



UNIVERSIDAD TÉCNICA DE AMBATO

**FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E
INDUSTRIAL**

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

Tema:

**SISTEMA BASADO EN UN FRAMEWORK MULTIPLATAFORMA PARA
EL SEGUIMIENTO Y CONTROL DE COBROS EN LA ACADEMIA
OFICIAL DE FÚTBOL PORTO AMBATO.**

Trabajo de titulación modalidad Proyecto de Investigación, presentado previo a la
obtención del título de Ingeniero en Tecnologías de la Información

ÁREA: Base de datos y Sistemas informativos

LÍNEA DE INVESTIGACIÓN: Software, Tecnologías de la Información y
Ciencias de datos.

AUTOR: Rogers Brando Bury Villacis

TUTOR: Ing. Clay Fernando Aldás Flores, Mg.

Ambato - Ecuador

enero - 2026

APROBACIÓN DEL TUTOR

En calidad de tutor del trabajo de titulación con el tema: SISTEMA BASADO EN UN FRAMEWORK MULTIPLATAFORMA PARA EL SEGUIMIENTO Y CONTROL DE COBROS EN LA ACADEMIA OFICIAL DE FÚTBOL PORTO AMBATO, desarrollado bajo la modalidad Proyecto de Investigación por el señor Rogers Brando Bury Villacis, estudiante de la Carrera de Tecnologías de la Información, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 17 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.3 del instructivo del reglamento referido.

Ambato, enero 2026.



Ing. Clay Fernando Aldás Flores, Mg.
TUTOR

AUTORÍA

El presente trabajo de titulación con el tema: SISTEMA BASADO EN UN FRAMEWORK MULTIPLATAFORMA PARA EL SEGUIMIENTO Y CONTROL DE COBROS EN LA ACADEMIA OFICIAL DE FÚTBOL PORTO AMBATO, es absolutamente original, auténtico y personal y ha observado los preceptos establecidos en la Disposición General Quinta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato. En tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato, enero 2026.



Rogers Brando Bury Villacis

C.C. 1850554450

AUTOR

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato para que reproduzca total o parcialmente este trabajo de titulación dentro de las regulaciones legales e institucionales correspondientes. Además, cedo todos mis derechos de autor a favor de la institución con el propósito de su difusión pública, por lo tanto, autorizo su publicación en el repositorio virtual institucional como un documento disponible para la lectura y uso con fines académicos e investigativos de acuerdo con la Disposición General Cuarta del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato.

Ambato, enero 2026.



Rogers Brando Bury Villacis

C.C. 1850554450

AUTOR

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del informe final del trabajo de titulación presentado por el señor Rogers Brando Bury Villacis, estudiante de la Carrera de Tecnologías de la Información, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado SISTEMA BASADO EN UN FRAMEWORK MULTIPLATAFORMA PARA EL SEGUIMIENTO Y CONTROL DE COBROS EN LA ACADEMIA OFICIAL DE FÚTBOL PORTO AMBATO, nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 19 del Reglamento para la Titulación de Grado en la Universidad Técnica de Ambato y el numeral 6.4 del instructivo del reglamento referido. Para cuya constancia suscribimos, conjuntamente con el señor presidente del Tribunal.

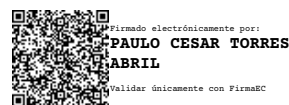
Ambato, enero 2026.



Ing. Franklin Oswaldo Mayorga Mayorga, Mg.
PRESIDENTE DEL TRIBUNAL



Ing. Leonardo David Torres Valverde, Mg.
PROFESOR CALIFICADOR



Ing. Paulo Cesar Torres Abril, Mg.
PROFESOR CALIFICADOR

DEDICATORIA

A Dios, por acompañarme en cada etapa de mi vida y por brindarme la fortaleza necesaria para cumplir mis objetivos.

A mis padres, Ana Villacís y Honorio Bury, cuyo amor, esfuerzo y ejemplo han sido el fundamento de mis metas y logros.

A mi abuelita, Carmen Sailema, a quien he querido profundamente desde mi infancia y cuya presencia, cariño y guía han sido una luz constante en mi vida.

A mis hermanos, Jordan Bury y Jessica Bury, por su apoyo, compañía y confianza. Cada paso que he dado ha sido más llevadero gracias a ustedes, y me enorgullece avanzar sabiendo que compartimos los mismos valores y el deseo de superarnos juntos.

AGRADECIMIENTO

Quiero agradecer a todas las personas que estuvieron presentes en este camino que me hicieron posible que este proyecto llegue a su fin. Cada gesto de apoyo, cada palabra de ánimo y cada momento compartido significó más de lo que puedo expresar aquí.

A mis amigos cercanos, gracias por acompañarme durante todo este proceso, por estar presentes en los momentos importantes y por compartir conmigo experiencias que hicieron este camino más llevadero. Su compañía y apoyo sincero fueron un regalo que siempre voy a valorar.

A mi tutor, Clay Fernando Aldas Flores, le agradezco por su guía y por su tiempo durante la elaboración de este trabajo. Su apoyo fue clave para avanzar con seguridad y claridad.

A mi pareja, Lisseth Stephania Vargas Tubón gracias por estar conmigo en cada etapa, por tu comprensión en los días difíciles, por tu apoyo constante y por todo el amor que me das. Tu presencia hizo que este camino fuera más fácil y significativo.

A la universidad, por brindarme un espacio de aprendizaje donde pude crecer, equivocarme, mejorar y descubrir nuevas oportunidades.

A todos ustedes, gracias. Este logro también es parte de lo que cada uno aportó en mi vida.

ÍNDICE GENERAL DE CONTENIDOS

PORTADA.....	i
APROBACIÓN DEL TUTOR.....	ii
AUTORÍA.....	iii
DERECHOS DE AUTOR	iv
APROBACIÓN DEL TRIBUNAL DE GRADO.....	v
DEDICATORIA.....	vi
AGRADECIMIENTO	vii
ÍNDICE GENERAL DE CONTENIDOS	viii
ÍNDICE DE TABLAS.....	xii
ÍNDICE DE FIGURAS.....	xiv
ÍNDICE DE ANEXOS	xvii
RESUMEN EJECUTIVO	xviii
ABSTRACT	xix
CAPÍTULO I. MARCO TEÓRICO.....	1
1.1 Tema de investigación.....	1
1.1.1 Planteamiento del problema.....	1
1.2 Antecedentes investigativos	2
1.3 Fundamentación teórica	4

1.3.1 Tecnologías de la Información.....	4
1.3.2 Informática	4
1.3.3 Desarrollo de Software.....	5
1.3.4 Framework Multiplataforma	5
1.3.5 Gestión y Control de Cobros.....	5
1.4 Objetivos	6
1.4.1 Objetivo general	6
1.4.2 Objetivos específicos	6
CAPÍTULO II. METODOLOGÍA.....	7
2.1 Materiales.....	7
2.2 Métodos.....	8
2.2.1 Modalidad de la investigación	8
2.2.2 Población y muestra	9
2.2.3 Recolección de información.....	9
2.2.4 Procesamiento y análisis de datos	22
CAPÍTULO III. RESULTADOS Y DISCUSIÓN.....	24
3.1 Análisis de procesos y discusión de los resultados	24
3.1.1 Modelo de procesos	24
3.1.2 Síntesis de problemas y requisitos del sistema	27
3.2 Diseño del proceso automatizado de cobros	28

3.2.1 Comparación entre el proceso actual y el proceso automatizado	32
3.3 Arquitectura del sistema y tecnologías seleccionadas	34
3.3.1 Arquitectura general del sistema	34
3.3.2 Normativas y estándares considerados.....	35
3.3.3 Análisis de frameworks multiplataforma	35
3.3.4 Framework multiplataforma Flutter	37
3.3.5 Back-end Node.js y Express	43
3.3.6 Base de datos relacional en PostgreSQL.....	46
3.3.7 Seguridad del Sistema	50
3.3.8 Diagrama metodológico del proyecto	53
3.4 Metodología de desarrollo.....	54
3.4.1 Comparación de metodologías de desarrollo	54
3.4.2 Metodología híbrida Scrumban.....	56
3.4.3 Beneficios obtenidos con el enfoque Scrumban	57
3.5 Desarrollo de la propuesta.....	57
3.5.1 Definición de requisitos	58
3.5.2 Backlog del proyecto	62
3.5.3 Planificación de actividades.....	66
3.5.4 Desarrollo iterativo.....	67
3.6 Eficiencia en la gestión de cobros en la Academia	106

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES	109
4.1 Conclusiones	109
4.2 Recomendaciones.....	110
REFERENCIAS BIBLIOGRÁFICAS.....	111
ANEXOS.....	114

ÍNDICE DE TABLAS

Tabla 1. Cadenas de búsqueda.	2
Tabla 2. Criterios de exclusión e inclusión.	3
Tabla 3. Población.....	9
Tabla 4. Validación de la encuesta en SPSS.	11
Tabla 5. Resultados del ítem 1.	11
Tabla 6. Resultados del ítem 2.	12
Tabla 7. Resultados del ítem 3.	13
Tabla 8. Resultados del ítem 4.	14
Tabla 9. Resultados del ítem 5.	15
Tabla 10. Resultados del ítem 6.	16
Tabla 11. Resultados del ítem 7.	17
Tabla 12. Resultados del ítem 8.	18
Tabla 13. Resultados del ítem 9.	19
Tabla 14. Resultados del ítem 10.	20
Tabla 15. Resultados de la ficha de observación.	21
Tabla 16. Descripción del proceso actual.	26
Tabla 17. Descripción del proceso automatizado.	30
Tabla 18. Tabla comparativa con el proceso actual y optimizado.	32
Tabla 19. Comparación de frameworks multiplataforma.	35
Tabla 20. Ventajas y desventajas.	38

Tabla 21. Comparación entre versiones de Flutter.....	40
Tabla 22. Casos de éxito con Flutter.	42
Tabla 23. Ventajas y desventajas de Node.js	45
Tabla 24. Ventajas y desventajas de PostgreSQL.....	48
Tabla 25. Comparación entre metodologías de desarrollo de software.	55
Tabla 26. Lista de verificación de accesibilidad aplicada.....	60
Tabla 27. Actores principales del proyecto.....	62
Tabla 28. Módulos del sistema.....	62
Tabla 29. Período de ejecución de actividades en la gestión de cobros.....	107
Tabla 30. Resultados de mejora porcentual en la gestión de cobros.....	108

ÍNDICE DE FIGURAS

Figura 1. Resultados del ítem 1.....	11
Figura 2. Resultados del ítem 2.....	12
Figura 3. Resultados del ítem 3.....	13
Figura 4. Resultados del ítem 4.....	14
Figura 5. Resultados del ítem 5.....	15
Figura 6. Resultados del ítem 6.....	16
Figura 7. Resultados del ítem 7.....	17
Figura 8. Resultados del ítem 8.....	18
Figura 9. Resultados del ítem 9.....	19
Figura 10. Resultados del ítem 10.....	20
Figura 11. Diagrama de procesos actual de la gestión de pagos.	25
Figura 12. Diagrama de procesos automatizado de la gestión de pagos.....	29
Figura 13. Diagrama metodológico del proyecto.....	53
Figura 14. Diagrama funcional del sistema..	64
Figura 15. Actividades en Planner.	66
Figura 16. Diccionario de datos tabla “profesores”.	68
Figura 17. Diagrama de la estructura principal de los usuarios y roles.	69
Figura 18. Diagrama de la estructura principal de los usuarios y roles.	70
Figura 19. Proyecto Node.js.....	72
Figura 20. Estructura del backend.....	72

Figura 21. Dependencias requeridas	73
Figura 22. Archivo de configuración index.ts.....	74
Figura 23. Archivo de configuración app.ts.....	75
Figura 24. Mockup del inicio de sesión.	76
Figura 25. Mockup del perfil.	77
Figura 26. Mockup de la pantalla roles.....	78
Figura 27. Mockup de la pantalla de usuarios.....	78
Figura 28. Mockup de la pantalla de profesores.	79
Figura 29. Mockup de la pantalla de categorías.....	80
Figura 30. Mockup de la pantalla de subcategorías.	80
Figura 31. Mockup de la pantalla de estudiantes.	81
Figura 32. Mockup de la pantalla de pagos de mensualidades.	82
Figura 33. Estructura del proyecto en Flutter.....	83
Figura 34. Función de login en el backend.....	84
Figura 35. Función de login en el frontend.	85
Figura 36. Servicio de crear un estudiante en el backend.	86
Figura 37. Controller del servicio crear un estudiante.	87
Figura 38. Servicio para crear un profesor.....	88
Figura 39. Controller de crear un profesor.....	88
Figura 40. Servio de crear categoría.	89
Figura 41. Servicio de crear subcategoría.....	89

Figura 42. Servicio de crear matricula.	90
Figura 43. Pantalla de estudiantes.	91
Figura 44. Pantalla para crear un estudiante con su matrícula.	91
Figura 45. Servicio de crear una mensualidad.	93
Figura 46. Servicio de actualizar pago.	93
Figura 47. Servicio de anular pago.	94
Figura 48. Servicio de crear pago.	95
Figura 49. Pantalla de pagos.	95
Figura 50. Datos de conexión para la base de datos.	98
Figura 51. Conexión al servidor de Railway.	99
Figura 52. Tablas en Railway.	100
Figura 53. Creación del backend en Railway.	100
Figura 54. Variables de configuración del backend.	101
Figura 55. Backend desplegado.	102
Figura 56. Configuración de enlace para poder usar el backend.	102
Figura 57. Creación del sitio en Render.	103
Figura 58. Parámetros iniciales.	104
Figura 59, Configuración de dominio.	105
Figura 60. Sitio desplegado.	105

ÍNDICE DE ANEXOS

Anexo A. Manual de usuario.	114
Anexo B. Resultado de la fiabilidad de la encuesta en Spss.	137
Anexo C. Repositorio del frontend.	137
Anexo E. Repositorio del backend.	138

RESUMEN EJECUTIVO

En este proyecto se analizó el proceso de gestión de cobros en la Academia Oficial de Fútbol Porto Ambato, con el objetivo de identificar las limitaciones derivadas del manejo manual y disperso de matrículas y mensualidades. El diagnóstico evidenció problemas como errores en los valores registrados, duplicidad de información, demoras en la verificación de pagos y dificultades para generar reportes financieros oportunos.

La investigación definió el problema central, los objetivos y los fundamentos teóricos vinculados a la administración deportiva y al uso de tecnologías multiplataforma. Se aplicó una metodología basada en revisión bibliográfica, observación directa y encuestas dirigidas a representantes, lo que permitió comprender el funcionamiento real del proceso y conocer la percepción de los usuarios respecto al sistema actual.

Con base en los resultados, se diseñó e implementó un sistema multiplataforma que integra estudiantes, matrículas, mensualidades y pagos en un único entorno. La solución se desarrolló utilizando Flutter para la interfaz, Node.js con Express para la lógica de negocio y PostgreSQL como base de datos, lo que garantiza coherencia, trazabilidad y acceso en tiempo real a la información financiera.

Los resultados demuestran que la incorporación del sistema moderniza significativamente el proceso de cobros, agiliza el registro y verificación de pagos y reduce la carga operativa del personal administrativo. Asimismo, mejora la transparencia hacia los representantes y facilita la generación de reportes clave para el control financiero y la toma de decisiones institucionales.

Palabras claves: Gestión de cobros, Framework multiplataforma, Flutter, Desarrollo de software, Scrumban, Automatización de procesos, Administración deportiva.

ABSTRACT

In this project, the fee management process at the Academia Oficial de Fútbol Porto Ambato was analyzed with the objective of identifying the limitations derived from the manual and fragmented handling of enrollments and monthly payments. The diagnostic phase revealed issues such as errors in recorded amounts, duplicated information, delays in payment verification, and difficulties in generating timely financial reports.

The research defined the central problem, objectives, and theoretical foundations related to sports administration and the use of multiplatform technologies. A methodology based on bibliographic review, direct observation, and surveys administered to representatives was applied, which made it possible to understand the real functioning of the process and to gather users' perceptions of the current system.

Based on the results, a multiplatform system was designed and implemented to integrate students, enrollments, monthly fees, and payments into a single environment. The solution was developed using Flutter for the user interface, Node.js with Express for the business logic, and PostgreSQL as the database, ensuring consistency, traceability, and real-time access to financial information.

The results show that the incorporation of the system significantly modernizes the fee management process, streamlines the registration and verification of payments, and reduces the administrative workload. It also improves transparency for representatives and facilitates the generation of key reports for financial control and institutional decision-making.

Keywords: Payment management, Multiplatform framework, Flutter, Software development, Scrumban, Process automation, Sports administration.

CAPÍTULO I. MARCO TEÓRICO

1.1 Tema de investigación

SISTEMA BASADO EN UN FRAMEWORK MULTIPLATAFORMA PARA EL SEGUIMIENTO Y CONTROL DE COBROS EN LA ACADEMIA OFICIAL DE FÚTBOL PORTO AMBATO.

1.1.1 Planteamiento del problema

A nivel mundial, se reconoce que incentivar la práctica deportiva en la infancia es fundamental para un desarrollo integral y saludable. Diversos estudios coinciden en que el deporte en edades tempranas fortalece no solo la condición física de los niños, sino también su bienestar emocional y social [1] De igual manera, investigaciones recientes destacan que realizar actividad física de forma regular ayuda a prevenir problemas de salud mental, como la ansiedad o la depresión, aportando beneficios duraderos en la calidad de vida de niños y adolescentes [2].

En América Latina, la creciente demanda de actividades extracurriculares ha impulsado la proliferación de academias deportivas, sin embargo, muchas de estas academias enfrentan desafíos relacionados con la gestión administrativa. La falta de herramientas tecnológicas en las operaciones cotidianas de las academias limita su capacidad para organizar, supervisar y optimizar procesos importantes como el seguimiento de cobros, pagos, inscripciones y la comunicación con los padres. Esto se traduce en una ineficiencia operativa que puede afectar la calidad del servicio y, en última instancia, el rendimiento de los estudiantes [3].

En Ecuador, la administración deportiva ha cobrado gran relevancia debido al crecimiento de clubes y organizaciones deportivas formativas. Sin embargo, muchos de estos clubes enfrentan desafíos en su gestión interna, ya que aún emplean prácticas tradicionales y métodos empíricos que no se ajustan a las demandas actuales. Esta falta de modernización en los procesos administrativos limita la eficiencia operativa y el desarrollo de la organización. La necesidad de mejorar la gestión y administración de estos clubes es cada vez más urgente, ya que la correcta organización de sus

actividades impacta directamente en la calidad del servicio y el bienestar de los Atletas[4].

En la ciudad de Ambato, reconocida por su dinamismo comercial y su compromiso con el deporte, academias como Porto Ambato desempeñan un papel crucial en la formación de jóvenes futbolistas. No obstante, esta institución enfrenta retos en la gestión eficiente de sus procesos administrativos, como el seguimiento de pagos y la comunicación con padres de familia, lo cual puede conllevar a errores humanos y a una desorganización financiera.

Estas deficiencias se traducen en retrasos en la actualización de registros, dificultades en el control de mensualidades, malentendidos con los representantes, y pérdida de ingresos por falta de seguimiento oportuno. En el largo plazo, estos problemas afectan la sostenibilidad de la academia, comprometen su reputación institucional y limitan su capacidad de crecimiento y mejora continua.

1.2 Antecedentes investigativos

Para el desarrollo del presente proyecto se realizó una revisión sistemática de la literatura en bases de datos científicas indexadas como Scopus, IEEE Xplore, Google Scholar, y repositorios de universidades ecuatorianas. Se realizará a fin de identificar investigaciones previas relacionadas con el desarrollo de sistemas web, uso de React, y el rubro de cobros una institución deportiva, las cadenas de búsqueda utilizadas se detallan en la

Tabla 1. Cadenas de búsqueda.

