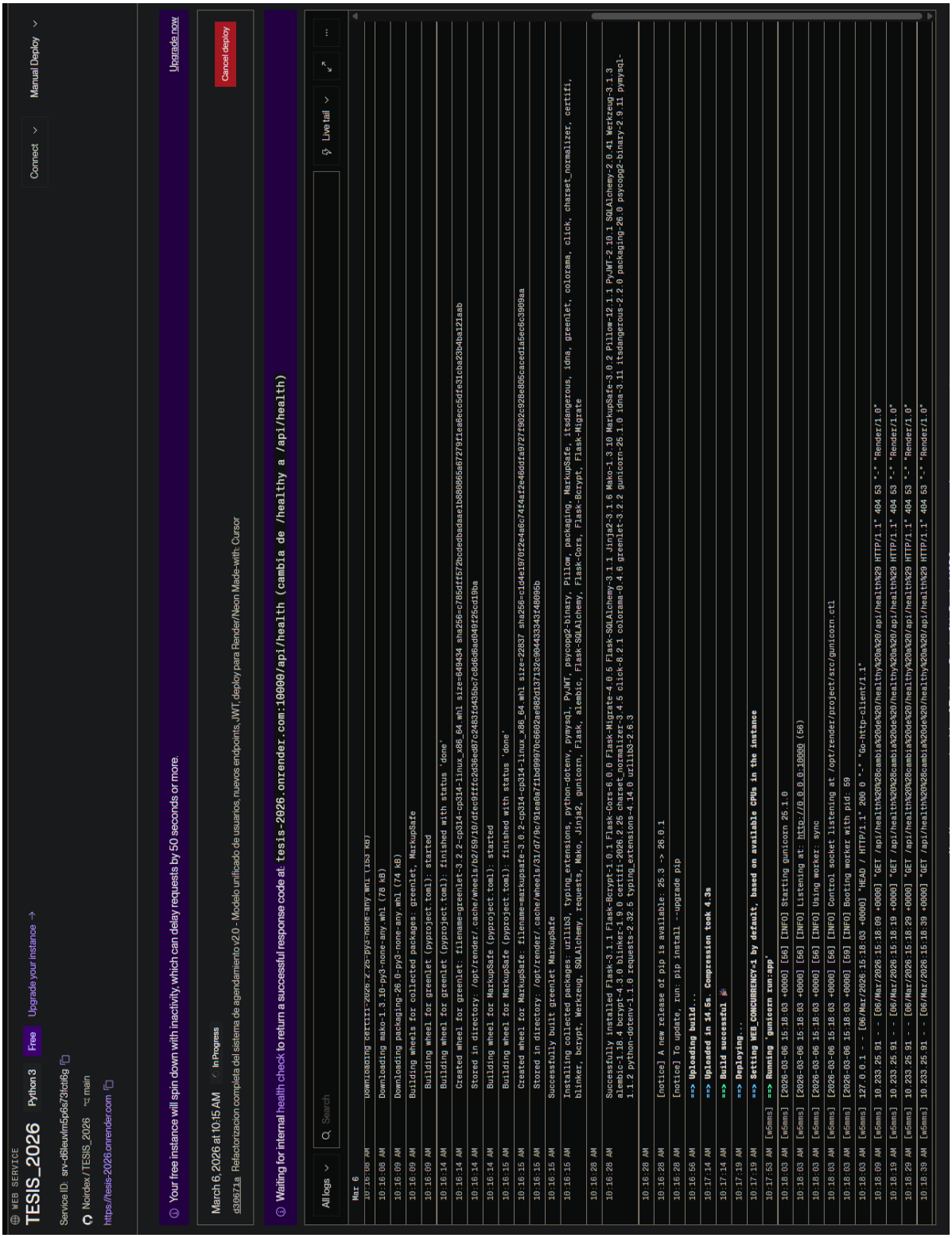


Fig. 33: Configuración del despliegue en Render

Variables de entorno configuradas:

- DATABASE_URL: Connection string de Neon
- SECRET_KEY: Clave secreta de Flask
- JWT_SECRET_KEY: Clave para firmar tokens JWT
- PYTHON_VERSION: 3.11.0

4.5.3 Configuración de Firebas

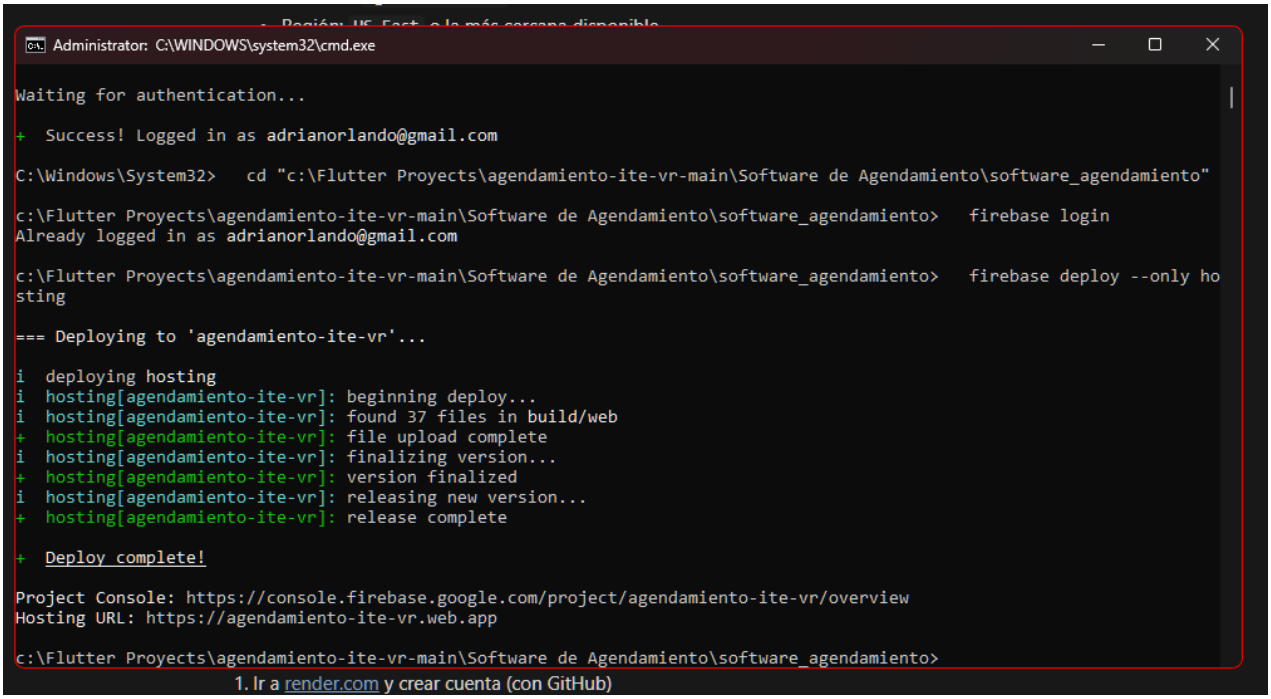


Fig. 34: Configuración de Firebase por CMD

La aplicación frontend desarrollada en Flutter Web se despliega utilizando Firebase Hosting, el servicio de hosting de Google que proporciona entrega rápida de contenido estático a través de su red CDN global. La configuración del proyecto en Firebase requiere los siguientes pasos:

- Se accede a la consola de Firebase (https://console.firebase.google.com) y se crea un nuevo proyecto denominado "agendamiento-ite-vr"
- Se habilita Google Analytics para el proyecto (opcional pero recomendado para métricas de uso)
- Se registra la aplicación web desde la sección "Agregar app" seleccionando el ícono web (</>)

Para gestionar el despliegue desde la línea de comandos, se instala Firebase CLI:

“npm install -g firebase-tools

firebase login ”

Desde el directorio raíz del proyecto Flutter, se ejecuta:

“firebase init hosting”

Durante la inicialización se seleccionan las siguientes opciones:

- Directorio público: build/web
- Configurar como SPA: Sí
- Sobrescribir index.html: No

```
{
  "hosting": {
    "public": "build/web",
    "ignore": [
      "firebase.json",
      "**/.*",
      "**/node_modules/**"
    ],
    "rewrites": [
      {
        "source": "**",
        "destination": "/index.html"
      }
    ],
    "headers": [
      {
        "source": "**/*.@(js|css|woff2|woff|ttf)",
        "headers": [
          {
            "key": "Cache-Control",
            "value": "max-age=31536000"
          }
        ]
      },
      {
        "source": "**",
        "headers": [
          {
            "key": "X-Frame-Options",
            "value": "DENY"
          },
          {
            "key": "X-Content-Type-Options",
            "value": "nosniff"
          }
        ]
      }
    ]
  }
}
```

Fig. 35: Configuración Firebase

Para diferenciar entre entornos de desarrollo y producción, se implementa una clase de configuración:

```
// lib/core/config/environment.dart
class Environment {
  static const String _devApiUrl =
'http://localhost:5000/api';
  static const String _prodApiUrl =
'https://agendamiento-vr-api.onrender.com/api';

  static bool get isProduction =>
      const String.fromEnvironment('ENV',
defaultValue: 'dev') == 'prod';

  static String get apiBaseUrl => isProduction ?
_prodApiUrl : _devApiUrl;
}
```

Fig. 36: Configuración .env

La compilación para producción se realiza con:

"flutter build web --release --dart-define=ENV=prod"

Despliegue a Firebase Hosting:

"firebase deploy --only hosting"

Parámetro	Valor
Proyecto ID	agendamiento-ite-vr
URL de producción	https://agendamiento-ite-vr.web.app
URL alternativa	https://agendamiento-ite-vr.firebaseapp.com
Región CDN	Global (automático)
SSL	Incluido (Let's Encrypt)
Directorio de build	build/web

Tabla 9: Configuración de Firebase

Firebase Hosting permite configurar dominios personalizados desde la consola. Para vincular un dominio institucional:

- Acceder a Hosting > Agregar dominio personalizado
- Verificar propiedad del dominio mediante registro DNS TXT
- Configurar registros A apuntando a las IPs de Firebase
- Esperar la provisión del certificado SSL (automático)

CAPÍTULO 5

RESULTADOS

5.1. Pruebas del sistema

5.1.1 Pruebas de endpoints (Backend)

Endpoint	Método	Caso de prueba	Resultado
/api/auth/register	POST	Registro con email válido	Exitoso (201)
/api/auth/register	POST	Registro con email duplicado	Error esperado (409)
/api/auth/login	POST	Login con credenciales válidas	Exitoso (200) + Token
/api/auth/login	POST	Login con credenciales inválidas	Error esperado (401)
/api/aulas	GET	Listar aulas sin autenticación	Exitoso (200)
/api/aulas	POST	Crear aula sin token	Error esperado (401)
/api/aulas	POST	Crear aula como admin	Exitoso (201)
/api/agendas	POST	Crear agenda sin capacitación	Error esperado (403)
/api/agendas	POST	Crear agenda con capacitación	Exitoso (201)
/api/agendas	POST	Crear agenda en horario ocupado	Error esperado (409)