Base de datos	Cadenas de búsqueda	Núm. de artículos
IEEE Xplore	("All Metadata": "cross platform" OR "All Metadata": multiplatform) AND ("All Metadata": payment OR "All Metadata": fee Refined By: Year: 2020-2025	45
Scopus	TITLE-ABS-KEY ("cross platform" OR multiplatform) AND TITLE-ABS-KEY (payment OR fee)	55
Science Direct	("cross platform" OR multiplatform) AND (payment OR fee OR billing) AND (football OR soccer OR sport) Date Published: 2020 2025	89
	Total	189

De los 189 trabajos de investigación recolectados se ha establecido criterios de inclusión y exclusión para analizar su relevancia para el presente proyecto de investigación.

Tabla 2. Criterios de exclusión e inclusión.

Criterios de inclusión	Criterios de exclusión
Investigaciones publicadas entre los años 2020 y 2025.	Investigaciones publicadas antes del año 2020.
Artículos académicos de libre acceso o disponibles en bases indexadas (IEEE, Scopus, etc.)	Documentos sin acceso completo, sin revisión por pares o con fuentes no verificables.
Investigaciones aplicadas a academias deportivas, educativas o instituciones similares.	Investigaciones fuera del ámbito educativo o deportivo.

La transformación digital ha impulsado a diversas instituciones, incluidas las academias deportivas, a adoptar soluciones tecnológicas que optimicen sus procesos administrativos y financieros. En este contexto, el desarrollo de sistemas multiplataforma ha cobrado relevancia por su capacidad para mejorar la eficiencia operativa y la gestión de recursos.

Kuzmin e [5] compartieron su experiencia en la creación de una aplicación utilizando Flutter, destacando su eficiencia en el desarrollo de aplicaciones multiplataforma con un único código base, lo que resulta en una reducción significativa del tiempo y los recursos necesarios.

Boukhary y Colmenares [6] propusieron la arquitectura "Flutter Clean Architecture" para el desarrollo de aplicaciones móviles. Esta arquitectura facilita la separación de responsabilidades y mejora la mantenibilidad del código, siendo especialmente útil en aplicaciones que requieren una gestión eficiente del estado y una estructura escalable.

Özkurt [7] presentó una aplicación móvil desarrollada en Flutter para supervisar operaciones de limpieza institucional. La aplicación permitió una coordinación eficiente entre el personal de limpieza y los supervisores, mejorando la transparencia y la eficiencia operativa mediante el seguimiento en tiempo real de las tareas.

Además, plataformas como 360Player han sido adoptadas por academias deportivas de renombre, como la Barça Academy y la Rafa Nadal Academy, para centralizar y

optimizar procesos de gestión deportiva, incluyendo herramientas de análisis de video, planificación de entrenamientos y soluciones de pago integradas [8]

Estos antecedentes evidencian la viabilidad y los beneficios de implementar sistemas multiplataforma en entornos que requieren una gestión eficiente de procesos y recursos. La aplicación de tecnologías como Flutter y arquitecturas limpias no solo mejora la eficiencia operativa, sino que también facilita la escalabilidad y el mantenimiento de las aplicaciones, aspectos esenciales para instituciones deportivas en crecimiento. estado del arte describe las investigaciones (tesis y artículos) más recientes y actuales, y, le sirve al investigador como referencia para saber qué se ha hecho o hasta donde ha llegado la investigación en el tema.

1.3 Fundamentación teórica

1.3.1 Tecnologías de la Información

Las Tecnologías de la Información son un conjunto de herramientas, procedimientos y conocimientos técnicos que permiten, recopilar, procesar, almacenar y distribuir la información. Son uno de los elementos clave para la transformación digital de las organizaciones, especialmente para la eficacia de sus procesos internos. Investigaciones académicas recientes han confirmado el papel crucial de las Computadoras de la Información como “instrumento estratégico” para fortalecer los sistemas administrativos y de gestión educativa [9]

1.3.2 Informática

La informática es la rama del conocimiento que examina el procesamiento automático de datos usando sistemas informáticos. Está muy ligada al diseño y creación de programas, uniendo partes teóricas y prácticas. Desde la perspectiva de la investigación técnica, esta ciencia ofrece bases lógicas y algorítmicas que ayudan a construir soluciones automáticas para tareas en administración [10]

1.3.3 Desarrollo de Software

El desarrollo de software incluye las formas de hacer, los pasos que se siguen en el análisis, diseño, hacer, probar y poner en marcha programas computacionales. Este tema ha crecido mucho siendo una herramienta muy útil para la novedad institucional, especialmente en los dos sectores de educación y deportes, al dejar hacer plataformas buenas y amplias [11].

1.3.4 Framework Multiplataforma

Un framework multiplataforma es un entorno de desarrollo que posibilita la creación de aplicaciones capaces de ejecutarse en diversos sistemas operativos desde un solo núcleo de programación. Esta tecnología ha sido reconocida por su eficiencia en la reducción de tiempos de desarrollo, así como por la consistencia que ofrece en la experiencia del usuario [12].

1.3.5 Gestión y Control de Cobros

La gestión y control de cobros comprende el conjunto de políticas, procesos, roles y herramientas orientados a asegurar la emisión oportuna de obligaciones (matrículas y mensualidades), su notificación, el registro y validación de pagos por distintos medios, la conciliación diaria con reportes para la toma de decisiones. En organizaciones educativas y deportivas, este ciclo se apoya en principios de oportunidad, integridad, trazabilidad, seguridad y cumplimiento (p. ej., protección de datos), e incorpora plataformas digitales que estandarizan catálogos y calendarios, automatizan recordatorios y confirmaciones, emiten folios únicos y mantienen auditoría de cambios. Un esquema así mejora la liquidez y reduce la morosidad al acortar el tiempo de recuperación, disminuir errores operativos y elevar la calidad de la información directiva, constituyéndose en un componente clave de la sostenibilidad administrativa de la academia [13].

1.4 Objetivos

1.4.1 Objetivo general

Implantar un sistema usando framework multiplataforma para el seguimiento y control de cobros en la Academia Oficial de Fútbol Porto Ambato.

1.4.2 Objetivos específicos

- Analizar el proceso de registro y seguimiento de cobros en la Academia Oficial de Fútbol Porto Ambato.
- Investigar las características y beneficios de frameworks multiplataforma para su aplicación en el desarrollo del sistema.
- Desarrollar un sistema multiplataforma para el seguimiento y control de cobros.

CAPÍTULO II. METODOLOGÍA

2.1 Materiales

Para obtener la información necesaria en esta investigación se recurrió a dos técnicas: la encuesta y la observación directa. En primera instancia se administró un cuestionario de 10 ítems, con preguntas cerradas y de selección múltiple, dirigido a quienes intervienen en el proceso de cobros de la Academia Oficial de Fútbol Porto Ambato, principalmente padres, madres o representantes, y de forma complementaria profesores y personal administrativo. El propósito fue reconocer los medios de pago más utilizados, identificar las dificultades que se presentan en el registro y control de mensualidades, valorar la claridad y la eficacia del procedimiento vigente y estimar la disposición de los usuarios a adoptar una solución digital que permita recibir recordatorios, consultar saldos e historial y descargar comprobantes, con miras a perfeccionar la gestión de matrículas y pagos recurrentes.

Se efectuó una observación directa mediante una ficha estructurada, orientada a constatar en la academia cómo se registran los pagos, cómo se trasladan y archivan los comprobantes y hasta qué punto existe integración entre las tareas operativas y la administración. Esta revisión permitió evidenciar una elevada dependencia de procedimientos manuales y herramientas dispersas (hojas de cálculo y mensajería), una integración tecnológica limitada entre los registros operativos y los sistemas administrativos, y demoras en la disponibilidad de información confiable. Tales condiciones restringen la toma oportuna de decisiones, dificultan la disminución de errores y afectan la experiencia del usuario final, por lo que se refuerza la pertinencia de implementar una plataforma unificada que estandarice procesos, automatice recordatorios y asegure trazabilidad y transparencia a lo largo del ciclo de cobros.

2.2 Métodos

Este proyecto se efectuó con un enfoque mixto. Por un lado, se procesaron de forma estadística las respuestas de la encuesta aplicada a la muestra definida para describir hábitos, problemas y tendencias. Por otro, se revisaron de manera cualitativa las notas de la observación directa, organizándolas por temas para entender el contexto operativo. La combinación de ambos frentes permitió contrastar lo que dicen los usuarios con lo que realmente ocurre en el proceso y, con ello, afinar los criterios y requisitos de la solución propuesta.

2.2.1 Modalidad de la investigación

Investigación bibliográfica documental

Esta modalidad es esencial para fundamentar y contextualizar teóricamente el estudio. Se reviso artículos científicos, tesis, libros, y documentos académicos relacionados con la implementación de tecnologías multiplataforma, herramientas digitales para gestión de cobros y administración deportiva. Esto permitirá identificar teorías relevantes, antecedentes investigativos y mejores prácticas aplicadas en contextos similares.

Investigación de campo

Se realizo directamente en la Academia de Oficial Fútbol Porto Ambato. Esta modalidad permitió obtener información real y actualizada sobre el proceso de cobros y pagos, la interacción cotidiana del personal administrativo con los padres de familia y estudiantes, así como el funcionamiento del nuevo sistema. La interacción directa en el contexto real del problema garantizo la obtención de datos valiosos y precisos. La combinación de ambas modalidades proporciono solidez y profundidad a la investigación, permitiendo validar en campo los conocimientos y herramientas identificadas en la revisión bibliográfica-documental.

2.2.2 Población y muestra

La población del estudio está constituida por todas las personas relacionadas directamente con los procesos administrativos y de gestión financiera en la Academia de Fútbol Porto Ambato. Dado que la población es relativamente reducida, se trabajará integralmente con la totalidad del personal administrativo y una muestra representativa de padres o representantes.

Tabla 3. Población.

Grupo poblacional	Número	Porcentaje
Gerente Propietario	1	1,64%
Entrenadores	4	6,56%
Asistente de entrenador	1	1,64%
Padre o representante de estudiantes	65	90,16%
Total	78	100%

Por tratarse de una población inferior a cien integrantes, el muestreo resultó innecesario y se abordó a todos los participantes de manera integral.

2.2.3 Recolección de información

La información se recolecta mediante encuestas y observación será organizada, codificada y tabulada para su análisis. Se aplicarán técnicas estadísticas descriptivas para los datos cuantitativos, mientras que los resultados cualitativos se interpretarán mediante análisis temático. El objetivo de este procesamiento es identificar patrones significativos que permitan evaluar la efectividad del sistema multiplataforma en el seguimiento y control de cobros dentro de la academia.

Por otra parte, la observación directa en sesiones de entrenamiento y en el área administrativa permitió registrar el flujo real de cobro y registro, el traspaso de información entre cuadernos, mensajes, los tiempos de atención y los errores por transcripción.

Estos hallazgos evidencian limitaciones del esquema manual datos dispersos, validaciones tardías, duplicidades y ausencia de controles de integridad y sustentan la necesidad de un sistema unificado que automatice los cobros.

Validación de Instrumento

La consistencia interna del cuestionario se evaluó mediante el Alpha de Cronbach (α), aplicando la fórmula estándar sobre los ítems:

$$\alpha = \frac{\kappa}{\kappa - 1} \left(1 - \frac{\sum_{i=1}^{\kappa} S_i^2}{S_t^2} \right)$$

Donde:

κ = números de ítems del instrumento.

S_i^2 = varianza del ítem.

S_t^2 = varianza de la puntuación total al sumar los ítems.

El Alpha de Cronbach es un estadístico que permite estimar la fiabilidad interna de un conjunto de ítems en un cuestionario. Su valor varía entre 0 y 1; de forma general, puntajes iguales o superiores a 0,70 se interpretan como aceptables para estudios aplicados, y mientras más se aproximan a 1, mayor es la consistencia del instrumento.

En este proyecto, el coeficiente se calculó con el software SPSS para evaluar la coherencia de las respuestas entre los ítems y sustentar la calidad del instrumento aplicado. Este procedimiento ofrece evidencia empírica sobre la estabilidad de la escala utilizada y orienta decisiones de mejora cuando sea necesario.

Aplicación del Alpha de Cronbach

Para comprobar la coherencia de la encuesta, se aplicó en SPSS la prueba de alfa de Cronbach con los 10 ítems del cuestionario. Se obtuvo un valor $\alpha = 0,768$, lo que indica una fiabilidad aceptable.

Se evaluó la fiabilidad interna del cuestionario mediante el Alpha de Cronbach en SPSS, considerando los 10 ítems de la escala y eliminación por lista de casos. El análisis arrojó $\alpha = 0,768$, valor que indica buena consistencia y supera el umbral de 0,70 comúnmente aceptado para estudios aplicados. Este resultado respalda la solidez del instrumento utilizado en la investigación. La Tabla 4 muestra el resultado.

Tabla 4. Validación de la encuesta en SPSS.

Estadísticas de fiabilidad	
Alfa de Cronbach	N de elementos
0.768	10

Resultados de la Encuesta

La encuesta se llevó en línea al personal y representantes de los niños en la Academia Oficial de Fútbol Porto Ambato.

Ítem 1. Considero útil un sistema digital para gestionar mis pagos.

Tabla 5. Resultados del ítem 1.

Indicador	Frecuencia	Porcentaje
Totalmente en desacuerdo	6	7,7%
En desacuerdo	23	29,5%
Ni de acuerdo ni en desacuerdo	27	34,6%
De acuerdo	16	20,5%
Totalmente de acuerdo	6	7,7%
Total	78	100%

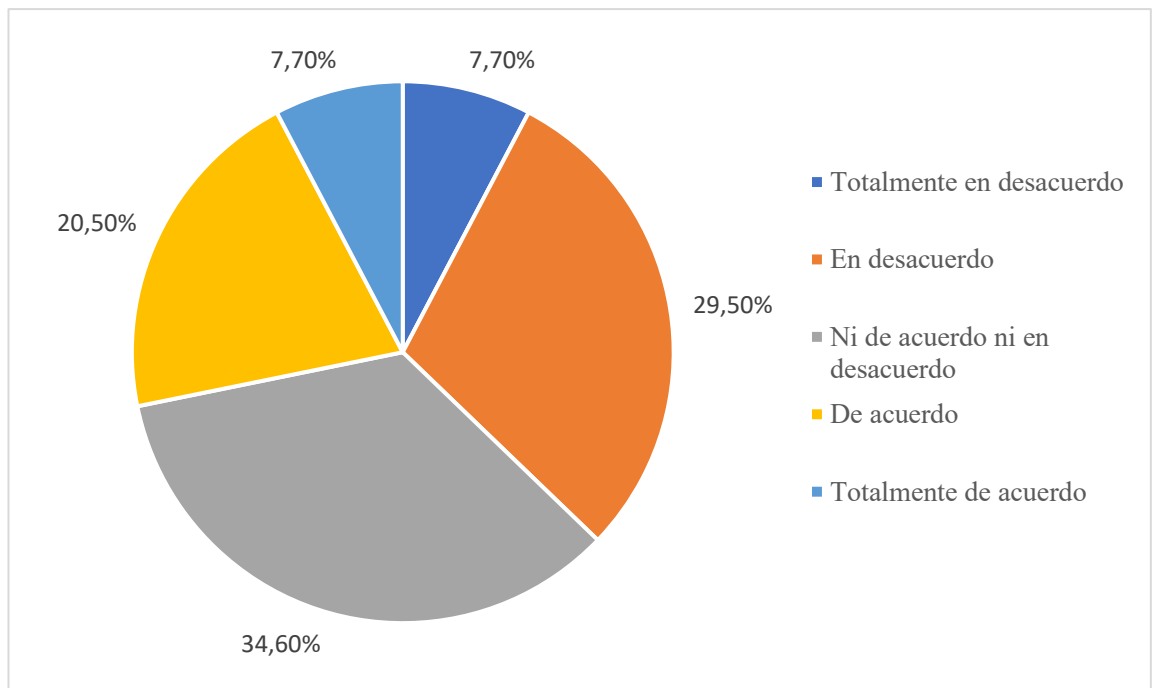


Figura 1. Resultados del ítem 1.

Análisis e Interpretación de Resultados

Con base en la Tabla 5 y la Figura 1, sobre el ítem 1 “Considero útil un sistema digital para gestionar mis pagos”, el 29,5% está en desacuerdo y el 7,7% totalmente en desacuerdo; el 34,6% se mantiene neutral; el 20,5% está de acuerdo y el 7,7% totalmente de acuerdo. En suma, el acuerdo alcanza 28,2% y el desacuerdo 37,2%.

Estos resultados muestran que la aceptación aún no es mayoritaria. Hace falta explicar mejor qué gana el usuario con el sistema, con información clara y una guía breve para el primer uso, una parte del grupo neutral y del desacuerdo podría pasar a una opinión favorable.

Ítem 2. Usar un sistema digital reduciría el tiempo que dedico al proceso de pagos.

Tabla 6. Resultados del ítem 2.

Indicador	Frecuencia	Porcentaje
Totalmente en desacuerdo	4	5,1%
En desacuerdo	19	24,4%
Ni de acuerdo ni en desacuerdo	26	33,3%
De acuerdo	23	29,5%
Totalmente de acuerdo	6	7,7%
Total	78	100%

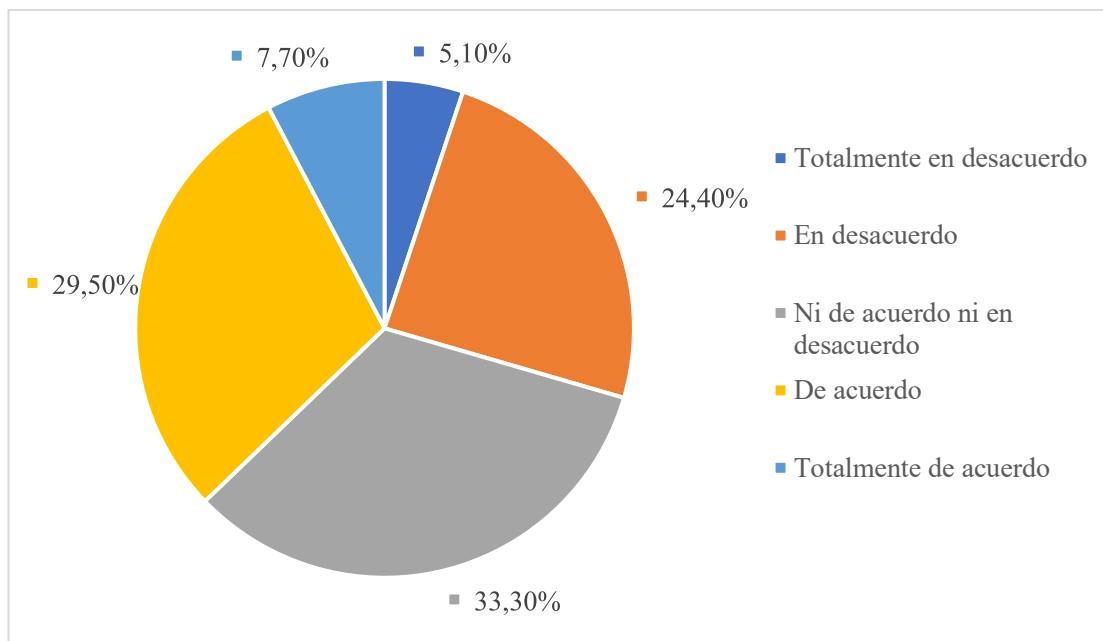


Figura 2. Resultados del ítem 2.