Tabla 10: Resultados de pruebas de endpoints principales

5.1.2 Pruebas funcionales (frontend)

Pantalla	Caso de prueba	Resultado esperado	Resultado obtenido
Landing Page	Carga inicial	Renderizado completo en <3s	Exitoso
Landing Page	Carrusel automático	Cambio de slide cada 4s	Exitoso
Landing Page	Cambio de tema	Toggle claro/oscuro	Exitoso
Landing Page	Cambio de idioma	ES/EN dinámico	Exitoso
Landing Page	Responsive	Adaptación mobile/desktop	Exitoso
Login	Campos vacíos	Mensaje de error	Exitoso
Login	Credenciales válidas (docente)	Redirección a dashboard docente	Exitoso
Login	Credenciales válidas (admin)	Redirección a dashboard admin	Exitoso
Login	Credenciales inválidas	Mensaje "Credenciales inválidas"	Exitoso
Dashboard	Sidebar expandible	Animación de 200ms	Exitoso
Dashboard	Navegación entre vistas	Cambio de contenido sin recarga	Exitoso
Dashboard	Logout	Retorno a página de login	Exitoso

Agendar	Selección de aula	Muestra calendario del aula	Exitoso
Agendar	Selección de horario libre	Abre modal de reserva	Exitoso
Agendar	Selección de horario ocupado	Muestra detalles de reserva	Exitoso
Agendas	Lista de reservas	Muestra cards con información	Exitoso
Agendas	Cancelar reserva	Modal de confirmación + actualización	Exitoso
Admin - Aulas	Crear aula con mapa	Selector de ubicación funcional	Exitoso
Admin - Aulas	Subir imagen	Image picker + preview	Exitoso
Admin - Landing	Editar características	Icon picker visual	Exitoso
Admin - Landing	Editar carrusel	Image picker desde dispositivo	Exitoso

Tabla 11: Resultados de pruebas funcionales del frontend

5.2. Capturas de pantalla del sistema

5.2.1 Landing page

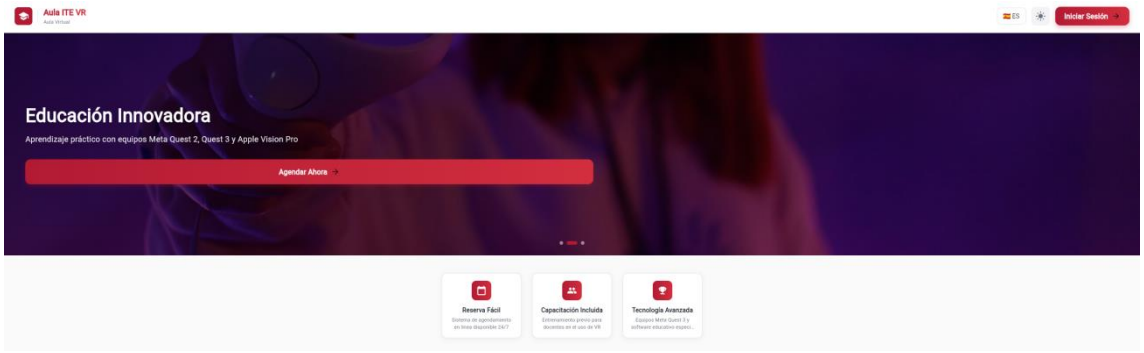


Fig. 37: Landing page Carrusel

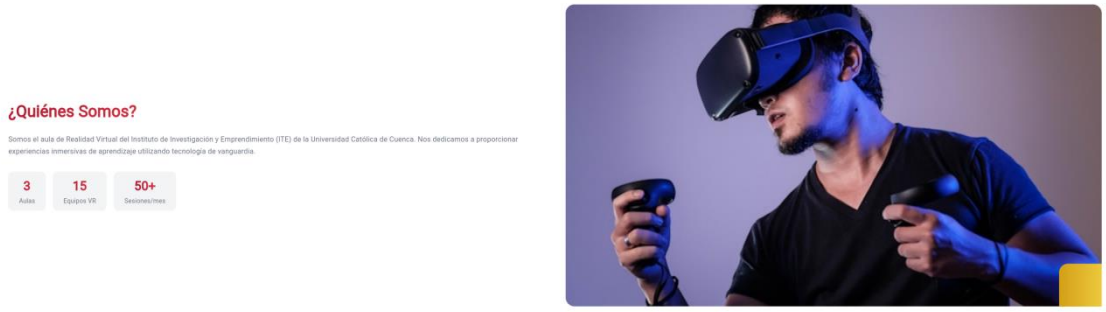


Fig. 38: Landing page Info 1

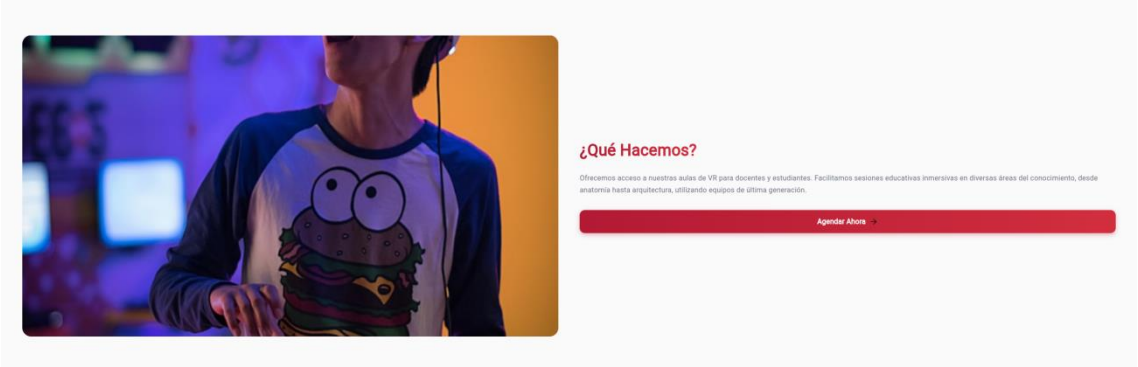


Fig. 39: Landing page Info 2

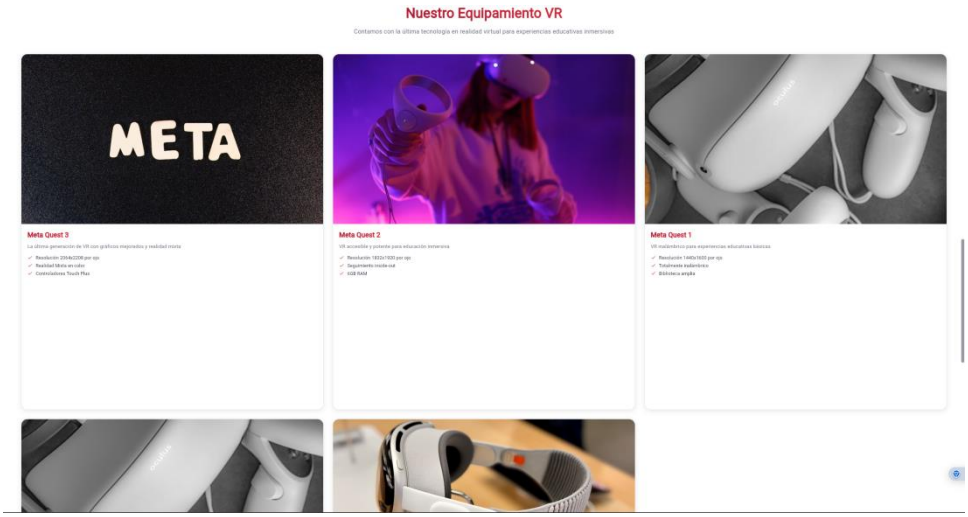


Fig. 40: Landing page Equipos

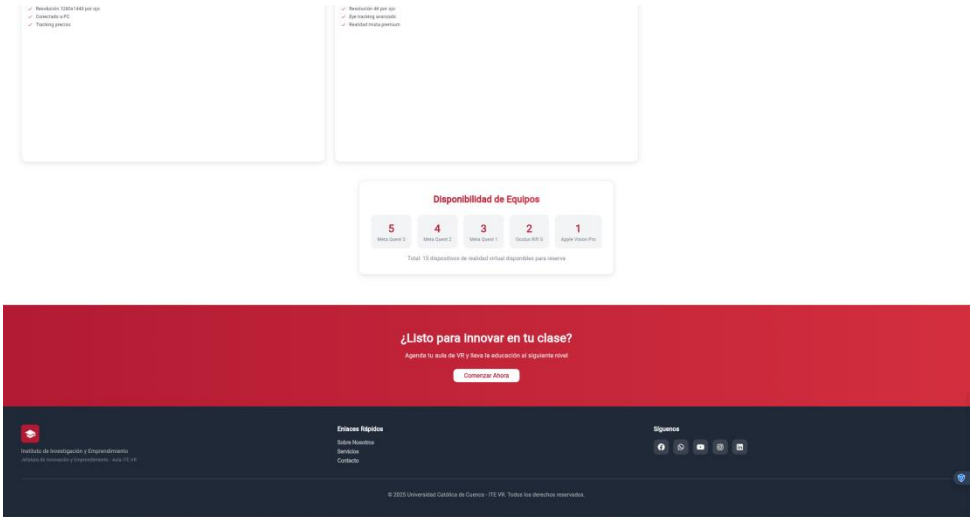


Fig. 41: Landing page Info 3

Incluye capturas de:

- Hero section con carrusel

- Sección de características
- Sección "Acerca de"
- Información de contacto
- Footer

5.2.2 Módulo de autenticación

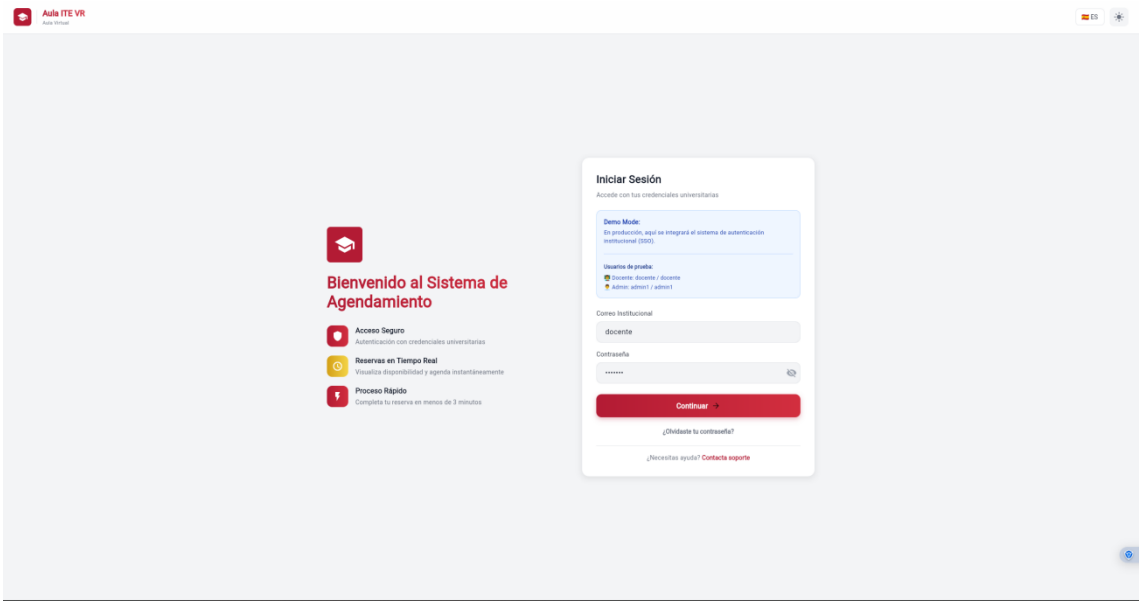


Fig. 42: Login Page

Incluye capturas de:

- Formulario de login
- Login de administrador

5.2.3 Panel de docentes

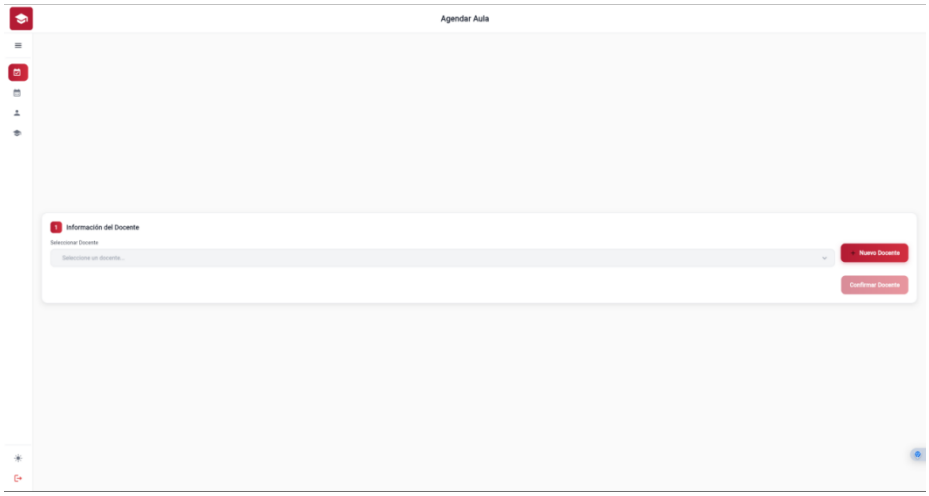


Fig. 43: Agendamiento de Aulas Docente 1

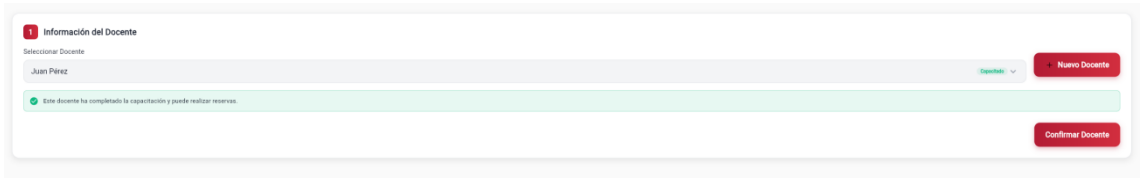


Fig. 44: Agendamiento de Aulas Docente 2

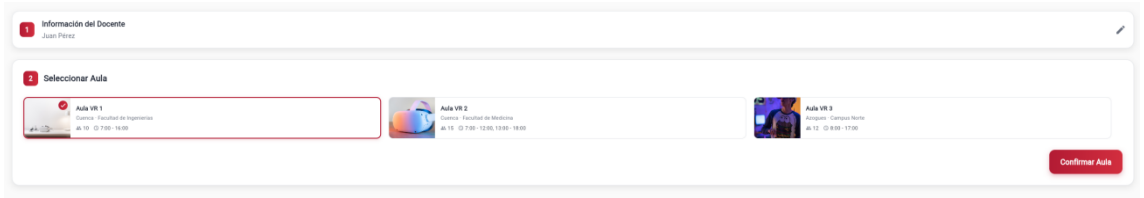


Fig. 45: Agendamiento de Aulas Docente 3

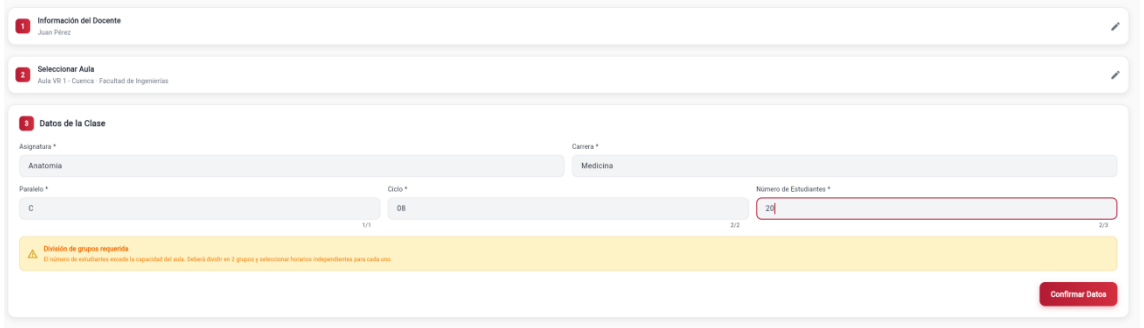


Fig. 46: Agendamiento de Aulas Docente 4

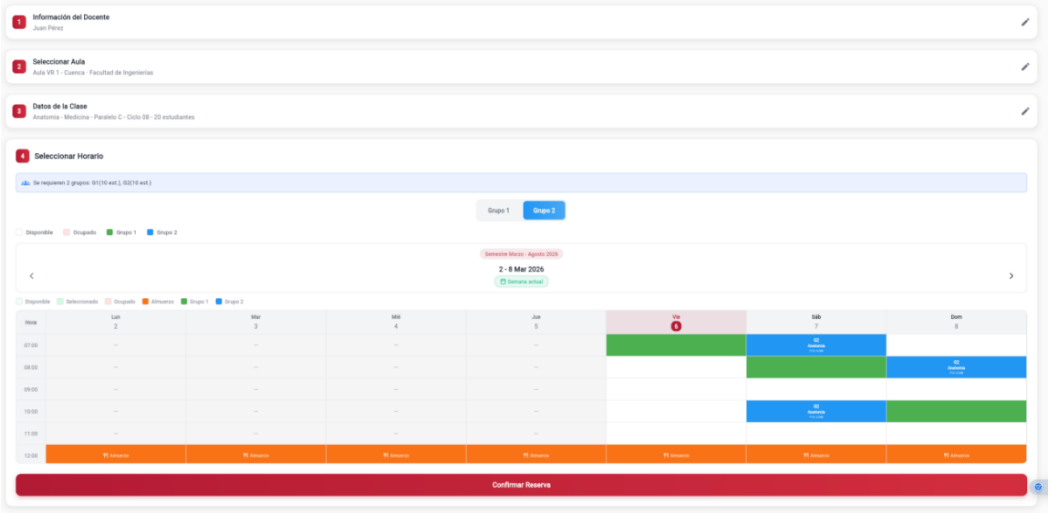


Fig. 47: Agendamiento de Aulas Docente 5

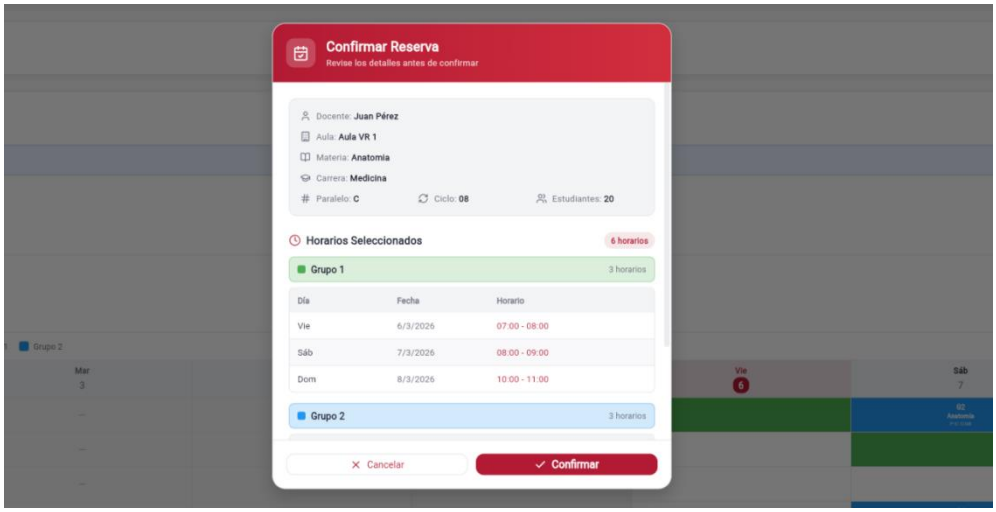


Fig. 48: Agendamiento de Aulas Docente 6

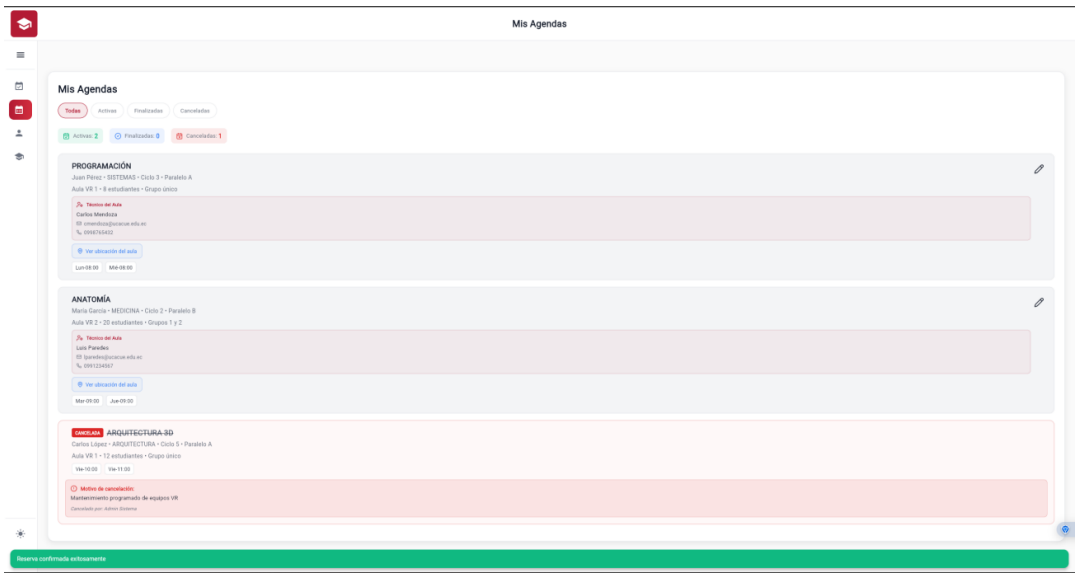


Fig. 49: Agendas Docente

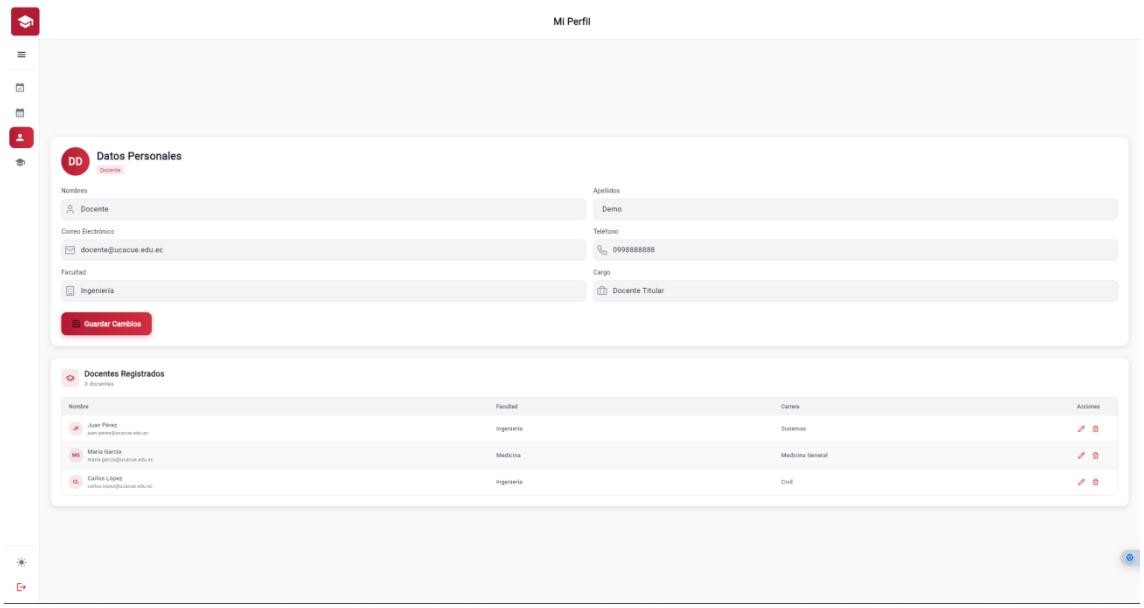


Fig. 50: Datos Personales Docente

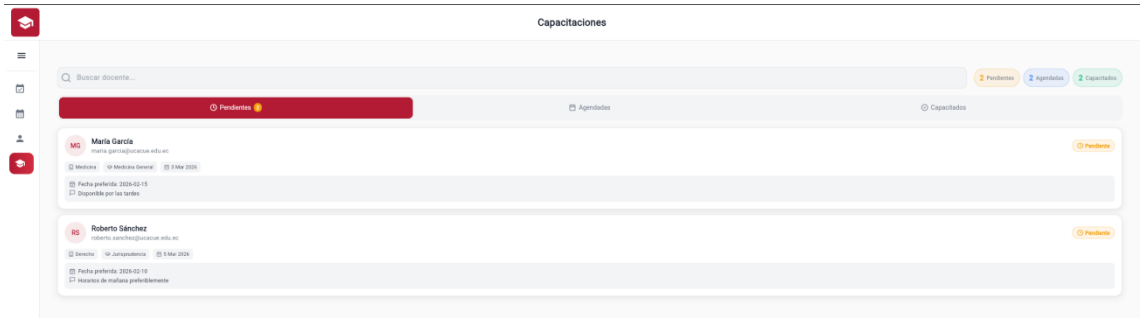


Fig. 51: Capacitaciones Docente

Incluye capturas de:

- Dashboard principal
- Vista de aulas disponibles
- Calendario de agendamiento
- Formulario de nueva reserva
- Lista de mis reservas