Análisis e Interpretación de Resultados

De acuerdo con la Tabla 6 y la Figura 2, sobre el ítem 2 “Usar un sistema digital reduciría el tiempo que dedico al proceso de pagos”, el 29,5% señaló estar de acuerdo y el 7,7% totalmente de acuerdo, sumando 37,2% a favor. El 33,3% se mantuvo neutral. El 24,4% indicó estar en desacuerdo y el 5,1% totalmente en desacuerdo, en total 29,5% en contra.

En resumen, hay más personas que ven ahorro de tiempo que quienes lo descartan, pero todavía existe un grupo grande que no se decide. Para avanzar, conviene mostrar ejemplos claros de cómo sería el proceso con él y ofrecer una guía breve de uso. Con información práctica y una prueba inicial acompañada, es probable que parte de la neutralidad se mueva hacia el acuerdo.

Ítem 3. Un sistema digital facilitaría el seguimiento de mi historial de pagos.

Tabla 7. Resultados del ítem 3.

Indicador	Frecuencia	Porcentaje
Totalmente en desacuerdo	4	5,1%
En desacuerdo	23	29,5%
Ni de acuerdo ni en desacuerdo	25	32,1%
De acuerdo	22	28,2%
Totalmente de acuerdo	4	5,1%
Total	78	100%

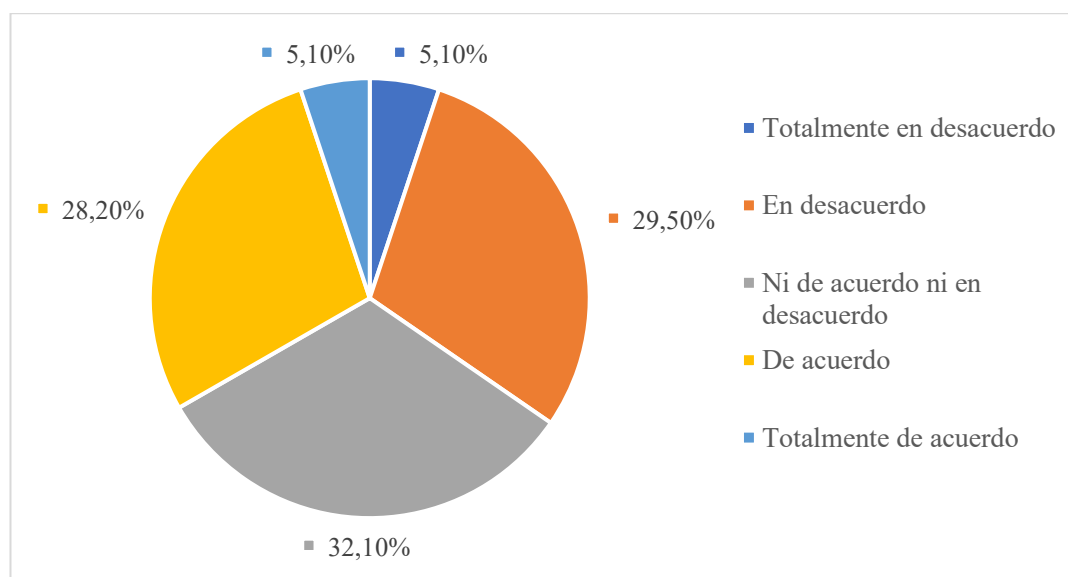


Figura 3. Resultados del ítem 3.

Análisis e Interpretación de Resultados

Como se observa en la Tabla 7 y la Figura 3, la opinión está dividida: 34,6% manifiesta desacuerdo, 33,3% expresa acuerdo y 32,1% se mantiene neutral respecto a que el sistema facilite el seguimiento del historial de pagos. No hay una postura dominante, lo que revela dudas sobre el beneficio concreto de esta funcionalidad.

Estos resultados sugieren reforzar la explicación práctica del historial, alertas de pagos, y un panel que muestre saldos y movimientos. Con demostraciones simples y acceso de prueba, es probable que parte de la neutralidad y del desacuerdo se traslade hacia el acuerdo, mejorando la aceptación de la herramienta.

Ítem 4. Consultar en línea estados y comprobantes de pago sería conveniente.

Tabla 8. Resultados del ítem 4.

Indicador	Frecuencia	Porcentaje
Totalmente en desacuerdo	4	5,1%
En desacuerdo	18	23,1%
Ni de acuerdo ni en desacuerdo	25	32,1%
De acuerdo	28	35,9%
Totalmente de acuerdo	3	3,8%
Total	78	100%

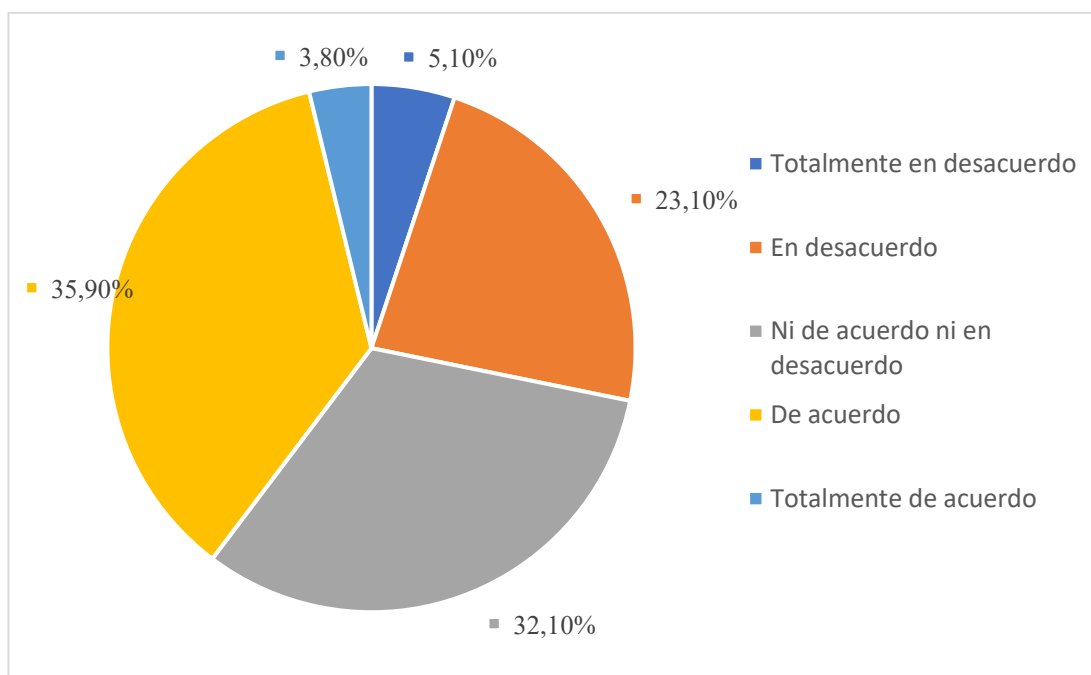


Figura 4. Resultados del ítem 4.

Análisis e Interpretación de Resultados

Sobre el ítem 4 “Consultar en línea estados y comprobantes de pago sería conveniente”, los datos muestran una tendencia favorable: 35,9% está de acuerdo y 3,8% totalmente de acuerdo, en total 39,7%. La neutralidad alcanza 32,1%, mientras que el desacuerdo suma 23,1% y el totalmente en desacuerdo 5,1%, en total 28,2%. En síntesis, hay más inclinación a favor que en contra, aunque todavía un tercio no se define.

Esto sugiere que la idea resulta atractiva, pero muchos aún no visualizan el beneficio. Conviene mostrar ejemplos sencillos: consulta 24/7, historial accesible y ahorro de tiempo frente a gestiones presenciales. Un tutorial breve y mensajes claros pueden ayudar a mover parte del grupo neutral hacia el acuerdo y consolidar la adopción.

Ítem 5. Las notificaciones del sistema me ayudarían a cumplir con los pagos a tiempo.

Tabla 9. Resultados del ítem 5.

Indicador	Frecuencia	Porcentaje
Totalmente en desacuerdo	3	3,8%
En desacuerdo	22	28,2%
Ni de acuerdo ni en desacuerdo	30	38,5%
De acuerdo	17	21,8%
Totalmente de acuerdo	6	7,7%
Total	78	100%

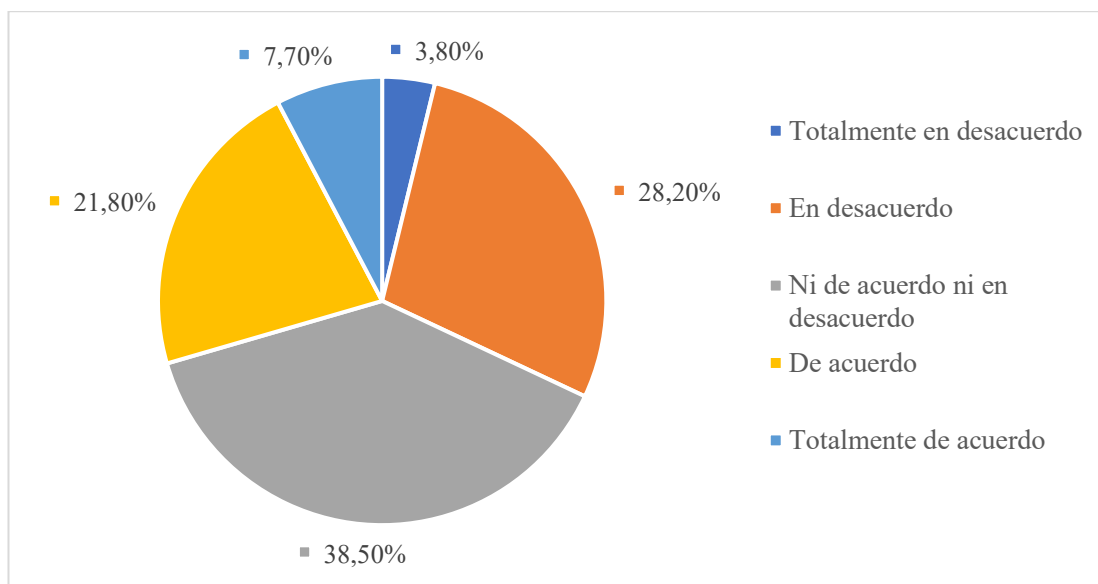


Figura 5. Resultados del ítem 5.

Análisis e Interpretación de Resultados

En el ítem “Las notificaciones del sistema me ayudarían a cumplir con los pagos a tiempo”, predomina la neutralidad con 38,5%. El acuerdo suma 29,5% y el desacuerdo 32,0%. Es decir, no hay una postura clara a favor; muchas personas aún no están seguras de que los recordatorios sean realmente útiles.

Esto sugiere trabajar la propuesta con ejemplos concretos: avisos antes del vencimiento, confirmaciones de pago y recordatorios solo cuando corresponde, sin saturar. También ayuda permitir que cada usuario elija cómo y cuándo recibirlos. Con mensajes útiles y configurables, es probable que parte del grupo neutral se incline hacia el acuerdo.

Ítem 6. Completar un pago implicaría pocos pasos.

Tabla 10. Resultados del ítem 6.

Indicador	Frecuencia	Porcentaje
Totalmente en desacuerdo	5	6,4%
En desacuerdo	18	23,1%
Ni de acuerdo ni en desacuerdo	33	42,3%
De acuerdo	19	24,4%
Totalmente de acuerdo	3	3,8%
Total	78	100%

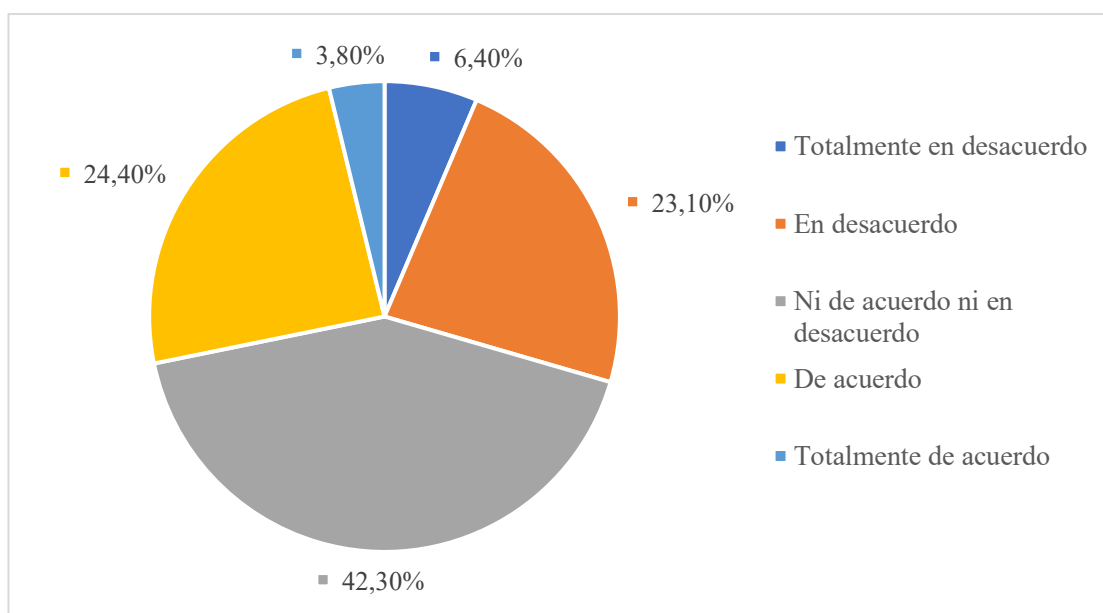


Figura 6. Resultados del ítem 6.

Análisis e Interpretación de Resultados

En “Completar un pago implicaría pocos pasos” predomina la neutralidad con 42,3%. El acuerdo reúne 24,4% y 3,8% (28,2% en total), mientras que el desacuerdo suma 23,1% y 6,4% (29,5%). En claro: la mayoría no se define y el resto está prácticamente dividido.

Esto indica que aún no se percibe con claridad cuán simple sería el flujo de pago. Conviene mostrar el proceso con ejemplos concretos: pocos clics, un solo formulario, autocompletado de datos, métodos de pago visibles y confirmación inmediata. Un video corto o capturas paso a paso pueden ayudar a que quienes hoy están neutrales o en desacuerdo vean el procedimiento como algo realmente sencillo.

Ítem 7. El proceso sería claro y ordenado de principio a fin.

Tabla 11. Resultados del ítem 7.

Indicador	Frecuencia	Porcentaje
Totalmente en desacuerdo	7	9,0%
En desacuerdo	24	30,8%
Ni de acuerdo ni en desacuerdo	27	34,6%
De acuerdo	19	24,4%
Totalmente de acuerdo	1	1,3%
Total	78	100%

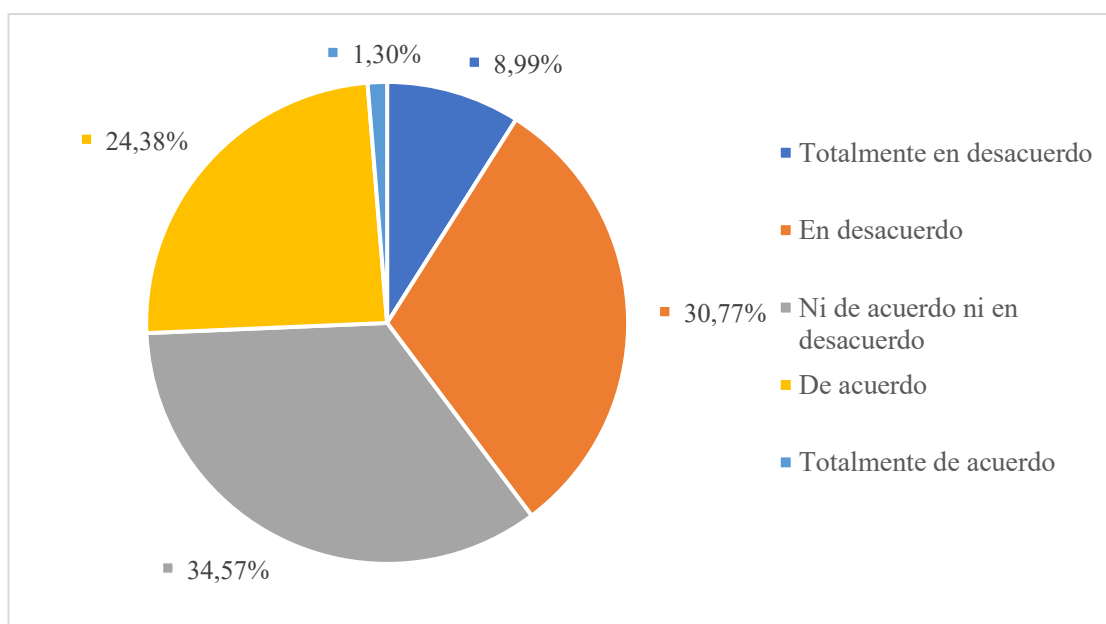


Figura 7. Resultados del ítem 7.

Análisis e Interpretación de Resultados

En “El proceso sería claro y ordenado de principio a fin” predomina la neutralidad con 34,6%. El desacuerdo alcanza 30,8% y el totalmente en desacuerdo 9,0%, mientras que el acuerdo llega a 24,4% y el totalmente de acuerdo con 1,3%. En conjunto, hay más dudas o desacuerdo que respaldo abierto a la idea de un proceso claro y ordenado.

Esto indica que la gente no termina de visualizar cómo sería el flujo. Conviene mostrar el recorrido completo con pasos simples, pantallas de ejemplo y mensajes claros: inicio, registro, pago, confirmación y recibo. Ayuda incluir validaciones visibles, un indicador de progreso y una guía rápida. Con ese tipo de mejoras y una demostración breve, parte de la neutralidad y del desacuerdo podría pasar a una opinión favorable.

Ítem 8. Tengo dificultades para que se registren con corrección mis pagos actualmente.

Tabla 12. Resultados del ítem 8.

Indicador	Frecuencia	Porcentaje
Nunca	4	5,1%
Casi nunca	32	41,0%
A veces	40	51,3%
Casi siempre	2	2,6%
Siempre	0	0%
Total	78	100%

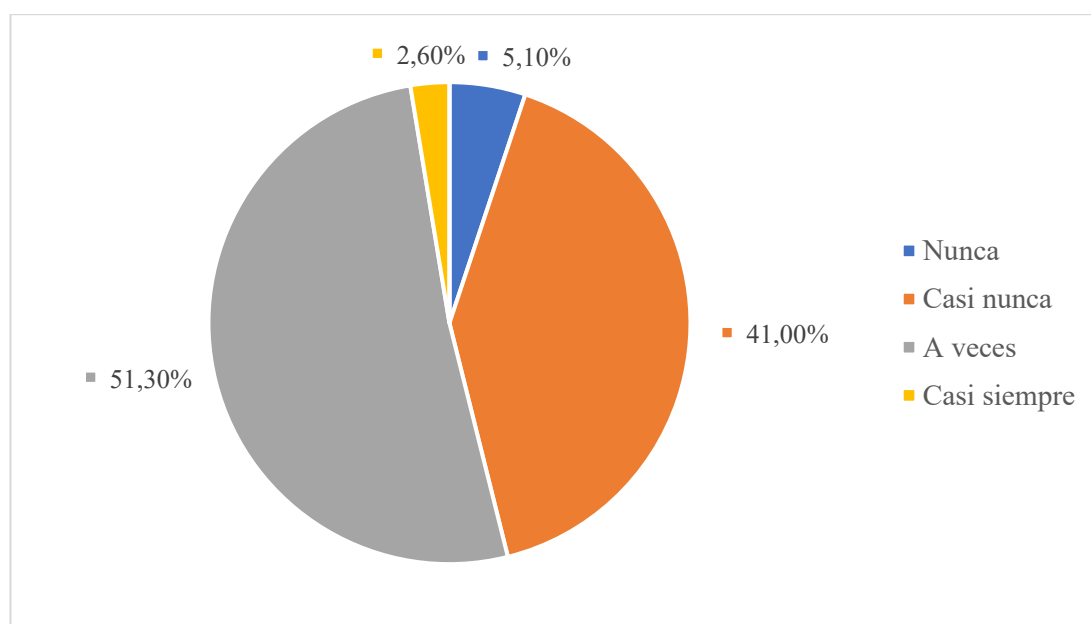


Figura 8. Resultados del ítem 8.