5.2.4 Panel de administración

Fig. 52: Agendas Administrador

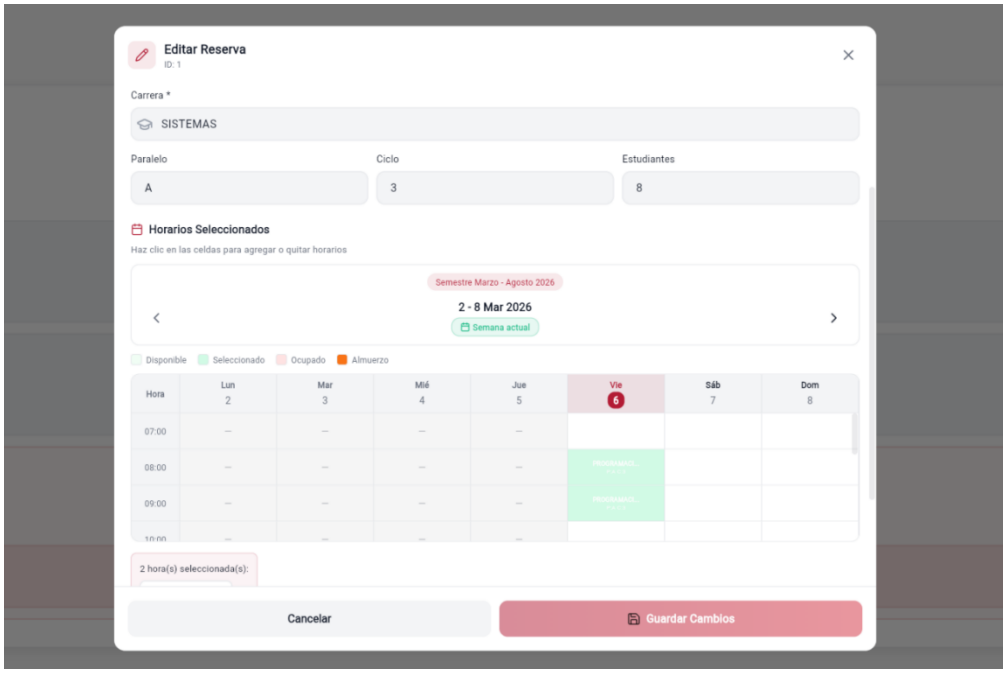


Fig. 53: Editar Agendas Administrador

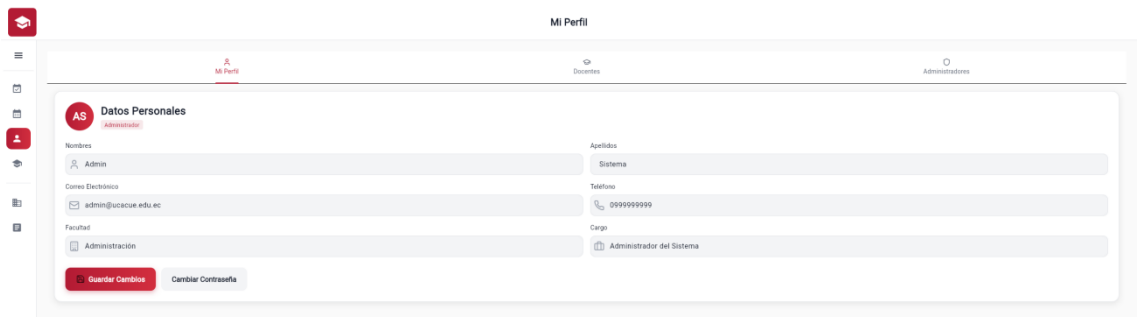


Fig. 54: Datos Personales Administrador

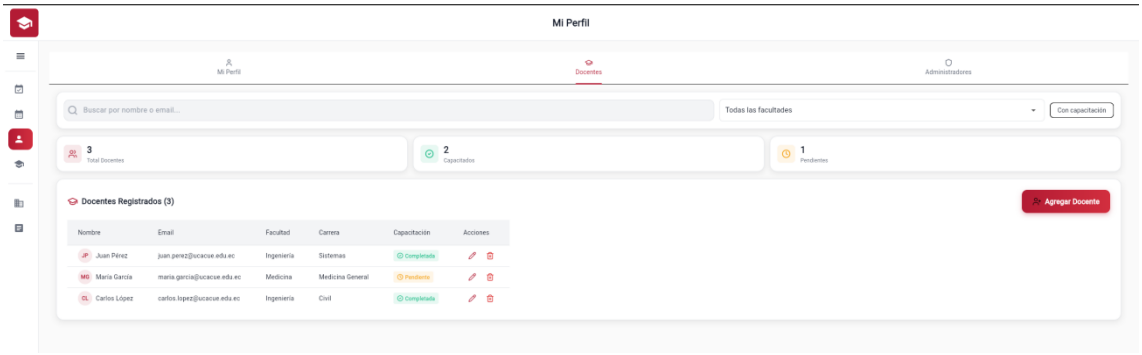


Fig. 55: Docentes Capacitados Administrador

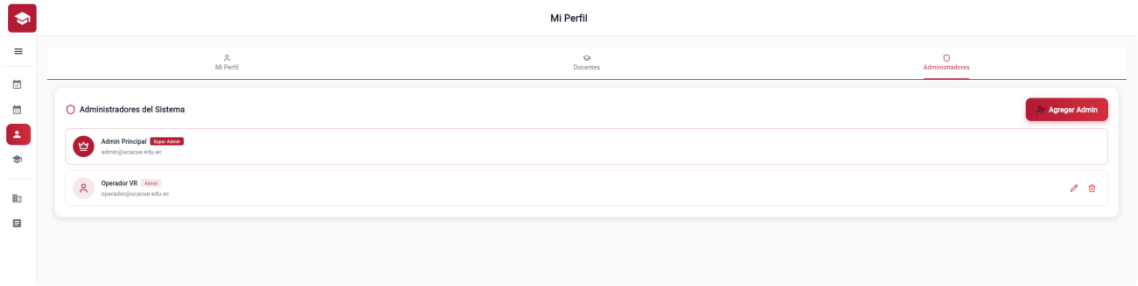


Fig. 56: Administradores Administrador

Fig. 57: Capacitaciones Administrador

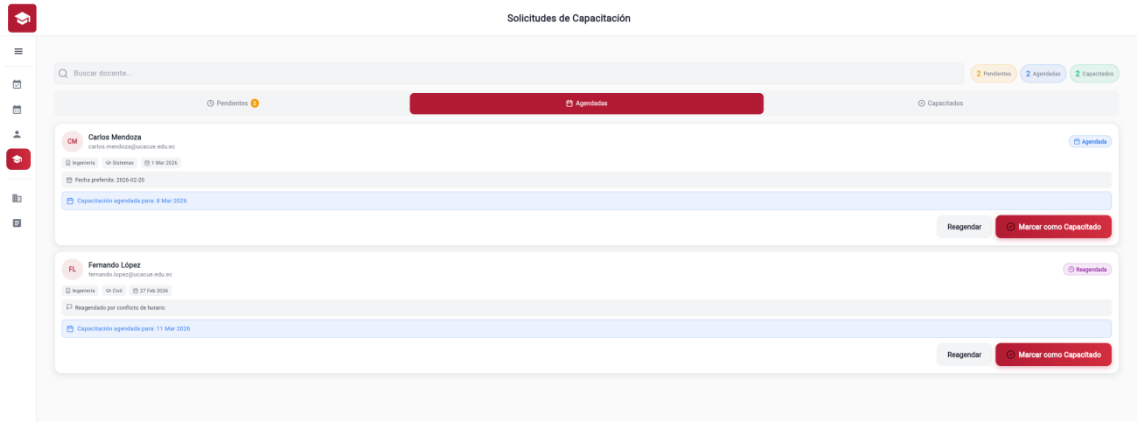


Fig. 58: Capacitaciones Agendadas Administrador

Fig. 59: Capacitaciones Finalizadas Administrador

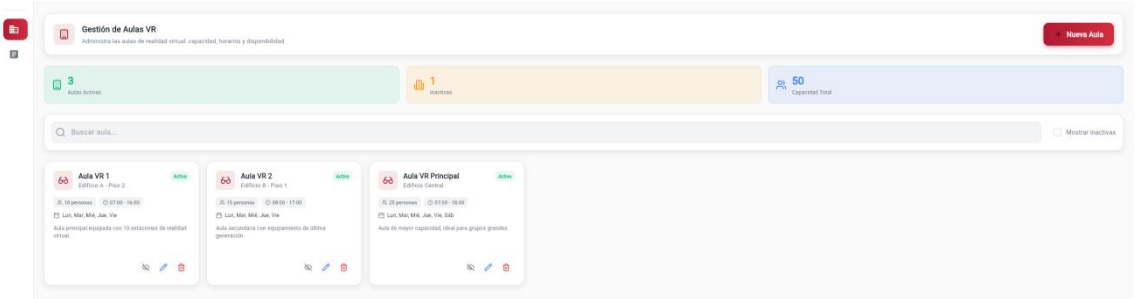


Fig. 60: Aulas Administrador

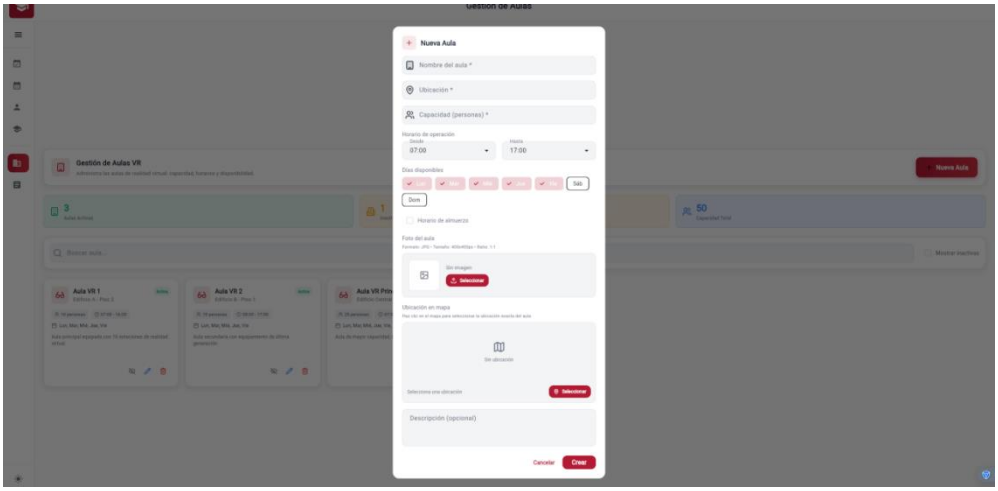


Fig. 61: Crear Aulas Administrador

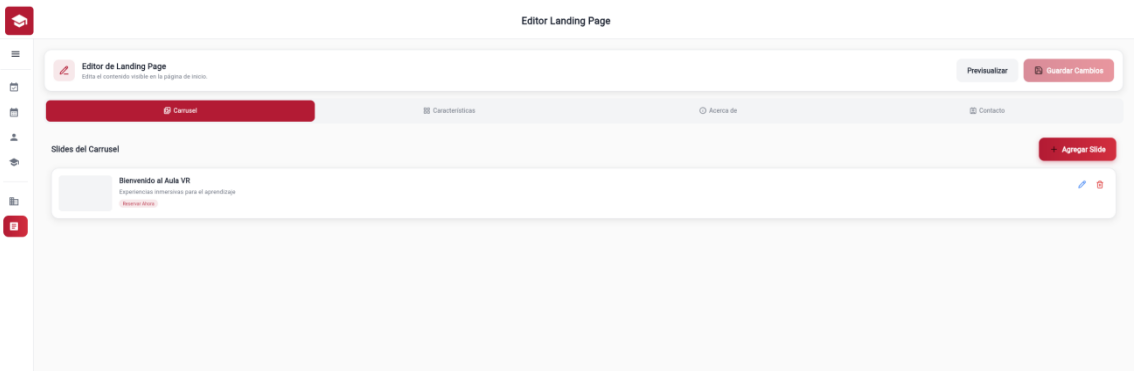


Fig. 62: Landing Page Carrusel Editor Administrador

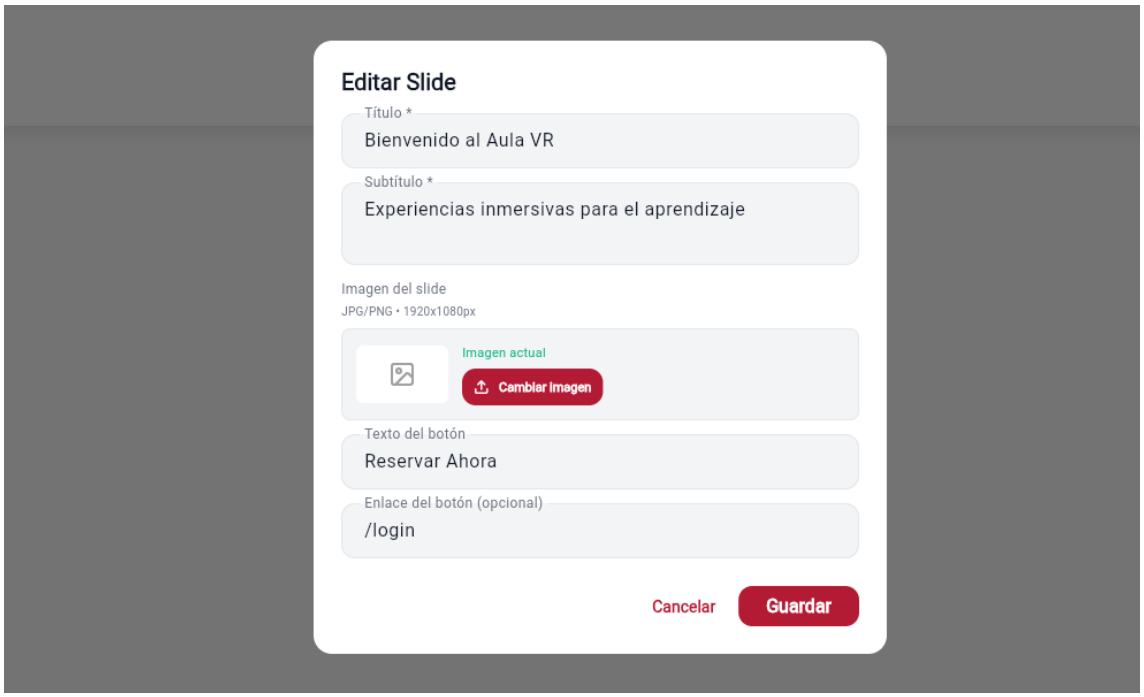


Fig. 63: Landing Page Slide Editor Administrador

Incluye capturas de:

- Dashboard de administración
- Gestión de usuarios
- Gestión de aulas
- Gestión de contenido de landing page

5.2.5 Módulo de capacitaciones

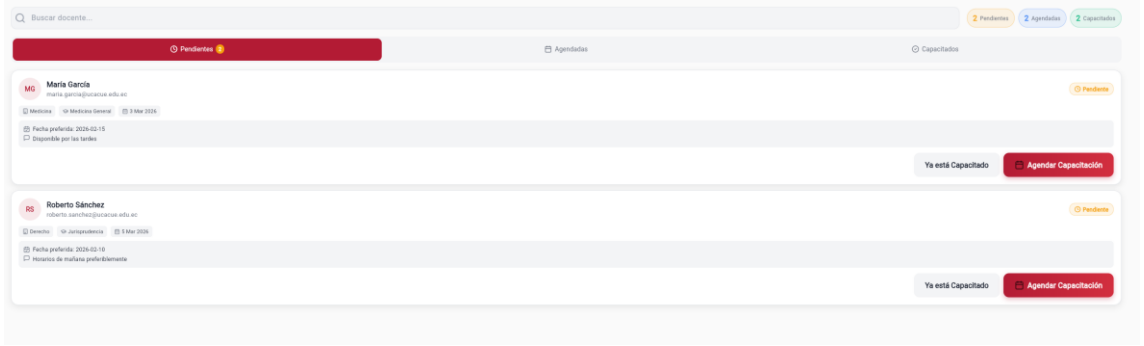


Fig. 64: Módulo de Capacitaciones

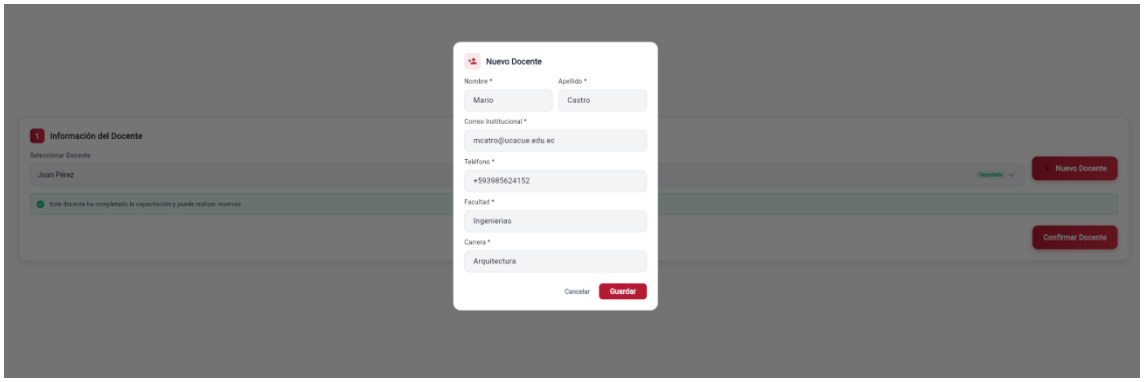


Fig. 65: Modal Creación Docente

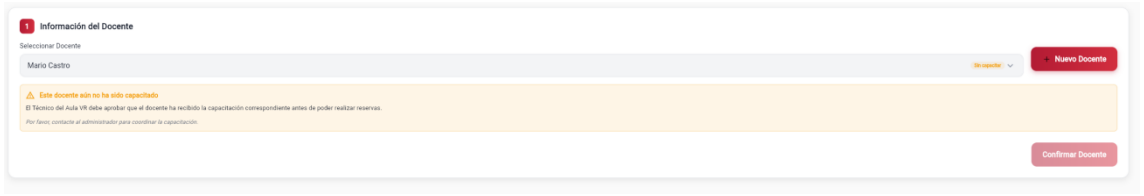


Fig. 66: Notificación del estado del Docente

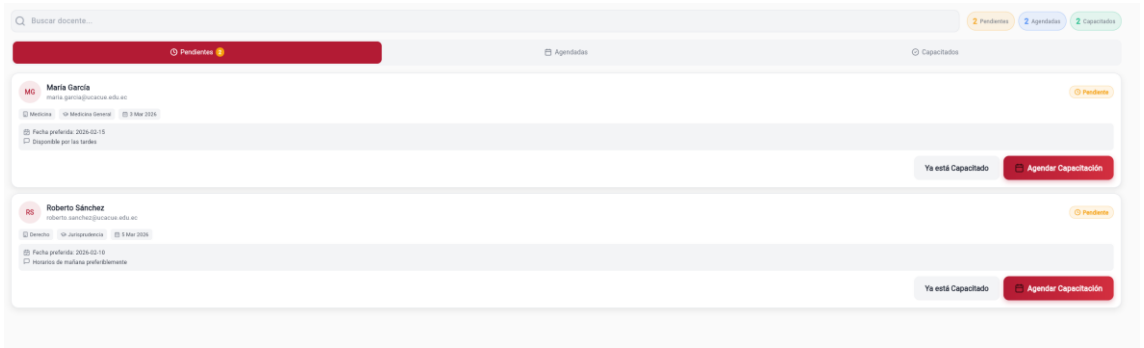


Fig. 67: Módulo de Capacitaciones, Sección de Agendas

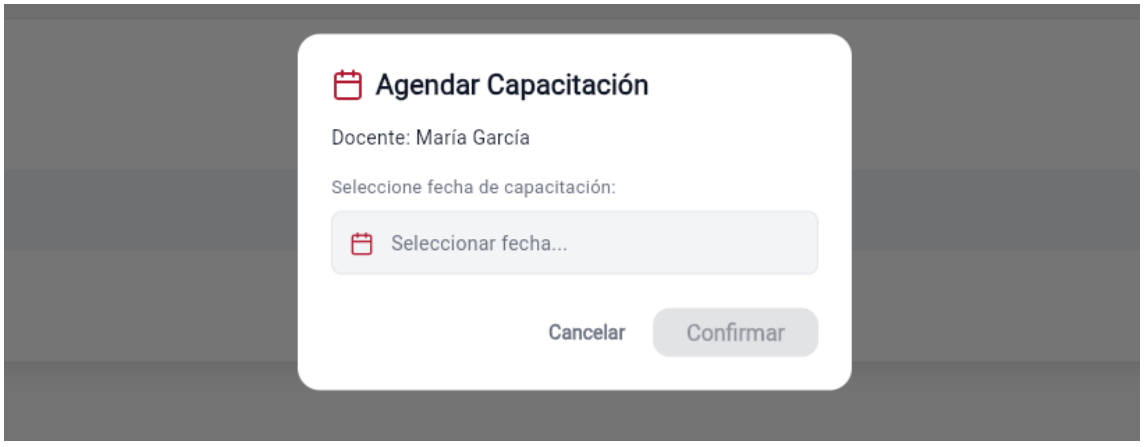


Fig. 68: Módulo de Capacitaciones, Agendar Capacitación

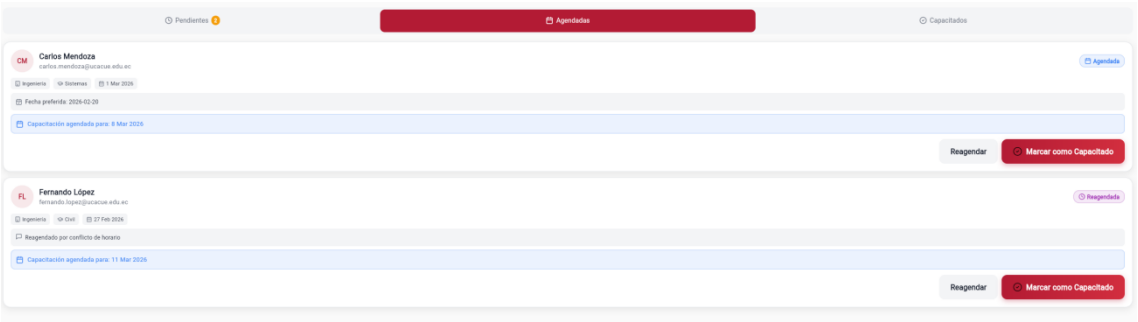


Fig. 69: Módulo de Capacitaciones, Agendadas

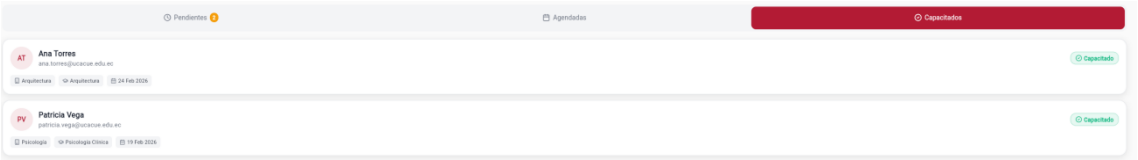


Fig. 70: Módulo de Capacitaciones, Finalizadas

Incluye capturas de:

- Formulario de solicitud de capacitación
- Lista de solicitudes (admin)
- Proceso de aprobación

5.3. Métricas de rendimiento

Las siguientes métricas fueron obtenidas utilizando las herramientas de desarrollo de Chrome (DevTools) y la extensión Lighthouse, ejecutando pruebas sobre la aplicación desplegada en producción. Las mediciones se realizaron en un equipo con procesador Intel Core i5, 8Gb de RAM y una conexión a internet de 10Mbps, promediando los resultados de tres ejecuciones consecutivas.

Métrica	Valor
Tiempo de carga inicial (First Contentful Paint)	1.8 segundos
Tiempo interactivo (Time to Interactive)	2.5 segundos
Tamaño del bundle de producción (CanvasKit)	~4.2 MB (comprimido: ~1.5 MB)
Tamaño del bundle de producción (HTML)	~2.8 MB (comprimido: ~900 KB)
Tiempo de navegación entre pantallas	< 100 ms
Tiempo de hot reload en desarrollo	< 1 segundo
Consumo de memoria en navegador	~85-120 MB

Tabla 12: Métricas de rendimiento del frontend

CAPÍTULO 6

CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

Se logro desarrollar un sistema web funcional para la gestión y agendamiento de aulas de realidad virtual que satisface los objetivos planteados en el presente trabajo de titulación. El backend implementado en Flask proporciona una API RESTful robusta con 67 endpoints, mientras que el frontend en Flutter Web ofrece una interfaz moderna y responsiva que consume estos servicios de manera eficiente.

1. La arquitectura cliente-servidor adoptada permite una separación clara de responsabilidades, facilitando el mantenimiento independiente de cada componente y posibilitando futuras expansiones hacia aplicaciones móviles nativas. La comunicación mediante API REST y JSON garantiza la interoperabilidad entre ambas capas del sistema.

2. El sistema de autentificación basado en JWT con tres niveles de roles (docente, admin, superAdmin) proporciona un control de acceso que responde a los requisitos de seguridad identificados durante el análisis. La implementación del patrón Provider en el frontend permite gestionar el estado de autenticación de manera reactiva, propagando los cambios a través de toda la jerarquía de widgets.

3. El módulo de capacitaciones implementado permite verificar que solo usuarios debidamente habilitados puedan realizar reservas de aulas VR, cumpliendo con los protocolos de seguridad establecidos para el uso de equipos especializados. El flujo de solicitud, aprobación y validación se refleja de manera coherente tanto en la API como en las interfaces de usuario.

4. La arquitectura modular basada en características (feature-based) tanto en el backend (blueprints de Flask) como en el frontend (estructura por features de Flutter) proporcionó una organización clara y mantenible, separando responsabilidades y facilitando la incorporación de nuevas funcionalidades.

5. El sistema de widgets declarativos de Flutter facilitó la creación de componentes reutilizables que mantienen consistencia visual en toda la aplicación, reduciendo el tiempo de desarrollo y la duplicación de código. La persistencia de preferencias mediante SharedPreferences garantiza una experiencia de usuario personalizada entre sesiones.