Análisis e Interpretación de Resultados

En “Tengo dificultades para que se registren correctamente mis pagos actualmente”, la mayoría señala problemas intermitentes: 51,3% respondió “a veces” y 2,6% “casi siempre”. En conjunto, 53,9% ha tenido algún inconveniente. Del otro lado, 46,1% indica que estas dificultades ocurren casi nunca o nunca.

El resultado muestra que el registro funciona, pero no de manera totalmente consistente. Conviene fortalecer el proceso con acciones simples: comprobantes estandarizados, verificación automática del pago, conciliación diaria y un canal de soporte claro para corregir errores rápido. Con estas mejoras, la frecuencia de incidencias debería bajar y la experiencia del usuario ser más confiable.

Ítem 9. Debo comunicarme con el personal para corregir errores en mis pagos actualmente.

Tabla 13. Resultados del ítem 9.

Indicador	Frecuencia	Porcentaje
Nunca	0	0%
Casi nunca	21	26,9%
A veces	45	57,7%
Casi siempre	12	15,4%
Siempre	0	0%
Total	78	100%

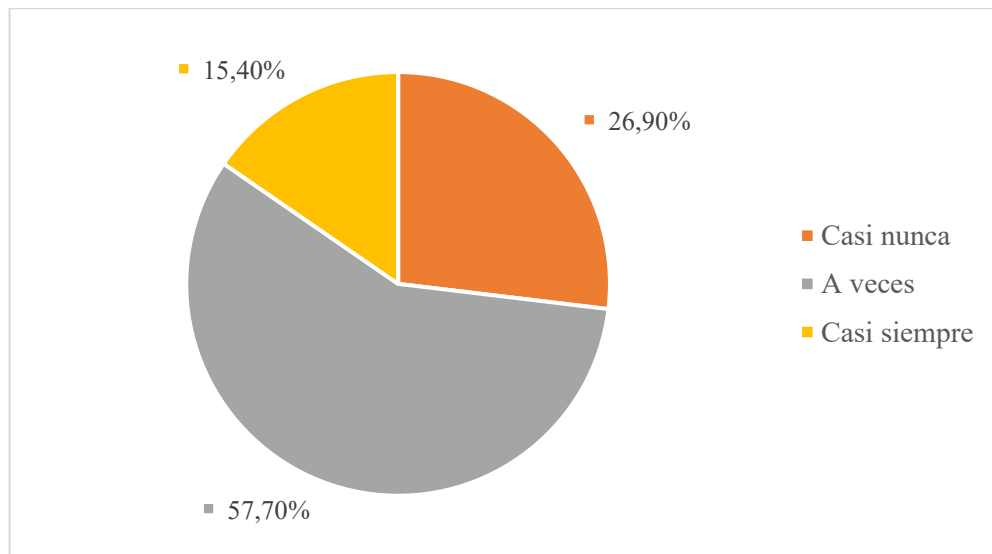


Figura 9. Resultados del ítem 9.

Análisis e Interpretación de Resultados

En “Debo comunicarme con el personal para corregir errores en mis pagos actualmente”, la mayoría indicó que a veces necesita hacerlo (57,7%) y un 15,4% dijo casi siempre. El 26,9% señaló casi nunca, mientras que nunca y siempre quedaron en 0%. En conjunto, siete de cada diez personas han tenido que contactar al personal en algún momento para resolver fallas.

Esto muestra que el proceso genera incidencias recurrentes que requieren apoyo. Para reducirlas conviene estandarizar los comprobantes, validar los datos antes de registrar el pago, habilitar una corrección en línea con número de caso y estado visible, y reforzar la conciliación diaria. Con estos ajustes, la necesidad de intervención del personal debería bajar y la experiencia del usuario mejorar.

Ítem 10. Encuentro demoras o confusiones al verificar el estado de mis pagos actualmente.

Tabla 14. Resultados del ítem 10.

Indicador	Frecuencia	Porcentaje
Nunca	29	37,2%
Casi nunca	49	62,8%
A veces	0	0%
Casi siempre	0	0%
Siempre	0	0%
Total	78	100%

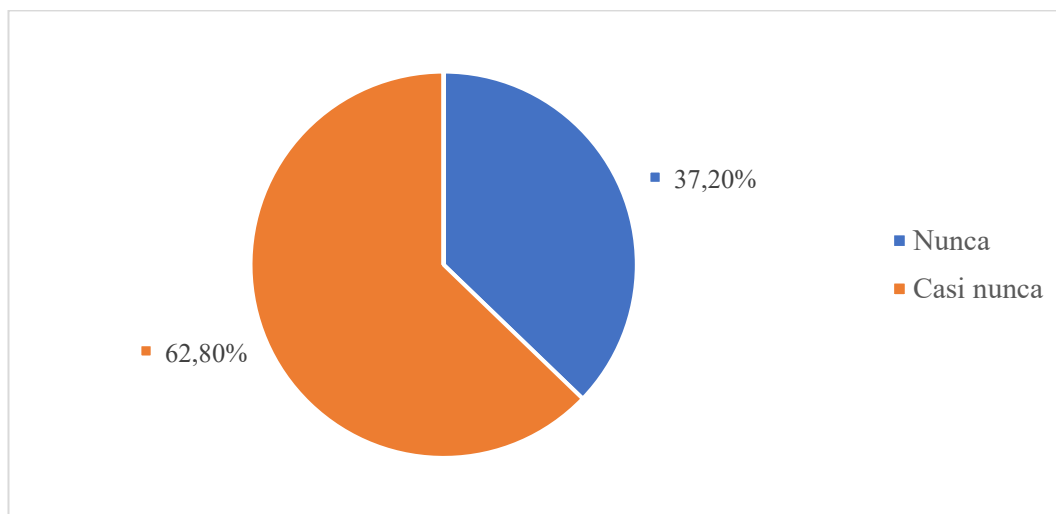


Figura 10. Resultados del ítem 10.

Análisis e Interpretación de Resultados

Los resultados del ítem 10 muestran una situación favorable: el 62,8% respondió casi nunca y el 37,2% nunca. No se registraron casos en las categorías “a veces”, “casi siempre” o “siempre”. En conjunto, toda la muestra señala que la situación evaluada rara vez ocurre.

Esto sugiere un proceso estable y con pocas incidencias. Para conservar ese desempeño conviene mantener controles básicos (verificación periódica, registro de eventos, trazabilidad de casos) y un canal de ayuda visible por si aparece alguna excepción. Con ese seguimiento, es probable que el indicador se mantenga en niveles muy bajos.

FICHA DE OBSERVACIÓN

Tabla 15. Resultados de la ficha de observación.

Fecha: martes 15 de octubre de 2025							
Hora de inicio: 08:00H							
Hora de fin: 09:00H							
Ubicación: Tungurahua – Ambato – Ficoa							
Aspectos para observar	Indicador	Cumplimiento (1 más bajo, 5 más alto)					Comentario
		1	2	3	4	5	
Uso de sistema para control de cobros	El personal administrativo utiliza un sistema donde anota y lleva el control de cobros	x					El personal no dispone de un sistema, por lo que se observó que se anota los cobros en cuadernos, Excel y otros envían el comprobante de la transferencia por teléfono WhatsApp, esto puede llevar a olvidos y errores.
Registro en el momento	Se registra al instante, sin postergar			x			En la caja sí, algunos pagos por transferencia no son registrados en el momento de realizarlos y puede llevar a posibles olvidos.
Verificación de datos	Se revisa monto, fecha y titular antes de guardar			x			Verificación visual, no hay validaciones automáticas.
Respaldo/backup	Existen copias digitales/escaneos del registro		x				Copia en pendrive y en los cuadernos si se llegan a perder o

							estropear se perdería la información.
Consulta de historial	Se puede buscar pagos por persona/mes			x			En cuaderno se demora mucho, en Excel es un poco más rápido.
Correcciones con trazabilidad	Queda registro de quién corrige		x				Se corrige sobre la misma hoja, no queda bitácora.
Tiempo por registro	Minutos desde que llega el pago hasta que queda anotado			x			Entre 5–10 min promedio (efectivo es más rápido).
Incidencias	Errores de monto/duplicados se minimizan			x			Ocurren duplicados y montos mal digitados ocasionalmente.
Soporte/atención	Hay ayuda rápida cuando algo falla		x				Se resuelve en caso de que el personal tenga disponibilidad.
Notificaciones/recordatorios	Avisos de vencimiento		x				No hay avisos automáticos, se contacta por WhatsApp manual o personalmente cuando el representante va a dejar al niño.
CONCLUSIONES El registro de cobros se maneja sin un sistema único: se combinan cuadernos, archivos en Excel y comprobantes enviados por WhatsApp. Esto provoca dispersión de la información, riesgo de olvidos y errores. El registro no siempre es inmediato (especialmente en transferencias), la verificación es solo visual (sin validaciones automáticas) y el respaldo es débil (pendrive/cuadernos, con riesgo de pérdida). Consultar historiales toma tiempo rápido en Excel, muy lento en cuaderno y las correcciones no dejan trazabilidad. El proceso demanda 5–10 minutos por pago y se observaron duplicados y montos mal digitados. El soporte depende de la disponibilidad del personal y no existen notificaciones automáticas; los recordatorios se hacen a mano o cuando el representante llega en persona.							

2.2.4 Procesamiento y análisis de datos

La encuesta se aplicó en línea a los actores involucrados de la Academia Oficial de Fútbol Porto Ambato con el propósito de medir la aceptación del sistema propuesto, identificar su valor percibido y detectar problemas del proceso actual sin sistema.

La información obtenida servirá para mejorar el flujo de registro y control de pagos e implementar una solución digital que apoye a la Academia. En general los procesos muestran, que se puede agregar un sistema que ayude con el proceso, sin embargo el alto índice de neutralidad indica que muchas personas no saben de qué va o como puede beneficiar esto a la academia y sus miembros. En términos generales, los resultados muestran aceptación moderada de la propuesta.

digital, pero con un bloque importante de neutralidad. Una parte de los encuestados ve utilidad en contar con un sistema para gestionar pagos y valora la posibilidad de consultar estados, aunque todavía muchos no visualizan con claridad cómo sería el uso cotidiano.

Respecto del proceso actual, se evidencian prácticas mixtas y manuales (cuadernos, hojas de cálculo y envío de comprobantes por mensajería), lo que dispersa la información y puede derivar en demoras, duplicados o montos mal ingresados. Además, varias personas señalaron que a veces deben comunicarse con el personal para corregir errores, lo que confirma la necesidad de estandarizar pasos, validar datos antes de registrar y contar con una trazabilidad básica de correcciones.

Sobre la experiencia de uso que se espera del nuevo sistema, el interés se concentra en disponer de historial 24/7 y un flujo de pago más simple. Las notificaciones se perciben como útiles por parte del grupo que ya está a favor; no obstante, persiste neutralidad en otros, lo que sugiere reforzar la comunicación con ejemplos prácticos y permitir que cada usuario configure el canal y la frecuencia de avisos.

Las observaciones en campo coinciden con la encuesta: el registro no siempre es inmediato, la verificación suele ser visual y los respaldos de la información son irregulares. Unificar el proceso en un solo sistema con validaciones automáticas y comprobante al instante resolvería los puntos críticos y facilitaría la consulta rápida por estudiante y por periodo.

El instrumento de medición utilizado mostró buena consistencia (alfa de Cronbach $\alpha = 0,768$ con 10 ítems), lo que respalda la lectura de los resultados y las decisiones derivadas. En síntesis, existe una necesidad real de ordenar el registro y control de pagos, junto con una disposición favorable a adoptar herramientas digitales, siempre que se prioricen simplicidad, claridad y acompañamiento en el primer uso. Esto abre una oportunidad concreta para modernizar la gestión y mejorar la experiencia de familias y personal en la Academia.

CAPÍTULO III. RESULTADOS Y DISCUSIÓN

En el este capítulo se presentarán y discutirán de forma sistemática estos resultados, se modelará el proceso actual de cobros, se identificarán sus principales problemas y riesgos, y se contrastarán con los requerimientos levantados en la investigación. A partir de ello, se describirá el proceso propuesto soportado por un sistema multiplataforma, detallando la arquitectura tecnológica seleccionada, la metodología de desarrollo empleada y las etapas de implementación que permitirán optimizar el control de pagos, mejorar la transparencia con los representantes y facilitar la toma de decisiones gerenciales.

3.1 Análisis de procesos y discusión de los resultados

A partir de la encuesta a padres, madres o representantes y la observación directa en la Academia Oficial de Fútbol Porto Ambato se evidenció que la gestión actual de matrículas y cobros se realiza casi por completo de forma manual, apoyada en cuadernos y archivos Excel, sin sistemas integrados ni validaciones automáticas. Esto provoca que los pagos dependan totalmente de la disponibilidad del administrador, que la consulta de saldos sea lenta y poco transparente, y que se generen con frecuencia errores de registro, montos mal calculados, pérdida de información y dificultades para emitir reportes consolidados por estudiante, categoría o periodo.

3.1.1 Modelo de procesos

a. Procedimiento manual

En la actualidad la Academia Oficial de Fútbol Porto Ambato Enfrenta diversas limitaciones en su proceso de registro y seguimiento de cobros de matrículas y mensualidades. A partir de la información recolectada se identifican debilidades que afectan la organización de la información, la eficiencia operativa y como se maneja la confianza en los pagos.

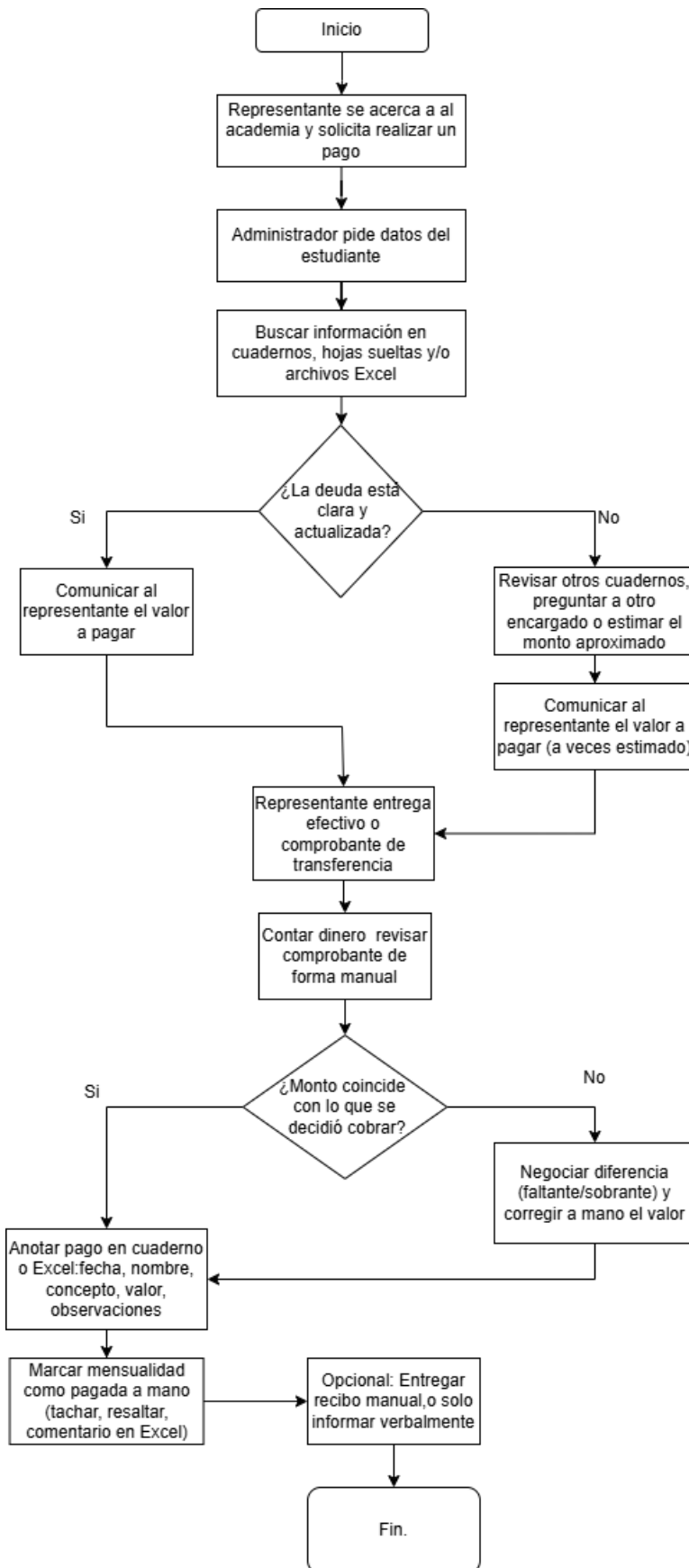


Figura 11. Diagrama de procesos actual de la gestión de pagos.

En la Figura 11 se describe el proceso actual que la Academia Oficial de Fútbol Porto Ambato realiza para el registro y cobro de mensualidades, desde el momento en que el representante se acerca a la academia y solicita realizar un pago, pasando por la búsqueda de información en cuadernos, hojas sueltas y/o archivos Excel, hasta el registro manual del abono y la marcación de la mensualidad como cancelada. A continuación, en la Tabla 16, se detallan los posibles problemas que conlleva continuar ejecutando este procedimiento de la manera actual.

Tabla 16. Descripción del proceso actual.

Paso	Proceso	Descripción
1	Inicio	Encabeza el procedimiento de cobro desde el momento en que se inicia la atención al representante.
2	Acercamiento del representante	El representante del estudiante se acerca a la academia y manifiesta su intención de realizar el pago de una mensualidad o concepto pendiente.
3	Solicitud de datos del estudiante	El administrador solicita el nombre y/o datos del estudiante para poder identificarlo dentro de los registros de la academia.
4	Búsqueda de información en registros	Se revisan cuadernos, hojas sueltas y/o archivos Excel para localizar el historial de pagos y el saldo actual del estudiante.
5	¿La deuda está clara y actualizada?	Se evalúa si la información encontrada permite conocer con precisión el valor pendiente a cancelar.
6	Revisión adicional de registros	Cuando la deuda no está clara, se consultan otros cuadernos, se pregunta a otro encargado o se estima el monto aproximado a cobrar
7	Comunicación del valor a pagar	Se informa al representante el valor que debe cancelar; en algunos casos el monto es aproximado por la falta de información consolidada y actualizada.
8	Entrega del pago	El representante entrega el dinero en efectivo o presenta el comprobante de transferencia bancaria correspondiente.
9	Conteo y revisión manual del pago	El administrador cuenta el dinero y/o revisa el comprobante de transferencia de manera manual, verificando el monto y el concepto del pago.
10	¿Monto coincide con lo que se decidió cobrar?	Se contrasta el monto recibido con el valor comunicado al representante para determinar si existe faltante o sobrante.
11	Negociación de diferencias	En caso de discrepancias, se negocia la diferencia con el representante y se corrige manualmente el valor que será registrado como pago.
12	Registro del pago	Una vez validado el valor, se anota el pago en cuaderno o archivo Excel, registrando fecha, nombre del estudiante, concepto, monto y observaciones pertinentes.
13	Marcación de mensualidad pagada	Se marca la mensualidad como cancelada de forma manual mediante tachaduras, resaltados o comentarios en los registros físicos o en el archivo Excel.
14	Emisión de comprobante	De forma opcional, se entrega un recibo manual al representante o únicamente se le informa de manera verbal que el pago ha sido registrado
15	Fin	Se da por concluido el proceso de cobro y el ciclo se reinicia con la siguiente operación cuando otro representante solicita realizar un pago.