6. El despliegue en servicios cloud gratuitos (Render para el backend, Neon para PostgreSQL y Firebase para el frontend y almacenamiento) demuestra la viabilidad de implementar soluciones tecnológicas de calidad con recursos limitados, lo cual es especialmente relevante en el contexto de proyectos académicos.

6.2. Recomendaciones

1. Implementar un sistema de notificaciones por correo electrónico y push para informar a los usuarios sobre confirmaciones de reservas, recordatorios, cancelaciones y actualizaciones de estado de capacitaciones.
2. Considerar la implementación de Progressive Web App (PWA) para mejorar la experiencia offline y permitir la instalación de la aplicación en dispositivos móviles sin necesidad de publicar en tiendas de aplicaciones.
3. Desarrollar aplicaciones móviles nativas (iOS y Android) aprovechando la API REST existente y reutilizando la mayor parte del código Flutter para ampliar la accesibilidad del sistema.
4. Implementar caching de datos tanto en el servidor (Redis) como en el cliente (Hive o Isar) para reducir las llamadas a la API y mejorar la velocidad de respuesta, especialmente en conexiones lentas.
5. Integrar el sistema con la plataforma académica institucional (SGA) para sincronizar información de docentes, carreras y materias de manera automática.
6. Evaluar la migración a una solución de gestión de estado más robusta como Riverpod o BLoC cuando la complejidad del proyecto aumente, particularmente si se requiere manejo de estados asíncronos más complejos o testing unitario exhaustivo de la lógica de negocio.
7. Considerar la migración a servicios cloud de pago cuando el sistema escale, para garantizar mejor rendimiento, disponibilidad y soporte técnico profesional.

6.3. Trabajos futuros

- Sistema de reservas recurrentes con gestión automática de conflictos.
- Integración con calendarios externos (Google Calendar, Outlook).
- Módulo de reportes exportables en múltiples formatos (PDF, Excel, CSV).
- Sistema de evaluación de satisfacción post-uso con métricas de calidad.
- Implementación completa de accesibilidad web (WCAG 2.1).
- Dashboard de analítica avanzada con visualización de patrones de uso.

En conclusión, el desarrollo del sistema web permitirá automatizar el proceso de reserva de aulas de realidad virtual en la Universidad Católica de Cuenca, el cual reducirá significativamente los conflictos de horarios y facilitará la planificación del uso de estos recursos tecnológicos. La implementación de una arquitectura basada en API REST y autenticación mediante JWT garantiza la escalabilidad y seguridad del sistema.

REFERENCIAS

[1] J. Martín-Gutiérrez, C. E. Mora, B. Añorbe-Díaz, y A. González-Marrero, "Virtual Technologies Trends in Education," EURASIA Journal of Mathematics, Science and Technology Education, vol. 13, no. 2, pp. 469-486, 2017.

[2] M. Bower, C. Howe, N. McCredie, A. Robinson, y D. Grover, "Augmented Reality in education – cases, places and potentials," Educational Media International, vol. 51, no. 1, pp. 1-15, 2014.

[3] A. Alharthi, M. Alassafi, R. J. Walters, y G. B. Wills, "An exploratory study for investigating the critical success factors for cloud migration in the Saudi Arabian higher education context," Telematics and Informatics, vol. 34, no. 2, pp. 664-678, 2017.

[4] K. Lee, "Augmented Reality in Education and Training," TechTrends, vol. 56, no. 2, pp. 13-21, 2012.

[5] P. Brusilovsky y C. Peylo, "Adaptive and Intelligent Web-based Educational Systems," International Journal of Artificial Intelligence in Education, vol. 13, pp. 159-172, 2003.

[6] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[7] L. Richardson y S. Ruby, RESTful Web Services. O'Reilly Media, 2007.

[8] J. Nielsen, "Usability Engineering," Morgan Kaufmann, 1994.

[9] SENESCYT, "Reglamento de Régimen Académico," Registro Oficial, Ecuador, 2019.

[10] IEEE, "Software Engineering Body of Knowledge (SWEBOK)," IEEE Computer Society, 2014.

[11] C. Laudon y J. Laudon, "Management Information Systems: Managing the Digital Firm," Pearson, 2020.

[12] R. Elmasri y S. B. Navathe, "Fundamentals of Database Systems," Pearson, 2016.

[13] M. Fowler, "Patterns of Enterprise Application Architecture," Addison-Wesley, 2002.

[14] T. Berners-Lee, R. Fielding, y H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0," RFC 1945, 1996.

[15] ISO/IEC, "ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE)," 2011.

[16] J. Radianti, T. A. Majchrzak, J. Fromm, y I. Wohlgenannt, "A systematic review of immersive virtual reality applications for higher education: Design elements, lessons learned, and research agenda," Computers & Education, vol. 147, 2020.

[17] G. Piccoli y B. Ives, "IT-Dependent Strategic Initiatives and Sustained Competitive Advantage: A Review and Synthesis of the Literature," MIS Quarterly, vol. 29, no. 4, pp. 747-776, 2005.

[18] A. S. Tanenbaum y M. Van Steen, "Distributed Systems: Principles and Paradigms," Pearson, 2017.

[19] S. Newman, "Building Microservices: Designing Fine-Grained Systems," O'Reilly Media, 2015.

[20] M. Masse, "REST API Design Rulebook," O'Reilly Media, 2011.

[21] R. T. Fielding y R. N. Taylor, "Principled Design of the Modern Web Architecture," ACM Transactions on Internet Technology, vol. 2, no. 2, pp. 115-150, 2002.

[22] L. Richardson, M. Amundsen, y S. Ruby, "RESTful Web APIs," O'Reilly Media, 2013.

[23] M. Jones, J. Bradley, y N. Sakimura, "JSON Web Token (JWT)," RFC 7519, 2015.

[24] A. Rahmatulloh, R. Gunawan, y F. M. S. Nursuwars, "Performance comparison of signed algorithms on JSON Web Token," IOP Conference Series: Materials Science and Engineering, vol. 550, 2019.

[25] OWASP, "Authentication Cheat Sheet," OWASP Foundation, 2023.

[26] M. Grinberg, "Flask Web Development: Developing Web Applications with Python," O'Reilly Media, 2018.

[27] Pallets Projects, "Flask Documentation," https://flask.palletsprojects.com/, 2024.

[28] M. Bayer, "SQLAlchemy Documentation," https://docs.sqlalchemy.org/, 2024.

[29] M. Grinberg, "Flask-Migrate Documentation," https://flask-migrate.readthedocs.io/, 2024.

[30] Bcrypt, "Bcrypt Password Hashing Documentation," https://pypi.org/project/bcrypt/, 2024.

[31] C. Leach, "Flask-CORS Documentation," https://flask-cors.readthedocs.io/, 2024.

[32] M. Bayer, "SQLAlchemy: The Database Toolkit for Python," O'Reilly Media, 2021.

[33] R. Copeland, "Essential SQLAlchemy: Mapping Python to Databases," O'Reilly Media, 2016.

[34] PostgreSQL Global Development Group, "PostgreSQL Documentation," https://www.postgresql.org/docs/, 2024.

[35] R. Obe y L. Hsu, "PostgreSQL: Up and Running," O'Reilly Media, 2017.

[36] C. J. Date, "An Introduction to Database Systems," Pearson, 2003.

[37] Render, "Render Documentation," https://render.com/docs, 2024.

[38] Neon, "Neon Documentation," https://neon.tech/docs, 2024.

[39] Google, "Firebase Documentation," https://firebase.google.com/docs, 2024.

[40] K. Schwaber y J. Sutherland, "The Scrum Guide," Scrum.org, 2020.

[41] M. Cohn, "Succeeding with Agile: Software Development Using Scrum," Addison-Wesley, 2010.

[42] J. Sutherland, "Scrum: The Art of Doing Twice the Work in Half the Time," Currency, 2014.

[43] R. Hernández-Sampieri, C. Fernández-Collado, y P. Baptista-Lucio, "Metodología de la Investigación," McGraw-Hill, 2014.

[44] J. W. Creswell, "Research Design: Qualitative, Quantitative, and Mixed Methods Approaches," SAGE Publications, 2014.

[45] K. S. Rubin, "Essential Scrum: A Practical Guide to the Most Popular Agile Process," Addison-Wesley, 2012.

[46] G. J. Myers, C. Sandler, y T. Badgett, "The Art of Software Testing," John Wiley & Sons, 2011.

[47] K. Morris, "Infrastructure as Code: Managing Servers in the Cloud," O'Reilly Media, 2016.

[48] Google, "Flutter Documentation," https://docs.flutter.dev/, accedido: enero 2026.

[49] E. Windmill y M. Biessek, "Flutter in Action," Manning Publications, 2020.

[50] Google, "Flutter Architectural Overview," https://docs.flutter.dev/resources/architectural-overview, accedido: enero 2026.

[51] R. Rappaport y S. Abramov, "Beginning Flutter: A Hands-On Guide to App Development," Wrox, 2019.

[52] R. Rousselet, "Provider Package Documentation," https://pub.dev/packages/provider, accedido: enero 2026.

[53] Google, "Flutter Web Support," https://docs.flutter.dev/platform-integration/web, accedido: enero 2026.

[54] Google, "Web Renderers in Flutter," https://docs.flutter.dev/platform-integration/web/renderers, accedido: enero 2026.

[55] M. Katz y K. Moore, "Flutter Apprentice," Kodeco, 2023.

[56] Material Design, "Material Design 3 Guidelines," https://m3.material.io/, accedido: enero 2026.

[57] W3C, "Web Content Accessibility Guidelines (WCAG) 2.1," https://www.w3.org/WAI/WCAG21/quickref/, accedido: enero 2026.

ANEXOS

7.1. Modelos de datos del sistema

```
Modelo Usuario (Python - SQLAlchemy)
class Usuario(db.Model):
    __tablename__ = 'usuarios'

    id = db.Column(mysql.BIGINT(unsigned=True), primary_key=True, autoincrement=True)
    email = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    first_name = db.Column(db.String(100), nullable=False)
    last_name = db.Column(db.String(100), nullable=False)
    phone = db.Column(db.String(20))
    role = db.Column(db.Enum('teacher', 'admin', 'superAdmin', name='user_role'), default='teacher')
    has_training = db.Column(db.Boolean, default=False)
    training_date = db.Column(db.Date)
    is_active = db.Column(db.Boolean, default=True)
    created_at = db.Column(db.DateTime, server_default=db.func.current_timestamp())
    updated_at = db.Column(db.DateTime, server_default=db.func.current_timestamp(),
                           onupdate=db.func.current_timestamp())

    def to_dict(self):
        return {
            'id': self.id, 'email': self.email,
            'first_name': self.first_name, 'last_name': self.last_name,
            'phone': self.phone, 'role': self.role,
            'has_training': self.has_training, 'is_active': self.is_active,
            'created_at': self.created_at.isoformat() if self.created_at else None
        }
Modelo Usuario (Dart - Flutter)
class User {
  final String id;
  final String firstName;
  final String lastName;
  final String email;
  final String? phone;
  final UserRole role;
  final DateTime? createdAt;

  const User({
    required this.id, required this.firstName, required this.lastName,
    required this.email, this.phone, required this.role, this.createdAt,
  });

  String get fullName => '$firstName $lastName';

  factory User.fromJson(Map<String, dynamic> json) => User(
    id: json['id'] as String,
    firstName: json['firstName'] as String,
    lastName: json['lastName'] as String,
    email: json['email'] as String,
    phone: json['phone'] as String?,
    role: UserRole.values.firstWhere((e) => e.name == json['role'],
        orElse: () => UserRole.teacher),
    createdAt: json['createdAt'] != null ? DateTime.parse(json['createdAt']) : null,
  );

  Map<String, dynamic> toJson() => {
    'id': id, 'firstName': firstName, 'lastName': lastName,
    'email': email, 'phone': phone, 'role': role.name,
    'createdAt': createdAt?.tolso8601String(),
  };
}
enum UserRole { teacher, admin, superAdmin }
```

7.2. Modelo de Aulas

```
class Aula(db.Model):
    __tablename__ = 'aulas'

    id = db.Column(mysql.BIGINT(unsigned=True), primary_key=True, autoincrement=True)
    nombre = db.Column(db.String(100), nullable=False)
    ubicacion = db.Column(db.String(100), nullable=False)
    capacidad = db.Column(db.Integer, nullable=False, default=15)
    horario_inicio = db.Column(db.Time, default=time(7, 0))
    horario_fin = db.Column(db.Time, default=time(16, 0))
    almuerzo_inicio = db.Column(db.Time, default=time(12, 0))
    almuerzo_fin = db.Column(db.Time, default=time(13, 0))
    dias_disponibles = db.Column(db.JSON, default=[1, 2, 3, 4, 5])
    foto_url = db.Column(db.String(500))
    descripcion = db.Column(db.Text)
    latitud = db.Column(db.Numeric(10, 8))
    longitud = db.Column(db.Numeric(11, 8))
    is_active = db.Column(db.Boolean, default=True)
    id_sede = db.Column(mysql.BIGINT(unsigned=True), db.ForeignKey('sedes.id_sede'))
    id_facultad = db.Column(mysql.BIGINT(unsigned=True), db.ForeignKey('facultades.id_facultad'))
    id_tecnico = db.Column(mysql.BIGINT(unsigned=True), db.ForeignKey('usuarios.id'))
    created_at = db.Column(db.DateTime, server_default=db.func.current_timestamp())

    agendas = db.relationship('Agenda', backref='aula', lazy=True)
```

7.3. Modelo de Agendamientos

```
class Agenda(db.Model):
    __tablename__ = 'agendas'

    id = db.Column(mysql.BIGINT(unsigned=True), primary_key=True, autoincrement=True)
    fecha = db.Column(db.Date, nullable=False)
    hora_inicio = db.Column(db.Time, nullable=False)
    hora_fin = db.Column(db.Time, nullable=False)
    id_user = db.Column(mysql.BIGINT(unsigned=True), db.ForeignKey('usuarios.id'), nullable=False)
    id_aula = db.Column(mysql.BIGINT(unsigned=True), db.ForeignKey('aulas.id'), nullable=False)
    docente_nombre = db.Column(db.String(100))
    materia = db.Column(db.String(100), nullable=False)
    ciclo = db.Column(db.String(20), nullable=False)
    carrera = db.Column(db.String(100), nullable=False)
    paralelo = db.Column(db.String(10), nullable=False)
    grupo = db.Column(db.String(50), default='Grupo único')
    numero_alumnos = db.Column(db.Integer, nullable=False)
    tipo = db.Column(db.Enum('regular', 'lunch', 'blocked', name='tipo_agenda'), default='regular')
    capacitacion = db.Column(db.Boolean, nullable=False, default=False)
    estado = db.Column(db.Enum('pendiente', 'aprobado', 'cancelado', 'penalizado',
                        name='estado_agenda'), default='pendiente')
    notas = db.Column(db.Text)
    repeat_weekly = db.Column(db.Boolean, default=False)
    cancellation_reason = db.Column(db.Text)
    cancelled_by = db.Column(mysql.BIGINT(unsigned=True), db.ForeignKey('usuarios.id'))
    cancelled_at = db.Column(db.DateTime)
    created_at = db.Column(db.DateTime, server_default=db.func.current_timestamp())
```