3.1.2 Síntesis de problemas y requisitos del sistema

El análisis conjunto de los resultados de la encuesta aplicada a padres, madres o representantes y del flujo actual de gestión de cobros en la Academia Oficial de Fútbol Porto Ambato permite identificar una serie de problemas recurrentes que afectan la eficiencia operativa, la claridad de la información y la transparencia en el registro de pagos. Entre los principales hallazgos destacan la dispersión de los datos en cuadernos, hojas de cálculo y mensajes de mensajería, la ausencia de validaciones automáticas, la dificultad para conocer en tiempo real el estado de las mensualidades y la carencia de reportes consolidados para la toma de decisiones administrativas.

Estas debilidades se traducen en riesgos concretos para la academia: posibilidad de errores en montos y saldos, duplicidad o pérdida de registros, dependencia excesiva de la memoria y experiencia del administrador, incremento del tiempo de atención en el momento del cobro y limitaciones para planificar financieramente en función de la cartera de pagos pendientes. Desde la perspectiva de los representantes, la falta de un canal único y confiable para consultar sus obligaciones económicas genera incertidumbre, necesidad de realizar consultas frecuentes y potenciales desacuerdos frente a los valores registrados.

El sistema debe reunir en un solo lugar toda la información de estudiantes, matrículas, mensualidades y pagos, para evitar el uso de cuadernos o archivos separados. También debe permitir registrar estudiantes y representantes, relacionarlos con sus categorías, controlar las matrículas y generar las mensualidades con sus respectivos estados. Además, el sistema tiene que facilitar el registro de pagos y mantener un historial completo que permita ver la información sin depender de documentos físicos. Como apoyo al trabajo administrativo, debe emitir comprobantes, generar reportes y manejar usuarios con distintos niveles de acceso.

En cuanto a los requisitos no funcionales, se espera que la plataforma sea fácil de usar, que proteja los datos personales y económicos, y que mantenga la información ordenada y coherente, el sistema debería permitir añadir nuevas funciones en el futuro sin complicar su funcionamiento general.

3.2 Diseño del proceso automatizado de cobros

Con base en los problemas identificados en el proceso manual y en los requisitos definidos en la sección anterior, se propone un proceso automatizado de gestión de cobros soportado por el sistema multiplataforma. Este nuevo flujo de trabajo busca centralizar la información en una base de datos única, reducir la dependencia de registros físicos, incorporar validaciones automáticas y ofrecer a los usuarios internos y externos una consulta ágil y transparente del estado de los pagos.

En este enfoque, el representante puede revisar el estado de cuenta y los valores pendientes desde la aplicación, mientras que el administrador registra y valida los pagos directamente en el sistema, el cual se encarga de actualizar de forma automática las mensualidades, recalcular saldos y conservar el historial de movimientos por estudiante.

Con base en las debilidades identificadas, se propone un nuevo flujo de trabajo soportado por el sistema multiplataforma que se visualiza en la Fig. 12. En este proceso optimizado, el representante puede consultar el estado de cuenta y generar comprobantes de pago desde una aplicación web o móvil; el administrador registra o valida los pagos directamente en el sistema, que actualiza de manera automática las mensualidades correspondientes, el saldo pendiente y el historial del estudiante.

Modelo de procesos automatizados

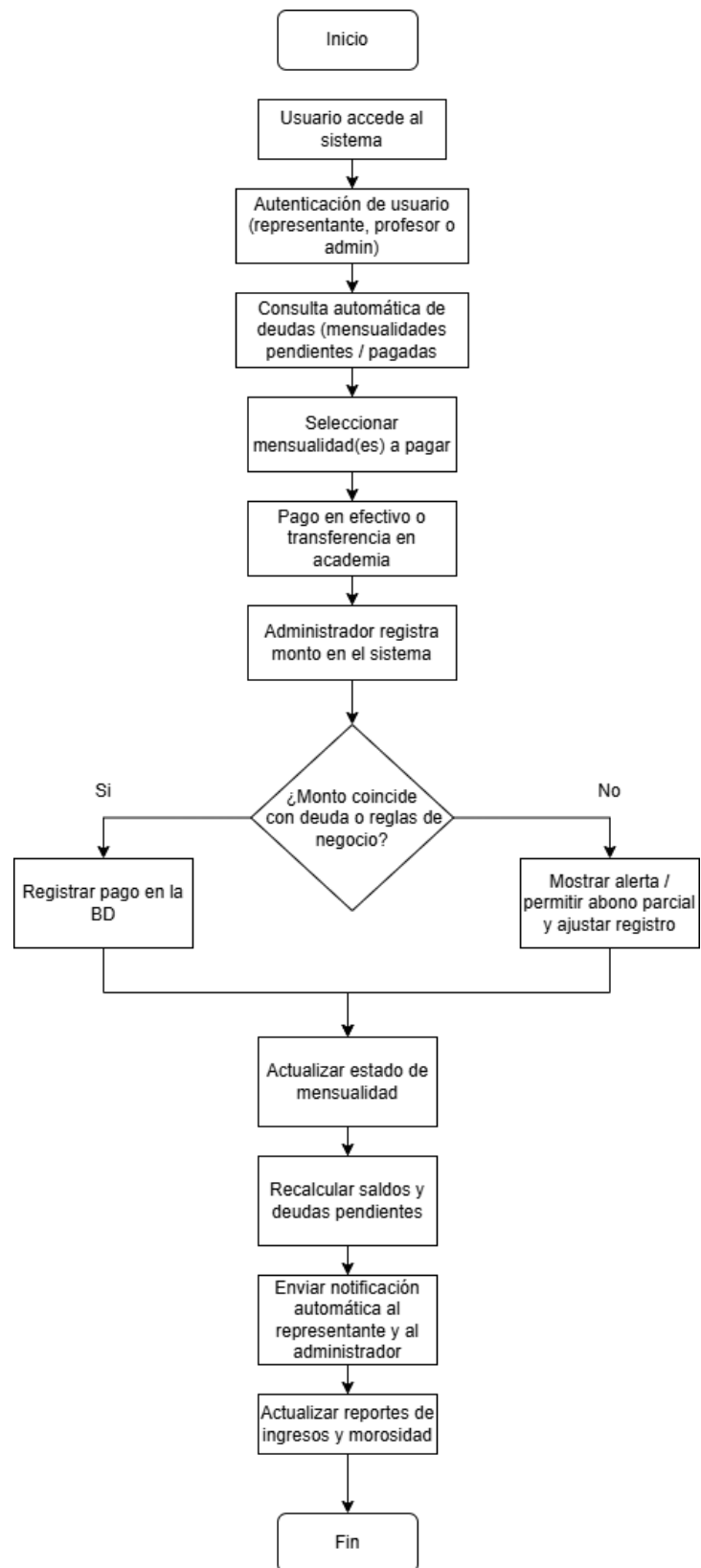


Figura 12. Diagrama de procesos automatizado de la gestión de pagos

Tabla 17. Descripción del proceso automatizado.

Paso	Proceso	Descripción
1	Inicio	Encabeza el procedimiento automatizado de cobro desde el momento en que se va a registrar un pago dentro del sistema.
2	Usuario accede al sistema	El representante, profesor o administrador ingresa al sistema y accede al módulo de pagos o estado de cuenta.
3	Autenticación de usuario	El sistema solicita credenciales y valida la identidad y el rol del usuario (representante, profesor o administrador) antes de permitir el acceso a la información.
4	Consulta automática de deudas	El sistema consulta la base de datos y muestra de forma automática las mensualidades generadas, pagadas y pendientes para el estudiante seleccionado.
5	Seleccionar mensualidad(es) a pagar	El usuario elige una o varias mensualidades que desea cancelar, según el estado de cuenta mostrado por el sistema.
6	Pago en efectivo o transferencia en academia	El representante realiza el pago en la academia, entregando efectivo o el comprobante de transferencia bancaria al administrador.
7	Administrador registra monto en el sistema	El administrador ingresa en el sistema el monto recibido, el medio de pago y los datos necesarios para asociar el pago al estudiante y a la(s) mensualidad(es) seleccionada(s).
8	¿Monto coincide con deuda o reglas de negocio?	El sistema valida automáticamente si el valor registrado coincide con la deuda configurada o con las reglas de negocio.
9	Registrar pago en la BD	Cuando el monto es correcto o se ajusta según las reglas de negocio, el sistema registra la transacción de pago en la base de datos.
10	Mostrar alerta / permitir abono parcial y ajustar registro	Si el monto no coincide, el sistema muestra una alerta y permite registrar un abono parcial o corregir el valor conforme a las políticas de la academia.
11	Actualizar estado de mensualidad	El sistema actualiza automáticamente el estado de la mensualidad (pagada, parcialmente pagada, pendiente) según el pago registrado.
12	Recalcular saldos y deudas pendientes	Se recalculan de manera automática los saldos del estudiante, así como las deudas pendientes totales o por periodo.
13	Enviar notificación automática al representante y al administrador	El sistema envía una notificación confirmando el pago y mostrando el saldo actualizado.
14	Actualizar reportes de ingresos y morosidad	Los reportes de ingresos y de morosidad se actualizan en tiempo real para la gestión administrativa y financiera de la academia.
15	Fin	Se da por concluido el proceso automatizado de cobro y el sistema queda disponible para registrar una nueva transacción.

El proceso automatizado de cobros se representa en la Figura 12, se detalla en la Tabla 17, este modelo organiza las actividades desde el momento en que un usuario accede al sistema hasta la actualización de saldos y el envío de notificaciones de confirmación.

El flujo inicia con el acceso al sistema por parte del representante, profesor o administrador, quien ingresa al módulo de pagos o estado de cuenta utilizando la

aplicación. A continuación, el sistema realiza la autenticación de usuario, validando las credenciales y el rol (representante, profesor o administrador) antes de permitir el acceso a la información sensible de cobros.

Una vez autenticado, el sistema ejecuta una consulta automática de deudas, mostrando las mensualidades generadas, pagadas y pendientes para el estudiante seleccionado, a partir de los registros almacenados en la base de datos. El usuario puede entonces seleccionar una o varias mensualidades a cancelar, de acuerdo con su estado de cuenta y sus posibilidades de pago.

En el caso de pagos realizados en la academia, el representante entrega el efectivo o el comprobante de transferencia al administrador, quien registra el monto, el medio de pago y la mensualidad directamente en el sistema. A partir de esta información, el sistema realiza una validación automática del monto: contrasta el valor registrado con la deuda configurada o con las reglas de negocio.

Cuando el valor es correcto o se ajusta a las reglas, el sistema registra la transacción de pago en la base de datos, generando el asiento correspondiente.

El sistema actualiza automáticamente el estado de las mensualidades afectadas (pagada, pendiente), asegurando que el estado de cuenta refleje en tiempo real la situación económica del estudiante. De forma simultánea, se ejecuta un proceso de recalcular de saldos y deudas pendientes, tanto a nivel del estudiante como del total adeudado por periodo, categoría o subcategoría.

El sistema envía una notificación automática al representante y, de ser necesario, al administrador, confirmando el pago registrado y mostrando el saldo actualizado. Este mecanismo de confirmación contribuye a la transparencia del proceso y reduce la necesidad de consultas presenciales o aclaraciones posteriores. Además, al mantenerse toda la información en una base de datos centralizada, se facilita la generación posterior de reportes de ingresos, morosidad y comportamiento de pago.

3.2.1 Comparación entre el proceso actual y el proceso automatizado

Tabla 18. Tabla comparativa con el proceso actual y optimizado.

Aspecto	Proceso actual (manual)	Proceso automatizado	Mejora principal
Fuente de información	Registros en cuadernos, hojas de cálculo y mensajes dispersos	Información centralizada en una base de datos única, consultable desde el sistema	Eliminación de la dispersión de datos y mayor confiabilidad de la información
Determinación de la deuda	Búsqueda manual en varios registros, a veces se recurre a estimaciones	Consulta directa a la base de datos que muestra mensualidades generadas, pagadas y pendientes en tiempo real	Reducción de demoras y errores de cálculo
Validación de montos	Verificación y ajustes realizados únicamente por el administrador, con cálculos manuales	Reglas de negocio implementadas en el sistema que validan montos y permiten abonos parciales cuando aplica	Disminución del margen de error humano y estandarización del tratamiento de cada transacción
Generación de reportes	Requiere recopilar y tabular datos de distintas fuentes de forma manual	Reportes generados a partir de registros estructurados en la base de datos	Agilidad para obtener reportes de ingresos, morosidad y comportamiento de pago
Notificaciones a representantes	Recordatorios y confirmaciones enviados de forma informal o esporádica	Notificaciones automáticas de vencimientos y pagos registrados	Mayor transparencia, comunicación oportuna y apoyo a la reducción de la morosidad

Tras comparar entre el diagrama del proceso actual en la Figura 11 y el diagrama del proceso automatizado en la Figura 12, se observan mejoras sustanciales en la gestión de cobros. Mientras el esquema manual depende de cuadernos, archivos Excel y comunicación informal para determinar saldos, el modelo automatizado concentra toda la información en un sistema único, al que pueden acceder en tiempo real los usuarios autorizados.

En el flujo tradicional, la determinación de la deuda exige revisar múltiples registros y, en ciertas ocasiones, recurrir a estimaciones aproximadas. Esto genera demoras en la atención y aumenta la posibilidad de errores de cálculo. En cambio, en el proceso automatizado la consulta de deudas y pagos se realiza mediante una consulta directa a la base de datos, que presenta de forma clara las mensualidades generadas, canceladas y pendientes.

Otra diferencia relevante se relaciona con la gestión de validaciones. En el esquema manual, la verificación de montos y el ajuste de diferencias recae exclusivamente en el administrador, quien se apoya en cálculos manuales. Esta dinámica incrementa la dependencia de una sola persona y eleva el riesgo de equivocaciones. En el modelo automatizado, las validaciones se implementan como reglas de negocio dentro del sistema, el cual verifica si el pago coincide con la deuda registrada o, en su defecto, si corresponde tratarlo como un abono parcial, reduciendo el margen de error y homogeneizando el tratamiento de los casos.

De igual forma, el flujo manual presenta dificultades para generar reportes consolidados de ingresos y morosidad, ya que la información se encuentra fragmentada y su integración implica un trabajo adicional de recopilación y tabulación. Por el contrario, en el proceso automatizado la estructura de la base de datos y el registro sistemático de las transacciones permiten obtener reportes por periodo, categoría, subcategoría o estudiante con menor esfuerzo, lo que facilita la toma de decisiones administrativas.

El manejo de notificaciones constituye otra mejora importante. En el esquema actual, los recordatorios y confirmaciones dependen de mensajes informales o comunicaciones esporádicas. En el modelo automatizado se integran notificaciones automáticas que informan tanto los vencimientos próximos como los pagos registrados, aportando transparencia al proceso y contribuyendo a disminuir la morosidad.

En conjunto, el proceso automatizado no solo reemplaza actividades manuales por operaciones digitales, sino que reconfigura el flujo de trabajo incorporando validaciones, trazabilidad, acceso multiusuario y generación de reportes. De esta manera, se alinea con los requisitos funcionales y no funcionales establecidos previamente y sienta las bases para la definición de la arquitectura tecnológica y de los módulos del sistema.

3.3 Arquitectura del sistema y tecnologías seleccionadas

El sistema propuesto para el seguimiento y control de cobros en la Academia Oficial de Fútbol Porto Ambato se concibe bajo una arquitectura cliente–servidor de tres capas lógicas: capa de presentación, capa de lógica de negocio y capa de datos. Este enfoque permite separar responsabilidades, facilitar el mantenimiento del software y posibilitar la evolución futura de la solución sin afectar la estabilidad del núcleo de datos.

Sobre esta base, se implementa un framework multiplataforma en la capa de presentación, una API REST en la capa de lógica de negocio y un gestor de base de datos relacional en la capa de datos. La combinación de estas tecnologías responde a los requisitos funcionales y no funcionales identificados previamente, en particular la necesidad de acceso desde distintos dispositivos, integridad de la información, seguridad y escalabilidad.[14]

3.3.1 Arquitectura general del sistema

La arquitectura del sistema puede resumirse en tres capas lógicas:

- **Capa de presentación:** formada por la aplicación desarrollada en Flutter, que se despliega inicialmente como aplicación web y se encuentra preparada para generar versiones móviles. Desde esta capa interactúan el administrador, los profesores y los representantes, según los permisos que tengan asignados.
- **Capa de lógica de negocio:** compuesta por un servidor de aplicaciones basado en Node.js y Express, que expone servicios REST. Aquí se implementan las reglas de generación de mensualidades, validación de pagos, actualización de estados, emisión de comprobantes y registro de auditoría.
- **Capa de datos:** constituida por el gestor de base de datos PostgreSQL, en el que se almacena de manera estructurada toda la información relacionada con las academias, usuarios, estudiantes, matrículas, mensualidades, pagos y bitácoras de acciones.

La comunicación entre la aplicación se base en la capa de presentación y la capa lógica la cual se realiza mediante solicitudes HTTP/HTTPS, utilizando JSON como formato de intercambio. De esta forma, la interfaz permanece desacoplada de la lógica de negocio y del motor de bases de datos, lo que permite modificar o evolucionar cada capa de manera independiente.

3.3.2 Normativas y estándares considerados

Para guiar el diseño del sistema se consideraron lineamientos de estándares internacionales relacionados con descripción de arquitectura y calidad del producto software. No obstante, debido al alcance académico del proyecto, al tiempo disponible y a la prioridad de implementar primero los módulos críticos, estos estándares se aplicaron de forma parcial como referencia de buenas prácticas, sin ejecutar un proceso formal de evaluación o certificación.

En particular, la documentación y descripción de la arquitectura se alineó con los conceptos de *architecture description* propuestos por **ISO/IEC/IEEE 42010:2022**, mientras que la definición de atributos de calidad se relacionó con el modelo de calidad de **ISO/IEC 25010:2023**, priorizando los atributos más relevantes para el contexto de la academia (seguridad, usabilidad y mantenibilidad).

3.3.3 Análisis de frameworks multiplataforma

Para la capa de presentación se analizó el uso de distintos frameworks multiplataforma, entre ellos React Native, Flutter, Ionic y .NET MAUI/Xamarin. Todos ellos permiten compartir buena parte del código entre plataformas, pero difieren en el lenguaje que utilizan, la forma en que construyen la interfaz y el rendimiento que ofrecen [15],[16].

Tabla 19. Comparación de frameworks multiplataforma.

Framework	Lenguaje principal	Enfoque de interfaz / renderizado	Fortalezas principales	Aspectos menos favorables para este proyecto
Flutter	Dart	Renderiza la interfaz con su propio motor gráfico; compila a código nativo	Alto rendimiento, apariencia uniforme en distintas plataformas, hot reload, una sola base de código para web y móvil	Lenguaje menos conocido, algunas integraciones nativas requieren configuración adicional

Framework	Lenguaje principal	Enfoque de interfaz / renderizado	Fortalezas principales	Aspectos menos favorables para este proyecto
React Native	JavaScript + React	Utiliza componentes nativos a través de un puente con JavaScript	Amplio ecosistema de librerías, aprovechamiento de experiencia previa en React	El uso del “bridge” puede afectar el rendimiento en aplicaciones muy exigentes
Ionic	TypeScript + Angular /React/ Vue	Se apoya en tecnologías web (HTML, CSS, JS) dentro de un WebVie	Adecuado para equipos con fuerte perfil web; curva de aprendizaje relativamente baja	Interfaz menos cercana a lo nativo; el desempeño puede verse limitado en vistas muy interactivas
.Net MAUI/Xamarin	C# y .NET	Abstrae componentes nativos mediante el entorno .NET	Integración sólida con el ecosistema Microsoft; buen acceso a APIs nativa	Más orientado a organizaciones que ya trabajan con stack Microsoft; menor presencia en contextos educativos pequeños

Aunque todas las alternativas son válidas, las necesidades específicas del proyecto como panel administrativo web, posibilidad de acceso móvil, manejo intensivo de tablas y formularios de datos esto hace que Flutter resulte especialmente adecuado[15]-[17].