7.4. Sistema de autenticación y autorización

```
Decoradores de autorización (Python)
from functools import wraps
from flask import request, jsonify, g
import jwt, os
from app.models import Usuario
def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = None
        auth_header = request.headers.get('Authorization')
        if auth_header:
            parts = auth_header.split()
            if len(parts) == 2 and parts[0].lower() == 'bearer':
                token = parts[1]

        if not token:
            return jsonify({'error': 'Token requerido', 'code': 'TOKEN_MISSING'}), 401

        try:
            payload = jwt.decode(token, os.getenv('JWT_SECRET_KEY'), algorithms=['HS256'])
            current_user = Usuario.query.get(payload.get('user_id'))
            if not current_user or not current_user.is_active:
                return jsonify({'error': 'Usuario no válido'}), 401
            g.current_user = current_user
        except jwt.ExpiredSignatureError:
            return jsonify({'error': 'Token expirado'}), 401
        except jwt.InvalidTokenError:
            return jsonify({'error': 'Token inválido'}), 401

        return f(*args, **kwargs)
    return decorated
def admin_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        if g.current_user.role not in ['admin', 'superAdmin']:
            return jsonify({'error': 'Se requiere rol de administrador'}), 403
        return f(*args, **kwargs)
    return decorated
def training_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        if g.current_user.role in ['admin', 'superAdmin']:
            return f(*args, **kwargs)
        if not g.current_user.has_training:
            return jsonify({'error': 'Se requiere capacitación'}), 403
        return f(*args, **kwargs)
    return decorated
Servicio de autenticación (Dart)
class AuthService {
  final ApiClient _api;
  static const String _tokenKey = 'auth_token';
  User? _currentUser;

  AuthService({ApiClient? api}) : _api = api ?? ApiClient();

  User? get currentUser => _currentUser;
  bool get isAuthenticated => _currentUser != null;
  bool get isAdmin => _currentUser?.role == UserRole.admin ||
                      _currentUser?.role == UserRole.superAdmin;

  Future<AuthResult> login({required String email, required String password}) async {
    if (email.isEmpty || password.isEmpty) {
      return AuthResult.failure('Complete todos los campos');
    }

    final response = await _api.post<Map<String, dynamic>>(
      '/auth/login',
      body: {'email': email, 'password': password},
    );

    if (response.isSuccess && response.data != null) {
      final token = response.data!['token'] as String?;
      final userData = response.data!['user'] as Map<String, dynamic>?;

      if (token != null && userData != null) {
        await _saveToken(token);
        _api.setAuthToken(token);
        _currentUser = User.fromJson(userData);
        return AuthResult.success(_currentUser!);
      }
    }
    return AuthResult.failure(response.error ?? 'Error al iniciar sesión');
  }

  Future<void> logout() async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.remove(_tokenKey);
    _api.setAuthToken(null);
    _currentUser = null;
  }
}
class AuthResult {
  final User? user;
  final String? error;
  final bool isSuccess;

  const AuthResult._({this.user, this.error, required this.isSuccess});
  factory AuthResult.success(User user) => AuthResult._(user: user, isSuccess: true);
  factory AuthResult.failure(String msg) => AuthResult._(error: msg, isSuccess: false);
}
```

7.5. Funciones utilitarias

```
def admin_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        if g.current_user.role not in ['admin', 'superAdmin']:
            return jsonify({'error': 'Se requiere rol de administrador'}), 403
        return f(*args, **kwargs)
    return decorated
def training_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        if g.current_user.role in ['admin', 'superAdmin']:
            return f(*args, **kwargs)
        if not g.current_user.has_training:
            return jsonify({'error': 'Se requiere capacitación'}), 403
        return f(*args, **kwargs)
    return decorated
Servicio de autenticación (Dart)
class AuthService {
  final ApiClient _api;
  static const String _tokenKey = 'auth_token';
  User? _currentUser;

  AuthService({ApiClient? api}) : _api = api ?? ApiClient();

  User? get currentUser => _currentUser;
  bool get isAuthenticated => _currentUser != null;
  bool get isAdmin => _currentUser?.role == UserRole.admin ||
                      _currentUser?.role == UserRole.superAdmin;

  Future<AuthResult> login({required String email, required String password}) async {
    if (email.isEmpty || password.isEmpty) {
      return AuthResult.failure('Complete todos los campos');
    }

    final response = await _api.post<Map<String, dynamic>>(
      '/auth/login',
      body: {'email': email, 'password': password},
    );
```

7.6. Servicio de procesamiento de imágenes

```
import os, uuid
from PIL import Image
from werkzeug.utils import secure_filename
from werkzeug.datastructures import FileStorage
from typing import Optional, Tuple
ALLOWED_EXTENSIONS = {'jpg', 'jpeg', 'png', 'webp', 'gif'}
MAX_FILE_SIZE = 5 * 1024 * 1024
RESIZE_CONFIG = {'aula': (800, 600), 'carousel': (1920, 1080),
             'about': (1200, 800), 'profile': (400, 400)}
class ImageService:
    @staticmethod
    def save_image(file: FileStorage, folder: str,
                   resize: Optional[Tuple[int, int]] = None) -> Tuple[Optional[str], Optional[str]]:
        if not file or not file.filename:
            return None, "No se proporcionó archivo"

        ext = file.filename.rsplit('.', 1)[1].lower() if '.' in file.filename else ''
        if ext not in ALLOWED_EXTENSIONS:
            return None, f"Formato no permitido. Use: {', '.join(ALLOWED_EXTENSIONS)}"

        file.seek(0, 2)
        if file.tell() > MAX_FILE_SIZE:
            return None, "El archivo excede 5MB"
        file.seek(0)

        stored_name = f"{uuid.uuid4()}.jpg"
        upload_path = os.path.join('static', 'uploads', folder)
        os.makedirs(upload_path, exist_ok=True)

        resize = resize or RESIZE_CONFIG.get(folder, (800, 600))

        try:
            img = Image.open(file)
            if img.mode != 'RGB':
                img = img.convert('RGB')
            img.thumbnail(resize, Image.Resampling.LANCZOS)
            filepath = os.path.join(upload_path, stored_name)
            img.save(filepath, 'JPEG', quality=85, optimize=True)
            return f"/static/uploads/{folder}/{stored_name}", None
        except Exception as e:
            return None, f"Error: {str(e)}"
```

7.7. Configuración de la aplicación Flask

```
import os, uuid
from PIL import Image
from werkzeug.utils import secure_filename
from werkzeug.datastructures import FileStorage
from typing import Optional, Tuple
ALLOWED_EXTENSIONS = {'jpg', 'jpeg', 'png', 'webp', 'gif'}
MAX_FILE_SIZE = 5 * 1024 * 1024
RESIZE_CONFIG = {'aula': (800, 600), 'carousel': (1920, 1080),
                 'about': (1200, 800), 'profile': (400, 400)}
class ImageService:
    @staticmethod
    def save_image(file: FileStorage, folder: str,
                   resize: Optional[Tuple[int, int]] = None) -> Tuple[Optional[str], Optional[str]]:
        if not file or not file.filename:
            return None, "No se proporcionó archivo"

        ext = file.filename.rsplit('.', 1)[1].lower() if '.' in file.filename else ''
        if ext not in ALLOWED_EXTENSIONS:
            return None, f"Formato no permitido. Use: {', '.join(ALLOWED_EXTENSIONS)}"

        file.seek(0, 2)
        if file.tell() > MAX_FILE_SIZE:
            return None, "El archivo excede 5MB"
        file.seek(0)

        stored_name = f"{uuid.uuid4()}.jpg"
        upload_path = os.path.join('static', 'uploads', folder)
        os.makedirs(upload_path, exist_ok=True)

        resize = resize or RESIZE_CONFIG.get(folder, (800, 600))

        try:
            img = Image.open(file)
            if img.mode != 'RGB':
                img = img.convert('RGB')
            img.thumbnail(resize, Image.Resampling.LANCZOS)
            filepath = os.path.join(upload_path, stored_name)
            img.save(filepath, 'JPEG', quality=85, optimize=True)
            return f"/static/uploads/{folder}/{stored_name}", None
        except Exception as e:
            return None, f"Error: {str(e)}"
```

7.8. Configuración de la aplicación Flutter

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:flutter_localizations/flutter_localizations.dart';
void main() {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(const AppProviders());
}
class AppProviders extends StatelessWidget {
  const AppProviders({super.key});
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => ThemeProvider()),
        ChangeNotifierProvider(create: (_) => LocaleProvider()),
      ],
      child: const MyApp(),
    );
  }
}
class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    final themeProvider = context.watch<ThemeProvider>();

    return MaterialApp(
      title: 'ITE VR - Sistema de Agendamiento',
      debugShowCheckedModeBanner: false,
      theme: AppTheme.lightTheme,
      darkTheme: AppTheme.darkTheme,
      themeMode: themeProvider.isDarkMode ? ThemeMode.dark : ThemeMode.light,
      localizationsDelegates: const [
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      initialRoute: '/',
      routes: {
        '/': (context) => const LandingPage(),
        '/login': (context) => const LoginPage(),
        '/teacher-scheduling': (context) => const TeacherSchedulingPage(),
        '/admin-scheduling': (context) => const AdminSchedulingPage(),
      },
    );
  }
}
```

7.9. Dependencias del proyecto

```
requirements.txt (Python)
Flask==3.1.1
Werkzeug==3.1.3
Flask-SQLAlchemy==3.1.1
SQLAlchemy==2.0.41
Flask-Migrate==4.0.5
Flask-Bcrypt==1.0.1
bcrypt==4.3.0
PyJWT==2.10.1
Flask-Cors==6.0.0
python-dotenv==1.1.0
pymysql
gunicorn
Pillow>=10.0.0
pubspec.yaml (Flutter - dependencias principales)
dependencies:
  flutter:
    sdk: flutter
  provider: ^6.0.0
  shared_preferences: ^2.2.0
  http: ^1.1.0
  intl: ^0.18.0
  flutter_localizations:
    sdk: flutter
```

Universidad Católica de Cuenca	AUTORIZACIÓN DE PUBLICACIÓN EN EL REPOSITORIO INSTITUCIONAL

7.10. Autorización de publicación en el repositorio institucional

Adrián Orlando Álvarez Dos Santos portador(a) de la cédula de ciudadanía Nº **0151547916**. En calidad de autor/a y titular de los derechos patrimoniales del trabajo de titulación "**Desarrollo de un aplicativo web para la gestión de laboratorios, aulas de realidad virtual y otros espacios educativos**" de conformidad a lo establecido en el artículo 114 Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación, reconozco a favor de la Universidad Católica de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos y no comerciales. Autorizo además a la Universidad Católica de Cuenca, para que realice la publicación de este trabajo de titulación en el Repositorio Institucional de conformidad a lo dispuesto en el artículo 144 de la Ley Orgánica de Educación Superior.

Cuenca, **23 de marzo de 2026**

F: .....

Adrián Orlando Álvarez Dos Santos

C.I. **0151547916**