Las razones principales para adoptar Flutter son las siguientes:

- Permite compartir la misma base de código para la aplicación web y para las futuras aplicaciones móviles, lo que reduce el esfuerzo de desarrollo y mantenimiento.
- Ofrece un rendimiento estable en pantallas con listas y tablas extensas, como las que se requieren para visualizar estudiantes, mensualidades y pagos.
- Proporciona una interfaz visual consistente entre navegadores y sistemas operativos, lo que simplifica el diseño y las pruebas.
- Incorpora herramientas que aumentan la productividad del desarrollo, como el recargado en caliente de cambios en la interfaz.
- Cuenta con una comunidad activa y en crecimiento, lo que facilita encontrar ejemplos, soluciones y paquetes reutilizables para acelerar la construcción de módulos.

Por ende, Flutter se adopta como tecnología base para la capa de presentación del sistema multiplataforma.

3.3.4 Framework multiplataforma Flutter

Flutter es un kit de desarrollo de interfaces creado inicialmente por Google con el objetivo de simplificar la construcción de aplicaciones nativas para diferentes plataformas a partir de una sola base de código. A diferencia de otros enfoques que se apoyan en componentes nativos o en vistas web incrustadas, Flutter utiliza su propio motor de renderizado para dibujar la interfaz directamente sobre la pantalla, lo que le permite ofrecer un alto nivel de control visual y un comportamiento consistente entre sistemas operativos [17].

a. Categorías de widgets

- Widgets estructurales y de disposición, como filas, columnas, contenedores, pilas o cuadrículas, que definen cómo se distribuyen los elementos en la pantalla.
- Widgets interactivos, como botones, campos de texto, selectores, interruptores y listas desplazables, que permiten capturar la entrada del usuario y responder a eventos.
- Widgets de estilo y tematización, que controlan colores, tipografías, tamaños, espacios y formas, permitiendo definir temas globales y estilos coherentes en toda la aplicación.
- Widgets especializados, orientados a animaciones, transiciones, efectos visuales, iconografía y composición avanzada.

El modelo distingue entre widgets sin estado (stateless) y widgets con estado (stateful). Los primeros representan elementos cuya configuración no cambia durante su ciclo de vida, mientras que los segundos contienen datos internos que pueden variar en respuesta a interacciones o actualizaciones provenientes de la lógica de negocio. Esta separación clarifica qué partes de la interfaz pueden actualizarse y cuáles se mantienen constantes, simplificando el seguimiento del flujo de datos.

En cuanto a la navegación, Flutter ofrece mecanismos para gestionar rutas y pilas de pantallas. A través del sistema de navegación es posible avanzar, retroceder, reemplazar vistas o establecer esquemas más complejos, como navegación por pestañas, paneles con secciones anidadas o flujos asistidos paso a paso. En proyectos de tamaño medio o grande es habitual combinar la navegación básica con patrones de organización de módulos por funcionalidad, agrupando pantallas y componentes relacionados en conjuntos coherentes.

Otro aspecto central de la capa de presentación es el diseño responsivo. Flutter proporciona herramientas para adaptar la interfaz al espacio disponible, detectando características del dispositivo como tamaño y orientación de la pantalla. Con estos recursos es posible mostrar una distribución rica y densa de información en pantallas amplias, y al mismo tiempo ofrecer vistas simplificadas y optimizadas en dispositivos móviles, conservando el mismo código fuente. A ello se suman consideraciones de accesibilidad, como tamaños de fuente configurables, contraste suficiente entre elementos y soporte para distintas interacciones de entrada (ratón, teclado, táctil).

En lo que respecta a la gestión del estado de la interfaz y su interacción con la lógica de negocio, Flutter admite múltiples enfoques. Desde soluciones sencillas basadas en `setState`, adecuadas para pantallas pequeñas, hasta patrones más estructurados que se apoyan en proveedores de estado, arquitecturas reactivas o separación estricta entre presentación y dominio, como ocurre en patrones. La elección del patrón depende del tamaño del proyecto y del grado de desacoplamiento que se desee entre la capa visual y la lógica de negocio [18].

b. Ventajas y desventajas de Flutter

Tabla 20. Ventajas y desventajas.

Aspecto	Ventajas	Desventajas
Productividad en el desarrollo	Recarga en caliente, componentes reutilizables y una sola base de código para varias plataformas	Requiere ajustar la forma de trabajar si el equipo viene de entornos tradicionales nativos o web
Experiencia de usuario (UI/UX)	Interfaz fluida, animaciones suaves y control detallado sobre el diseño	Si no se cuida el diseño, es fácil terminar con interfaces sobrecargadas o poco consistentes
Consistencia entre plataformas	Misma apariencia y comportamiento en móvil, web y escritorio	Puede ser necesario adaptar ciertos detalles para que la app se sienta “natural” en cada plataforma

Aspecto	Ventajas	Desventajas
Lenguaje y curva de aprendizaje	Sintaxis clara, tipado estático y buen soporte de herramientas	Dart es menos conocido que JavaScript o C#, lo que implica un periodo de adaptación
Integración con funcionalidades nativas	Amplia oferta de paquetes para acceder a cámara, archivos, notificaciones, etc.	Algunas integraciones específicas pueden requerir escribir código nativo o configurar canales propios
Rendimiento	Buen rendimiento en listas, formularios complejos y vistas con muchas animaciones	El tamaño inicial del ejecutable y ciertas particularidades en web pueden ser mayores que en apps nativas minimalistas
Mantenimiento y evolución	Arquitecturas modulares y patrones de gestión de estado facilitan escalar el proyecto	La mala elección de patrón de estado puede complicar el código y volverlo difícil de mantener
Comunidad y ecosistema	Documentación abundante, ejemplos, paquetes de terceros y comunidad activa	Algunos paquetes no están igual de maduros o mantenidos que en otros ecosistemas más antiguos

La Tabla 20 resume los principales puntos fuertes y las limitaciones que conviene tener presentes al utilizar Flutter en la capa de presentación. Desde el punto de vista productivo, trabajar con una única base de código y disponer de recarga en caliente permite avanzar con rapidez en el diseño y ajuste de pantallas, algo especialmente útil cuando se están afinando formularios, tablas de datos o flujos de navegación. Además, el modelo basado en widgets facilita reutilizar componentes y mantener una línea visual coherente en toda la aplicación.

En términos de experiencia de usuario, Flutter ofrece un control fino sobre el diseño y las animaciones, lo que posibilita interfaces fluidas y modernas. No obstante, esa flexibilidad implica también una responsabilidad: si no se planifica bien la estructura visual, es fácil saturar la pantalla con elementos o generar interfaces poco claras. Por ello, suele ser recomendable apoyarse en guías de diseño, sistemas de estilos y patrones de navegación bien definidos.

La curva de aprendizaje es otro punto para considerar. Aunque Dart resulta relativamente sencillo para quienes ya tienen experiencia en programación orientada a objetos, sigue siendo un lenguaje menos difundido que JavaScript o C#, por lo que el equipo necesita un periodo de adaptación. Algo similar ocurre con los patrones de gestión de estado: existen varias alternativas y la elección de una u otra puede influir bastante en la claridad del código a medio y largo plazo.

En cuanto a las integraciones, Flutter dispone de numerosos paquetes que resuelven necesidades típicas (acceso a cámara, almacenamiento local, notificaciones,

autenticación, etc.). Sin embargo, en escenarios que demandan acceso muy específico a funcionalidades del sistema operativo puede ser necesario escribir pequeños módulos nativos o configurar canales de comunicación adicionales, lo que añade algo de complejidad técnica.

El ecosistema y la comunidad se han ido consolidando con el tiempo. Hay ejemplos, plantillas y buenas prácticas disponibles, lo que facilita encontrar soluciones a problemas habituales. Aun así, no todos los paquetes externos tienen el mismo grado de madurez o mantenimiento, de modo que es importante seleccionar las dependencias con criterio y revisarlas periódicamente.

c. Versiones de Flutter

Tabla 21. Comparación entre versiones de Flutter.

Versión	Fecha Aprox.	Enfoque principal	Comentarios clave
3.27	Diciembre-2025	Estabilidad y base del motor	Mejoras en fidelidad gráfica, animaciones y componentes; Impeller gana protagonismo
3.29	Febrero-2025	Pulido visual e Impeller más estable	Modelo de hilos unificado en móvil, optimizaciones de motor y ajustes en DevTools.
3.32	Marzo-2025	Mejora global y experiencia web	Nuevos widgets, refinamientos en Material/Cupertino y mejor hot reload en web.
3.35	Agosto-2025	Productividad en web y Wasm inicial	Hot reload con estado en web por defecto y primeras pruebas serias con WebAssembly.
3.38	Noviembre-2025	Refinamiento del framework y tooling	Mejor integración de Material 3, gestos mejorados, Dart más cómodo y DevTools más pulidos.

La Tabla 21 permite contemplar la evolución de 5 seis versiones recientes de Flutter se observa un patrón claro: el framework ha pasado de una etapa centrada en la corrección de errores y la estabilización del motor a otra en la que gana peso la productividad del desarrollador, el soporte completo para web y escritorio y el refinamiento de la experiencia de usuario.

Con Flutter 3.27, el énfasis se desplazó hacia el pulido visual y la fidelidad gráfica. El nuevo motor de renderizado asumió un rol más protagonista y se introdujeron mejoras en animaciones, transiciones y componentes que emulan con mayor precisión el aspecto y el comportamiento nativo. Paralelamente, se revisaron detalles que impactan en la experiencia de desarrollo, como el manejo de enlaces profundos, el foco, el texto y las herramientas de depuración [19].

La versión 3.29 marcó una etapa de cambio interno relevante. Se reorganizó la forma en que se gestionan los hilos de ejecución en plataformas móviles, facilitando la interacción con APIs nativas y reduciendo intermediarios innecesarios. Junto con estos ajustes estructurales se incorporaron optimizaciones adicionales en el motor, mejoras en los conjuntos de widgets estándar y cambios en las herramientas que apuntan a un flujo de trabajo más fluido y a un mejor rendimiento en escenarios exigentes[20].

En Flutter 3.32 el foco se centró en una mejora transversal del framework y, en particular, de la experiencia en web. Se añadieron nuevos widgets y patrones de interfaz, se afinó el soporte para las guías de diseño más recientes y se profundizó en aspectos como accesibilidad, entrada de texto y compatibilidad entre plataformas. Al mismo tiempo, se trabajó para que el ciclo de edición y recarga en aplicaciones web fuera más predecible y rápido.

La versión 3.35 se orientó claramente a incrementar la productividad en el desarrollo web y a preparar el terreno para el uso de tecnologías como WebAssembly. El recargado con mantenimiento de estado en el navegador dejó de ser experimental y pasó a formar parte del comportamiento estándar, acercando la experiencia de desarrollo en web a la que ya se tenía en móvil. Además, se introdujeron pruebas y verificaciones tempranas de compilación para facilitar futuras migraciones a nuevos objetivos de compilación y se continuó ajustando la estabilidad y la accesibilidad del framework [21].

Por último, Flutter 3.38 puede entenderse como un ciclo de refinamiento más que de ruptura. Se incorporaron mejoras en el lenguaje que permiten escribir código más conciso, se avanzó en la adopción de los estilos de diseño más actuales, se revisaron animaciones, gestos y comportamientos de desplazamiento, y se unificaron ciertos aspectos del flujo de trabajo para web. El objetivo general de esta versión es que el desarrollador pueda escribir menos código repetitivo, ver los cambios con mayor rapidez y disponer de un entorno de trabajo más estable y coherente en móvil, web y escritorio [22].

d. Casos de éxito usando Flutter

Tabla 22. Casos de éxito con Flutter.

Aplicación	Sector / Uso principal	Alcance / escala aproximada	Rol de Flutter
Google Ads	Publicidad digital	App oficial para gestionar campañas de Google Ads	App móvil construida con Flutter, UI unificada y estadísticas en tiempo real.
Alibaba Xianyu	Comercio electrónico	Plataforma con decenas de millones de usuarios	Flutter permite una interfaz dinámica y tiempos de carga rápidos en gran escala.
My BMW App	Automoción / control remoto del vehículo	App desplegada en más de 30 países	Flutter unifica las versiones iOS y Android con la misma base de código.
Nubank	Banca digital / fintech	Uno de los bancos digitales más grandes de Latinoamérica	Usa Flutter para mantener un ritmo alto de lanzamientos con buena performance.
eBay Motors	e-commerce (compra/venta de vehículos)	App oficial de eBay para el mercado de autos	Flutter duplicó la velocidad de desarrollo y mejoró la experiencia del equipo.
Philips Hue	Domótica / iluminación inteligente	App global para controlar sistemas de iluminación	Flutter aporta una UI moderna y consistente para controlar dispositivos IoT.

En la Tabla 22 se concluye que Flutter no solo se utiliza en proyectos pequeños o académicos, sino que ha sido adoptado por empresas de gran alcance en sectores muy diversos, como la publicidad digital, el comercio electrónico, la banca, la automoción y la domótica. Aplicaciones como Google Ads, plataformas de compraventa a gran escala, soluciones de banca digital, apps oficiales de fabricantes de vehículos y sistemas de control de dispositivos inteligentes muestran que el framework se emplea en entornos donde la continuidad del servicio, la seguridad de la información y la calidad de la experiencia de usuario son factores críticos. En estos casos, Flutter ha permitido unificar la base de código para distintas plataformas, mantener interfaces modernas y consistentes y ofrecer funcionalidades avanzadas como estadísticas en tiempo real o control remoto de dispositivos [23],[24].

Estos ejemplos evidencian que Flutter es capaz de escalar a escenarios con millones de usuarios, soportando aplicaciones con altos volúmenes de transacciones, actualizaciones frecuentes y uso intensivo de notificaciones. El hecho de que se utilice en soluciones de finanzas, comercio electrónico y automoción indica que resulta adecuado para construir sistemas que gestionan pagos, muestran reportes, emiten alertas y funcionan de manera integrada en dispositivos móviles y aplicaciones web.

En consecuencia, se confirma que el framework es una opción viable para el desarrollo de aplicaciones de gestión y control con requerimientos de disponibilidad, trazabilidad de la información y acceso multiplataforma similares a los planteados en el presente proyecto[25],[26].

3.3.5 Back-end Node.js y Express

La capa de lógica de negocio concentra las reglas que gobiernan el funcionamiento del sistema y actúa como intermediaria entre la interfaz de usuario y la base de datos. En esta capa se implementa una API REST construida sobre Node.js y el framework Express, que expone servicios para gestionar usuarios, procesos de negocio, consultas de datos y operaciones de actualización. De este modo, los clientes no acceden directamente al gestor de base de datos, sino que interactúan con un conjunto de endpoints bien definidos y controlados.

a. Node.js

Node.js es un entorno de ejecución de JavaScript del lado del servidor, diseñado en torno a un modelo de entrada/salida asíncrono y orientado a eventos. En lugar de bloquearse mientras espera la respuesta de la base de datos o del sistema de archivos, delega esas operaciones a mecanismos del sistema y continúa atendiendo otras solicitudes. Este enfoque permite manejar muchas conexiones concurrentes con un consumo de recursos relativamente moderado, lo que resulta apropiado para aplicaciones web que deben atender múltiples usuarios de forma simultánea.

Internamente, Node.js se basa en un bucle de eventos que coordina la ejecución de tareas. El código JavaScript se ejecuta en un único hilo, pero las operaciones potencialmente lentas (consultas a la base de datos, acceso a disco, peticiones a otros servicios) se gestionan de manera asíncrona, de forma que no bloquean la atención del resto de solicitudes. Con el paso del tiempo, el ecosistema ha evolucionado desde el uso intensivo de funciones de retorno de llamada (callbacks) hacia un modelo más legible basado en promesas y en la sintaxis `async/await`, lo que facilita la construcción de código más claro y mantenible.

Otra característica relevante de Node.js es la existencia de un gestor de paquetes integrado, que permite incorporar fácilmente librerías externas para autenticación, validación, seguridad, registro de eventos, pruebas automatizadas y muchas otras necesidades habituales en una API moderna. Esto favorece el desarrollo modular y la reutilización de soluciones ya probadas.

b. Express como framework para la API REST

Sobre Node.js se utiliza Express, un framework minimalista que proporciona una estructura clara para construir servicios web. Express facilita la definición de rutas, la gestión de parámetros, el envío de respuestas en distintos formatos y el encadenamiento de funciones intermedias que se ejecutan antes o después de los controladores.

En una API REST típica construida con Express es posible distinguir varios componentes:

- Rutas: indican qué URL y qué tipo de petición (GET, POST, PUT, DELETE, etc.) se asocian con cada operación del sistema.
- Controladores: contienen la lógica que se ejecuta cuando llega una solicitud a una ruta concreta; validan datos, aplican reglas de negocio y coordinan la comunicación con los servicios internos.
- Servicios: agrupan funciones reutilizables que encapsulan operaciones complejas, por ejemplo, cálculos de totales, actualizaciones encadenadas o aplicación de políticas específicas.
- Middleware: se encargan de tareas transversales como autenticación, autorización, registro de peticiones, manejo de errores y validación del cuerpo de las solicitudes.
- Adaptadores de acceso a datos: elementos que se conectan con la capa de base de datos, ejecutan consultas y devuelven resultados a los servicios.

Esta separación de responsabilidades ayuda a mantener el código organizado, facilita la incorporación de nuevas funcionalidades y permite probar de manera aislada las distintas partes de la lógica de negocio.

c. Ventajas y desventajas de Node.js y Express en la lógica de negocio

Tabla 23. Ventajas y desventajas de Node.js

Aspecto	Ventajas	Desventajas / Puntos de atención
Lenguaje y ecosistema	Uso de JavaScript en servidor y cliente; gran cantidad de librerías disponibles	Ecosistema muy dinámico; es necesario seleccionar bien las dependencias y mantenerlas al día
Modelo de ejecución	Entrada/salida asíncrona y orientada a eventos; buena capacidad para manejar muchas conexiones	No es ideal para tareas intensivas en CPU si no se diseñan mecanismos adicionales
Estructura de la API	Express proporciona rutas, middleware y una estructura sencilla de entender y extender	Un diseño poco disciplinado puede derivar en proyectos desordenados y difíciles de mantener
Productividad	Herramientas modernas, recarga en caliente, integración con sistemas de construcción y pruebas	La curva de aprendizaje de modelos asíncronos puede ser un reto para quienes vienen de entornos puramente síncronos
Mantenibilidad y modularidad	Fácil separación de rutas, controladores y servicios; favorece la creación de módulos reutilizables	Sin una arquitectura definida (capas, dominios, etc.), el código puede crecer de forma caótica
Seguridad	Existe una amplia oferta de paquetes para autenticación, cifrado, protección contra ataques comunes	Muchas dependencias implican revisar con cuidado vulnerabilidades y mantener actualizaciones

En la Tabla 23 se entiende que la combinación Node.js–Express ofrece un equilibrio interesante entre flexibilidad y simplicidad. Permite levantar rápidamente una API REST funcional y, al mismo tiempo, ofrece herramientas suficientes para estructurar proyectos de tamaño medio o grande siguiendo buenas prácticas de modularización y separación por capas.

d. Evolución y consideraciones de versión

A lo largo de su evolución, Node.js ha incorporado mejoras significativas en rendimiento, gestión de memoria, compatibilidad con nuevas características del lenguaje y soporte para estándares modernos. Las versiones más recientes de tipo LTS (Long Term Support) suelen ser la opción recomendada para proyectos que buscan estabilidad a medio y largo plazo, ya que reciben actualizaciones de seguridad y correcciones críticas durante un periodo prolongado.

En cuanto a Express, el framework ha mantenido una filosofía minimalista y estable, evitando cambios bruscos en su forma de uso. Esto ha permitido que proyectos contruidos sobre versiones anteriores puedan actualizarse sin grandes reescrituras. Sobre esta base, han ido apareciendo patrones y bibliotecas complementarias que ayudan a resolver preocupaciones recurrentes, como la organización de carpetas, la validación de datos de entrada o la definición de esquemas de respuesta coherentes.

Desde la perspectiva de diseño de la lógica de negocio, lo más importante no es tanto la versión concreta utilizada como la arquitectura elegida encima de Node.js y Express: definir capas claras, establecer reglas de acceso a datos, centralizar el manejo de errores y documentar adecuadamente los endpoints. Cuando estos aspectos se abordan de forma sistemática, la API resultante se vuelve más predecible, más fácil de probar y más sencilla de mantener a medida que evoluciona el sistema.

3.3.6 Base de datos relacional en PostgreSQL

La capa de datos del sistema se apoya en PostgreSQL, un sistema gestor de bases de datos relacional de código abierto orientado a ofrecer alta fiabilidad, integridad de la información y soporte para consultas complejas. A diferencia de soluciones más ligeras, PostgreSQL se ha consolidado como una alternativa robusta tanto para proyectos de tamaño medio como para sistemas empresariales, gracias a su enfoque en el cumplimiento de las propiedades ACID (Atomicity, Consistency, Isolation and Durability), su mecanismo de control de concurrencia y su amplia capacidad de extensión.

Desde el punto de vista histórico, PostgreSQL tiene su origen en proyectos académicos centrados en sistemas de gestión de datos avanzados, lo que se refleja en su diseño modular y en la atención especial que presta a la corrección y a la consistencia. Con el paso del tiempo, el sistema ha ido incorporando funcionalidades que lo sitúan al nivel de otros gestores comerciales, manteniendo a la vez la filosofía de software libre y una comunidad activa que contribuye de forma constante a su desarrollo.

a. Modelo relacional y diseño lógico

En la capa de datos, PostgreSQL se basa en el modelo relacional, en el que la información se organiza en tablas relacionadas mediante claves primarias y foráneas. Cada tabla representa una entidad del dominio del sistema, mientras que las relaciones entre ellas permiten reconstruir el contexto completo de los datos.

El diseño lógico se apoya en principios de normalización, lo que contribuye a reducir redundancias, evitar inconsistencias y facilitar la actualización de información. Las claves primarias garantizan la unicidad de cada registro dentro de una tabla, mientras que las claves foráneas aseguran la integridad referencial entre tablas relacionadas. Además, es posible definir restricciones adicionales (constraints) para validar formatos, rangos de valores, unicidad de campos y condiciones específicas del negocio.

PostgreSQL ofrece, además, soporte para tipos de datos avanzados, arrays, campos JSON y extensiones especializadas, lo que permite adaptar el modelo a necesidades particulares sin abandonar la estructura relacional básica. Esta flexibilidad puede ser útil cuando se requiere almacenar configuraciones, metadatos o información semiestructurada asociada a los registros principales.

b. Mecanismos clave: ACID, concurrencia e índices

Uno de los pilares de PostgreSQL es el cumplimiento de las propiedades ACID, esenciales en sistemas donde se registran operaciones sensibles y se debe evitar que errores o fallos intermedios dejen la base de datos en estados incoherentes. Cada operación de inserción, actualización o eliminación se gestiona dentro de transacciones que garantizan que los cambios se apliquen por completo o se deshagan por completo ante cualquier problema.

Para gestionar el acceso concurrente de múltiples usuarios, PostgreSQL utiliza un mecanismo denominado MVCC (Multi-Version Concurrency Control). En lugar de bloquear de forma agresiva los datos, el sistema mantiene versiones temporales de los registros mientras se realizan operaciones, de modo que cada transacción vea un estado consistente de la base de datos sin impedir que otras transacciones lean o escriban en

paralelo. Este enfoque mejora el rendimiento en escenarios donde varias personas o procesos consultan y actualizan la información de forma simultánea.

Otro componente fundamental son los índices, estructuras auxiliares que aceleran la búsqueda de registros en función de uno o varios campos. PostgreSQL dispone de varios tipos de índices, que pueden seleccionarse según el tipo de consultas que se ejecuten con más frecuencia. Un diseño adecuado de índices permite reducir notablemente los tiempos de respuesta en reportes, listados filtrados y consultas de detalle.

Además, PostgreSQL admite funciones almacenadas, vistas (incluidas vistas materializadas) y disparadores (triggers), que permiten trasladar parte de la lógica de validación y cálculo directamente a la base de datos. Estos mecanismos pueden emplearse para verificar reglas de negocio, mantener registros de auditoría o automatizar la actualización de ciertos campos derivados cuando se producen cambios en las tablas principales.

c. Ventajas y desventajas de PostgreSQL como gestor de datos

Tabla 24. Ventajas y desventajas de PostgreSQL.

Aspecto		Ventajas	Desventajas / Puntos de atención
Integridad y consistencia	y	Cumplimiento estricto de propiedades ACID y mecanismos sólidos de integridad referencial	Requiere un diseño cuidadoso del esquema; un mal modelado puede repercutir en rendimiento
Concurrencia y rendimiento	y	MVCC permite múltiples lecturas y escrituras concurrentes con buen rendimiento	Consultas muy complejas o sin índices adecuados pueden degradar el desempeño
Flexibilidad del modelo	del	Soporte para tipos de datos avanzados, JSON, funciones almacenadas, vistas y extensiones	El uso excesivo de lógica compleja en la base de datos puede dificultar la portabilidad
Escalabilidad y robustez	y	Apto para proyectos que crecen en volumen de datos y usuarios, con opciones de replicación y particionado	La configuración de alta disponibilidad y replicación puede requerir conocimientos especializados
Herramientas y ecosistema	y	Dispone de herramientas de copia de seguridad, monitoreo y administración ampliamente utilizadas	Es necesario planificar políticas de respaldo y mantenimiento para evitar problemas a largo plazo
Licenciamiento y costos	y	Software libre, sin costo de licencia, con amplia comunidad de soporte	En entornos críticos puede ser recomendable contar con soporte profesional adicional

En términos prácticos, PostgreSQL ofrece una base sólida para sistemas que requieren registros fiables, reportes consistentes y trazabilidad de las operaciones. La

combinación de integridad referencial, control de concurrencia y opciones avanzadas de modelado de datos permite abordar escenarios complejos sin sacrificar estabilidad.

Al mismo tiempo, su potencia implica la necesidad de tomar decisiones de diseño informadas: definir adecuadamente las relaciones entre tablas, elegir índices eficientes para las consultas más utilizadas y establecer políticas de mantenimiento que mantengan la base de datos en buen estado a medida que crece el volumen de información.

d. Evolución, versiones y buenas prácticas de uso

A lo largo de los años, PostgreSQL ha incorporado múltiples mejoras en rendimiento, administración de recursos y soporte para nuevas funcionalidades. Las versiones recientes han reforzado el manejo de grandes volúmenes de datos, las capacidades analíticas y el soporte para extensiones orientadas a distintos dominios.

Para proyectos nuevos, es habitual optar por una versión estable y soportada durante varios años, lo que permite beneficiarse de las mejoras acumuladas en rendimiento y seguridad, sin asumir riesgos asociados a versiones obsoletas.

En cuanto a buenas prácticas, destacan las siguientes:

- Definir un esquema relacional claro, documentando las entidades y relaciones principales.
- Diseñar índices en función de las consultas más frecuentes, evitando tanto la ausencia total de índices como la creación indiscriminada de índices innecesarios.
- Establecer políticas de respaldo y restauración periódicas, incluyendo pruebas de recuperación para garantizar que los procedimientos funcionan en caso de fallo.
- Monitorizar el rendimiento mediante herramientas de análisis de consultas y estadísticas internas, ajustando configuraciones o consultas cuando sea necesario.

- Separar, cuando resulte apropiado, la lógica de negocio que debe residir en la base de datos de aquella que es más conveniente tratar en la capa de aplicación.

En conjunto, PostgreSQL se presenta como una opción robusta y flexible para la capa de datos, capaz de sostener tanto las operaciones diarias del sistema como las necesidades de análisis y crecimiento a medio y largo plazo. Su combinación de estabilidad, potencia y ausencia de costos de licencia lo convierte en un componente adecuado para proyectos que buscan un equilibrio entre calidad técnica y viabilidad operativa.

3.3.7 Seguridad del Sistema

En el desarrollo de la plataforma se incorporaron medidas de seguridad transversales que permiten proteger la información de estudiantes, representantes y del personal encargado de la administración. Estas medidas se aplican tanto en la capa de presentación (Flutter) como en el backend y en la base de datos, con el objetivo de asegurar un acceso controlado y reducir riesgos de manipulación, exposición o pérdida de datos.

a. Seguridad en el backend

En la capa de lógica de negocio se implementó un sistema de autenticación basado en tokens JWT (JSON Web Tokens), lo que permite validar cada solicitud HTTP y mantener sesiones con tiempo definido de vigencia. Cuando un usuario inicia sesión correctamente, el servidor genera un token firmado que contiene información básica del usuario y su rol. Este token se envía al cliente y debe incluirse en las cabeceras de todas las peticiones subsiguientes a endpoints protegidos.

Las contraseñas se almacenan mediante hashing con bcrypt, un algoritmo de cifrado unidireccional que incorpora un factor de trabajo ajustable (salt rounds). Esto evita que, en caso de exposición de la base de datos, las contraseñas puedan ser recuperadas en texto plano. Durante el proceso de autenticación, la contraseña ingresada se compara con el hash almacenado sin necesidad de descifrarla.

Se estableció un control de permisos por tipo de usuario, de manera que cada perfil (administrador, profesor, representante) accede únicamente a las funciones necesarias para su trabajo. Este control se implementa mediante middleware de autorización que verifica el rol del usuario antes de permitir el acceso a ciertas rutas de la API.

El sistema también registra en una bitácora las acciones importantes, tales como creación, modificación y eliminación de registros críticos (estudiantes, pagos, mensualidades). Cada entrada de auditoría incluye el identificador del usuario que realizó la acción, la tabla afectada, el tipo de operación y la marca de tiempo. Esto permite revisar actividades, detectar fallas o comportamientos irregulares y mantener un historial útil para auditoría y cumplimiento.

Adicionalmente, se configuró CORS (Cross-Origin Resource Sharing) de manera restrictiva, permitiendo únicamente solicitudes provenientes de los dominios autorizados de la aplicación web y móvil. Esto reduce el riesgo de ataques desde orígenes no confiables.

b. Seguridad en el frontend

En la aplicación desarrollada con Flutter se añadieron controles adicionales para reforzar la seguridad en el lado del cliente:

- **Límite de intentos fallidos al iniciar sesión:** Si un usuario supera el número permitido de intentos incorrectos, el acceso queda temporalmente bloqueado durante un periodo definido. Esto evita ataques de fuerza bruta sobre las credenciales.
- **Temporizadores en funciones sensibles:** En operaciones como recuperación de contraseña y envío de códigos de verificación por correo electrónico, se implementaron temporizadores que impiden solicitar correos de manera continua. El usuario debe esperar un tiempo mínimo (60 segundos) antes de poder solicitar un nuevo envío, lo que previene el abuso del servicio y reduce la carga en el servidor de correo.
- **Inicio de sesión con Google OAuth 2.0:** Se integró la autenticación mediante Google como alternativa al método tradicional con correo y contraseña. Esta

opción utiliza los mecanismos de validación del proveedor externo, lo que ofrece mayor confiabilidad y reduce la carga de gestión de credenciales por parte del sistema. El flujo OAuth garantiza que el sistema nunca reciba directamente la contraseña del usuario de Google.

- **Validación continua del token:** La aplicación verifica en todo momento la validez del token JWT almacenado localmente. Si el token expira o es inválido, se cierra la sesión automáticamente y se redirige al usuario a la pantalla de inicio de sesión, evitando accesos indebidos con tokens comprometidos o caducados.
- **Validaciones de entrada en formularios:** Todos los formularios de la aplicación incorporan validaciones del lado del cliente (longitud, formato, caracteres permitidos) antes de enviar los datos al servidor. Esto reduce errores de entrada y previene envíos de información malformada.

c. Seguridad en la base de datos

A nivel de PostgreSQL se aplicaron las siguientes medidas:

- **Restricciones de integridad referencial:** Todas las relaciones entre tablas están protegidas mediante claves foráneas con acciones ON DELETE y ON UPDATE apropiadas, evitando estados inconsistentes.
- **Permisos de usuario de base de datos:** Se configuró un usuario específico para la aplicación con permisos restringidos únicamente a las operaciones necesarias (SELECT, INSERT, UPDATE, DELETE en tablas específicas), sin acceso a funciones administrativas del servidor.
- **Conexión cifrada:** La comunicación entre el backend y la base de datos se realiza mediante SSL/TLS, protegiendo los datos en tránsito.
- **Procedimientos almacenados y triggers:** Para operaciones críticas (como la actualización de estados de mensualidades tras un pago), se utilizan funciones almacenadas que centralizan la lógica y reducen la superficie de ataque.

3.3.8 Diagrama metodológico del proyecto

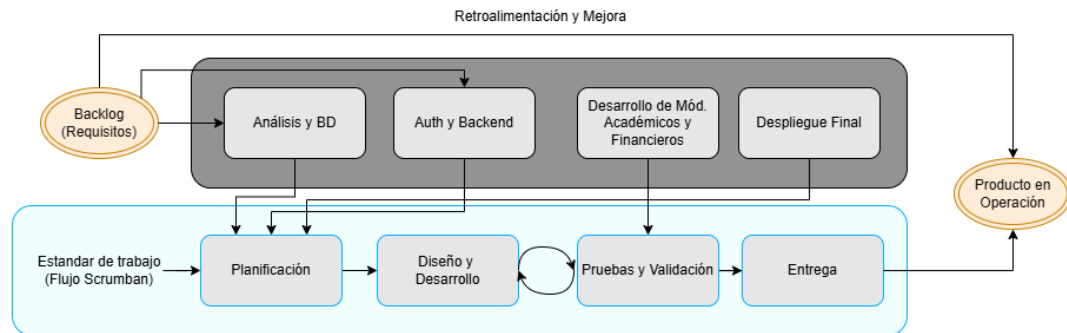


Figura 13. Diagrama metodológico del proyecto.

En la Figura 13 se presenta el esquema de ejecución del proyecto bajo la metodología híbrida Scrumban, empleada para organizar y controlar el desarrollo del sistema multiplataforma de la Academia Oficial de Fútbol Porto Ambato. El trabajo inicia en el “Backlog (Requisitos)”, donde se consolidan los requerimientos funcionales y no funcionales levantados durante la investigación. A partir de este backlog, las actividades se priorizan y se ejecutan de forma incremental mediante bloques de trabajo que estructuran el avance del proyecto, manteniendo la flexibilidad de ajuste conforme se valida el sistema.

De acuerdo con lo representado, el proyecto se organizó en cuatro iteraciones principales: (1) Análisis y BD, orientada al modelado del dominio y la definición de la base de datos; (2) Auth y Backend, enfocada en la autenticación y la base operativa de la API; (3) Desarrollo de Mód. Académicos y Financieros, destinada a la implementación de las funcionalidades de gestión académica y de cobros; y (4) Despliegue Final, dirigida a la estabilización, ajustes finales y puesta en marcha. Esta secuencia evidencia un desarrollo por bloques funcionales con entregables verificables, reduciendo el riesgo de concentrar validaciones únicamente al cierre del proyecto.

La figura integra además el “Estandar de trabajo (Flujo Scrumban)” como un flujo continuo compuesto por las fases de Planificación, Diseño y Desarrollo, Pruebas y Validación y Entrega. Este flujo proporciona visibilidad permanente del estado de las tareas y facilita la identificación de cuellos de botella durante la ejecución. Finalmente,

se destaca la “Retroalimentación y Mejora” como un eje transversal: los resultados obtenidos en pruebas y en la entrega pueden retornar al backlog para ajustes o refinamientos, manteniendo un ciclo iterativo de mejora continua hasta alcanzar el estado final del sistema como “Producto en Operación”.

3.4 Metodología de desarrollo

La construcción de un sistema de información no solo implica definir la arquitectura técnica y las tecnologías a utilizar, sino también adoptar una metodología de desarrollo que oriente la forma de trabajar. La elección de una metodología adecuada resulta fundamental para organizar las tareas, gestionar los requisitos, controlar los tiempos, manejar los cambios y garantizar que el producto final responda a las necesidades planteadas.

Una metodología bien seleccionada aporta un marco de referencia claro, reduce la improvisación y proporciona herramientas para planificar, ejecutar y evaluar el avance del proyecto de manera sistemática.

En este contexto, se consideró necesario analizar distintos enfoques de desarrollo, tanto tradicionales como ágiles, para seleccionar aquel que mejor se ajustara a las características del proyecto, al tamaño del equipo y a las restricciones de tiempo propias de un proyecto.

3.4.1 Comparación de metodologías de desarrollo

Para sustentar la elección de la metodología se realizó una comparación entre cuatro enfoques representativos: el modelo en cascada como metodología tradicional, Scrum y Kanban como marcos ágiles ampliamente difundidos, y Scrumban como propuesta híbrida que integra elementos de ambos.

Tabla 25. Comparación entre metodologías de desarrollo de software.

Aspecto	Modelo en cascada	Scrum	Kanban	Scrumban (enfoque híbrido)
Estructura del proceso	Secuencial, fases rígidas y bien delimitadas	Iteraciones fijas (<i>sprints</i>) con objetivos definidos	Flujo continuo sin iteraciones obligatorias	Iteraciones ligeras combinadas con tablero de flujo visual
Gestión de requisitos	Se definen casi por completo al inicio	Se priorizan y refinan en cada sprint	Se reordenan según flujo y capacidad	Se priorizan en un backlog y se ajustan según el flujo del tablero
Entregas parciales	Normalmente al final del proyecto	Entregas incrementales al final de cada sprint	Entregas cuando las tarjetas llegan a “Terminado”	Entregas incrementales dentro de sprints, visibles en el tablero
Roles formales	Rol fuerte del analista y del jefe de proyecto	Definidos (Product Owner, Scrum Master, equipo)	No exige roles formales específicos	Puede adoptar roles de Scrum con aplicación flexible
Visualización del trabajo	Principalmente mediante documentación y cronogramas	El tablero no es el foco central pero puede utilizarse	Tablero Kanban con columnas y tarjetas	Tablero Kanban + planificación por sprints
Gestión del cambio	Costosa, cambios tardíos impactan en fases previas	Admitida entre sprints, gestionada vía backlog	Muy flexible; cambios según prioridad y capacidad	Flexible, controlando cambios entre iteraciones y en el tablero
Control del trabajo en curso (WIP)	No es un concepto central	Se controla a través del alcance del sprint	Es uno de los principios básicos	Se aplican límites de WIP sobre el tablero dentro de cada sprint
Adecuación a proyectos con incertidumbre	Limitada	Alta	Alta	Alta; combina planificación y adaptación continua

De la Tabla 25 se define que el modelo en cascada proporciona una estructura muy definida, pero su rigidez lo hace poco adecuado cuando los requisitos evolucionan durante el desarrollo o cuando se desea validar funcionalidad de forma temprana. Los cambios importantes introducidos en etapas avanzadas suelen tener un impacto elevado en tiempo y coste.

En contraste, Scrum aporta una organización basada en iteraciones de duración fija, con roles y eventos que favorecen la entrega incremental y la revisión periódica del producto. Esto facilita la adaptación a cambios de requisitos, aunque requiere una disciplina clara en la planificación y cierre de cada sprint.

Por su parte, Kanban pone el énfasis en el flujo continuo de trabajo y en la visualización del estado de las tareas mediante un tablero. Su principal ventaja es la flexibilidad para introducir y priorizar tareas, así como la capacidad de detectar rápidamente cuellos de botella. Sin embargo, al no definir iteraciones ni roles formales, puede resultar demasiado abierto si no se complementa con prácticas adicionales de planificación.

En este escenario, Scrumban se posiciona como un enfoque intermedio que combina la planificación por iteraciones de Scrum con la gestión visual del flujo y los límites de trabajo en curso propios de Kanban. Esto permite contar con objetivos claros por periodo de trabajo, sin renunciar a la flexibilidad y a la visibilidad del proceso.

3.4.2 Metodología híbrida Scrumban

A partir del análisis anterior, se optó por adoptar Scrumban como metodología de desarrollo para el proyecto. La decisión se apoya en los siguientes argumentos:

Es necesario planificar el trabajo en bloques de tiempo definidos, a fin de cumplir con plazos de entrega parciales y con el cronograma general. La noción de sprints tomada de Scrum resulta útil para establecer objetivos concretos en cada periodo.

Al mismo tiempo, resulta importante mantener una visibilidad permanente del estado de las tareas, para poder identificar rápidamente qué actividades están pendientes, cuáles se encuentran en progreso y cuáles han sido completadas. El tablero Kanban y los límites de trabajo en curso proporcionan este control visual y ayudan a evitar la dispersión.

La naturaleza iterativa del proyecto, con posibles ajustes de requisitos y prioridades a medida que se avanza, exige un enfoque que permita adaptarse al cambio sin perder el orden, algo que Scrumban facilita al combinar backlog, sprints y flujo continuo.

El tamaño reducido del equipo y la variedad de actividades (análisis, diseño, implementación, pruebas y documentación) se benefician de un marco ligero, que no imponga una estructura excesivamente formal, pero que ofrezca directrices claras para organizar el trabajo.

En síntesis, Scrumban se selecciona por ofrecer un equilibrio adecuado entre planificación y flexibilidad, entre estructura y adaptación, resultando apropiado para proyectos con restricciones de tiempo y alcance variable durante su desarrollo.

3.4.3 Beneficios obtenidos con el enfoque Scrumban

La adopción de Scrumban como metodología de desarrollo permitió obtener varios beneficios:

- Transparencia del estado del proyecto, gracias al tablero visual y a la actualización continua de las tareas.
- Control del trabajo en curso, evitando la dispersión en múltiples actividades simultáneas y favoreciendo la finalización efectiva de las tareas iniciadas.
- Capacidad de respuesta ante cambios, al combinar un backlog priorizado con iteraciones cortas y revisión constante de las prioridades.
- Entrega progresiva de funcionalidades, lo que facilitó validar el funcionamiento del sistema por partes y realizar ajustes tempranos.
- Mejora continua del proceso, a través de las retrospectivas, que permitieron introducir pequeños cambios en la organización del trabajo para aumentar la eficiencia y la calidad.

En conjunto, la metodología Scrumban proporcionó un marco flexible y estructurado que acompañó de forma adecuada el desarrollo del sistema, permitiendo avanzar de manera ordenada, responder a los cambios y mantener una visión clara del estado del proyecto durante todo el proceso.

3.5 Desarrollo de la propuesta

La propuesta tecnológica planteada en esta investigación consiste en un sistema multiplataforma para el seguimiento y control de cobros en la Academia Oficial de Fútbol Porto Ambato. El sistema integra:

- Una aplicación desarrollada con Flutter.
- Un servicio backend implementado en Node.js con Express exponiendo una API REST.
- Una base de datos relacional en PostgreSQL, diseñada especialmente para el contexto de academias de fútbol, con soporte para roles, estudiantes, matrículas, mensualidades, pagos, auditoría y módulos complementarios.

El desarrollo de la propuesta se organizó y ejecutó aplicando la metodología híbrida Scrumban, que combina la planificación por iteraciones de Scrum con el control visual y los límites de trabajo en progreso propios de Kanban. Esta combinación resultó adecuada para un contexto académico, con tiempo limitado, equipo reducido y un alcance funcional amplio.

3.5.1 Definición de requisitos

a. Análisis de requisitos funcionales

En primer lugar, se realizó un levantamiento detallado de los requisitos funcionales del sistema, utilizando como insumos:

- Encuesta con el administrador de la Academia Porto Ambato.
- Observación directa del proceso actual de cobros.
- Revisión de comprobantes, cronogramas de pagos y tablas de categorías existentes.

Con base en esta información se identificaron las funcionalidades esenciales, entre ellas:

- Registro y gestión de usuarios (administrador, profesores, representantes).
- Registro y actualización de estudiantes y sus datos personales.
- Gestión de categorías y subcategorías (por edad, grupo).

- Registro de matrículas por estudiante, vinculadas a una categoría y a un periodo.
- Generación de mensualidades para cada matrícula, con valor, fecha de vencimiento y estado.
- Registro de pagos (totales o parciales) y actualización automática del estado de las mensualidades.
- Consulta del estado de cuenta por estudiante (mensualidades pagadas, pendientes o vencidas).
- Emisión de reportes de pagos y morosidad por periodo, categoría y estudiante.

b. Análisis de requisitos no funcionales

Paralelamente se establecieron los requisitos no funcionales, relacionados con atributos de calidad del sistema. Para estructurar y describir estos atributos se tomó como referencia el modelo de calidad de producto de **ISO/IEC 25010:2023**; sin embargo, debido al alcance académico del proyecto, al tiempo disponible y a la priorización de módulos críticos, esta referencia se aplicó de manera parcial, seleccionando únicamente las características más relevantes para el contexto de la academia (seguridad, usabilidad, rendimiento, mantenibilidad y disponibilidad), sin ejecutar un proceso formal de evaluación o certificación.

Los requisitos no funcionales definidos fueron:

- **Seguridad:** protección de datos personales de estudiantes y representantes, autenticación con credenciales y manejo de tokens de acceso.
- **Rendimiento:** tiempos de respuesta aceptables en las operaciones más utilizadas (listar estudiantes, generar mensualidades, registrar pagos).
- **Usabilidad:** interfaz clara y comprensible para usuarios no técnicos, con mensajes de error amigables y flujos sencillos.

- **Escalabilidad:** posibilidad de soportar un incremento de estudiantes y categorías sin rehacer la arquitectura.
- **Disponibilidad:** facilidad para desplegar el sistema en un servidor accesible vía web.

c. Accesibilidad y diseño inclusivo (WCAG 2.2).

Adicionalmente, se incorporaron criterios básicos de accesibilidad para que el sistema sea utilizable por personas con distintas necesidades, tomando como referencia WCAG (Web Content Accessibility Guidelines).

Al igual que con ISO, se aplicaron criterios seleccionados y verificables, sin declarar conformidad total.

- **Accesibilidad auditiva:** el sistema no depende de señales sonoras; confirmaciones, errores y notificaciones se presentan en formato visual y textual.
- **Accesibilidad motriz:** priorización de operabilidad en web mediante teclado, controles claros y reducción de pasos en tareas críticas (consulta y registro de pagos).
- **Accesibilidad cognitiva:** mensajes simples, consistencia en etiquetas, prevención de errores en formularios y retroalimentación inmediata ante acciones sensibles.

Tabla 26. Lista de verificación de accesibilidad aplicada.

Criterio (WCAG 2.2)	Qué se implementa en el sistema	Evidencia	Estado
Operable por teclado	Navegación por campos/botones en web (Tab/Shift+Tab)	Anexo A	Aplicado
Foco visible	El usuario identifica el control activo al navegar	Anexo A	Aplicado
No depender solo del color	Estados “Pagado/Pendiente/Vencido” en texto, no solo color	Figura 47	Aplicado
Identificación de errores	Mensajes claros en validaciones de formularios	Figura 42	Aplicado
Prevención de errores en acciones críticas	Confirmación antes de anular/registrar operaciones sensibles	Anexo A	Aplicado
Ayuda consistente	Etiquetas/ayudas similares en formularios equivalentes	Figura 42	Aplicado

Criterio (WCAG 2.2)	Qué se implementa en el sistema	Evidencia	Estado
Reducción de entrada redundante	Reutilización/auto-carga de datos del estudiante cuando aplica	Figura 41/42	Aplicado
Objetivos de toque adecuados	Botones principales con tamaño suficiente (web/móvil)	Anexo A	Aplicado
Autenticación accesible (sin fricción innecesaria)	Login claro, mensajes comprensibles; no se bloquea por confusión del usuario	Figura 22/Anexo a	Aplicado
Claridad del contenido	Textos de estado y retroalimentación entendibles por usuario no técnico	Figura 41/47	Aplicado

La Tabla 26 presenta la lista de verificación de accesibilidad aplicada al sistema, tomando como referencia criterios puntuales de WCAG 2.2. En ella se resume qué medidas se implementaron en la aplicación y cómo se evidencian, priorizando aspectos clave de uso real en un entorno administrativo y de representantes. Se incluyen criterios de operabilidad como la navegación por teclado (Tab/Shift+Tab) y la presencia de foco visible, asegurando que el usuario identifique en todo momento el control activo durante la interacción; estas evidencias se documentan en el Anexo A. Asimismo, se considera la interacción en diferentes dispositivos mediante objetivos de toque adecuados, garantizando botones principales con tamaño suficiente tanto en web como en móvil.

De igual manera, la tabla integra criterios orientados a la comprensión y prevención de errores, esenciales en módulos sensibles como pagos y mensualidades. Por ejemplo, se evita depender únicamente del color al mostrar estados “Pagado/Pendiente/Vencido” también en texto (Figura 47), y se incorporan mensajes claros de validación para la identificación de errores (Figura 42). Para reducir fallos en operaciones críticas, se implementan confirmaciones antes de registrar o anular acciones relevantes (Anexo A), manteniendo además ayuda consistente en formularios equivalentes (Figura 42) y reduciendo la entrada redundante mediante reutilización o autocompletado de datos (Figuras 41 y 42). En conjunto, estas medidas fortalecen la claridad del contenido y la usabilidad general del sistema, mostrando retroalimentación comprensible para usuarios no técnicos (Figuras 41 y 47).

3.5.2 Backlog del proyecto

El backlog representa el listado priorizado de tareas y funcionalidades necesarias para culminar con el proyecto. En este caso, se definieron de manera general los módulos del sistema y los actores del proyecto, quienes interactuarán con la aplicación de acuerdo con sus responsabilidades.

a. Actores del proyecto

Los actores son los distintos tipos de usuarios o entidades que interactúan con el sistema, cada uno con necesidades y comportamientos específicos. Para el presente proyecto, los actores principales se describen en la Tabla 27.

Tabla 27. Actores principales del proyecto.

Actor	Descripción	Funciones principales
Administrador	Responsable de la gestión administrativa y financiera de la academia	Configurar categorías y subcategorías, registrar estudiantes, matrículas y mensualidades; registrar y validar pagos; generar reportes y consultas financieras
Profesor	Personal técnico encargado del entrenamiento deportivo de los estudiantes	Consultar listados de estudiantes por subcategoría, revisar estado de pagos (solo lectura), apoyar en la actualización de información básica cuando sea necesario
Representante	Padre, madre o tutor responsable del estudiante	Consultar el estado de cuenta del estudiante, revisar historial de pagos y mensualidades, descargar comprobantes o evidencias de pago

b. Módulos

Los módulos representan los requisitos funcionales principales del sistema, es decir, el comportamiento que debe ofrecer la aplicación al interactuar con los actores. La Tabla 28 resume los módulos considerados y sus funcionalidades generales.

Tabla 28. Módulos del sistema.

Módulo	Funcionalidad principal	Usuario
Login	Inicio de sesión seguro mediante credenciales, validación de rol y emisión de token de acceso.	Todos los actores con cuenta (Administrador, Profesor, Representante).
Perfil	Sección de perfil para que cada usuario autorizado consulte su información básica y cambie su contraseña	Todos los usuarios autenticados.

Módulo	Funcionalidad principal	Usuario
Gestión de usuarios y roles	Administración de cuentas de usuario, asignación de roles y control de acceso a módulos.	Administrador.
Profesores	Gestión de información del cuerpo técnico (datos personales, estado, asignación a categorías).	Administrador.
Estudiantes	Registro, actualización y consulta de datos personales de los estudiantes, estado (activo/inactivo).	Administrador, Profesor.
Categorías y subcategorías	Definición y administración de las categorías y subcategorías de fútbol (rango de edad, grupo, horario, etc.).	Administrador, Profesor.
Matrículas	Registro de la inscripción de estudiantes en subcategorías y periodos; definición de valores de matrícula.	Administrador, Profesor.
Mensualidades	Generación automática de mensualidades por estudiante/periodo, con montos, fechas de vencimiento y estado.	Administrador.
Pagos	Registro de pagos totales o parciales; actualización de estados de mensualidad y generación de comprobantes.	Administrador.
Reportes y consultas	Visualización de reportes de ingresos, morosidad, estados de cuenta, historiales de cobro por estudiante o periodo.	Administrador, Representante.

Con el fin de sintetizar la organización del sistema más allá del listado de módulos, se elaboró un diagrama funcional que resume el flujo general de uso desde la autenticación hasta el acceso a los módulos principales. Este esquema no reemplaza los diagramas de proceso (Figuras 11 y 12), sino que complementa la visión del sistema mostrando la navegación y la disponibilidad de opciones según roles y permisos (RBAC).

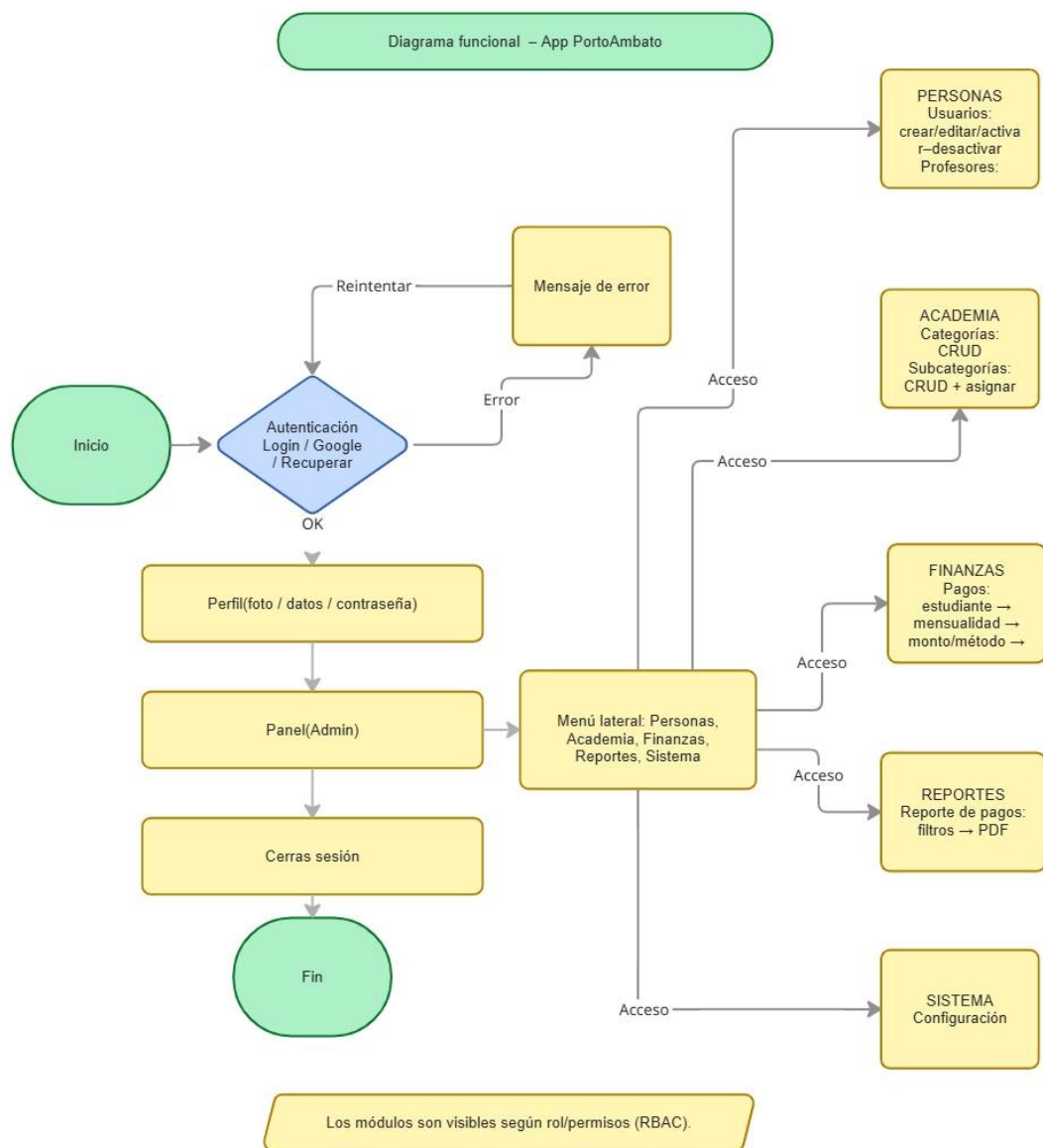


Figura 14. Diagrama funcional del sistema.

En la Figura 14 se observa que el flujo inicia en la autenticación (credenciales, Google o recuperación). Si ocurre un error, el sistema informa mediante un mensaje y permite reintentar. Con una autenticación exitosa, el usuario accede al perfil y al panel, desde donde se habilita el menú lateral con los módulos Personas, Academia, Finanzas, Reportes y Sistema. La visibilidad de cada módulo depende del rol y permisos asignados, lo que refuerza el control de acceso y evita que usuarios no autorizados ejecuten acciones administrativas.

c. Tareas del proyecto

Además de los requisitos funcionales, el proyecto incluye un conjunto de requisitos no funcionales y tareas transversales que garantizan la calidad del sistema:

- ***Mejoras***

- Optimización de la interfaz de usuario en los módulos de pagos y mensualidades, facilitando la búsqueda por nombre, cédula, categoría o estado.
- Mejora de la experiencia de usuario en la consulta de estados de cuenta y reportes financieros, incorporando filtros y paginación.

- ***Correcciones de errores***

- Ajustes en la lógica de generación de mensualidades para evitar duplicidades en un mismo periodo.
- Corrección de inconsistencias en el registro de pagos parciales y cálculo de saldos pendientes.

- ***Tareas técnicas***

- Diseño y normalización de la base de datos en PostgreSQL.
- Implementación del backend en Node.js con Express y TypeScript, aplicando buenas prácticas de modularización.
- Integración de la aplicación cliente desarrollada en Flutter (web/móvil) con la API REST mediante peticiones HTTP seguras.
- Configuración de mecanismos de seguridad (encriptación de contraseñas, tokens de autenticación, control de acceso por roles).
- Implementación de copias de seguridad y configuración del entorno de despliegue.

- **Pruebas**

- Ejecución de pruebas unitarias sobre las funciones críticas del backend.
- Pruebas de integración entre el frontend Flutter y la API REST.
- Pruebas de usabilidad con el administrador y profesores para asegurar que los flujos sean comprensibles.

- **Documentación**

- Elaboración del manual de usuario administrativo (uso de módulos principales de cobros).

3.5.3 Planificación de actividades

Cada módulo del sistema se desglosa en actividades específicas necesarias para cumplir sus requerimientos. Para representar dichas actividades y su evolución a lo largo del proyecto se empleó un tablero Kanban en una herramienta web de gestión de proyectos (Planner).

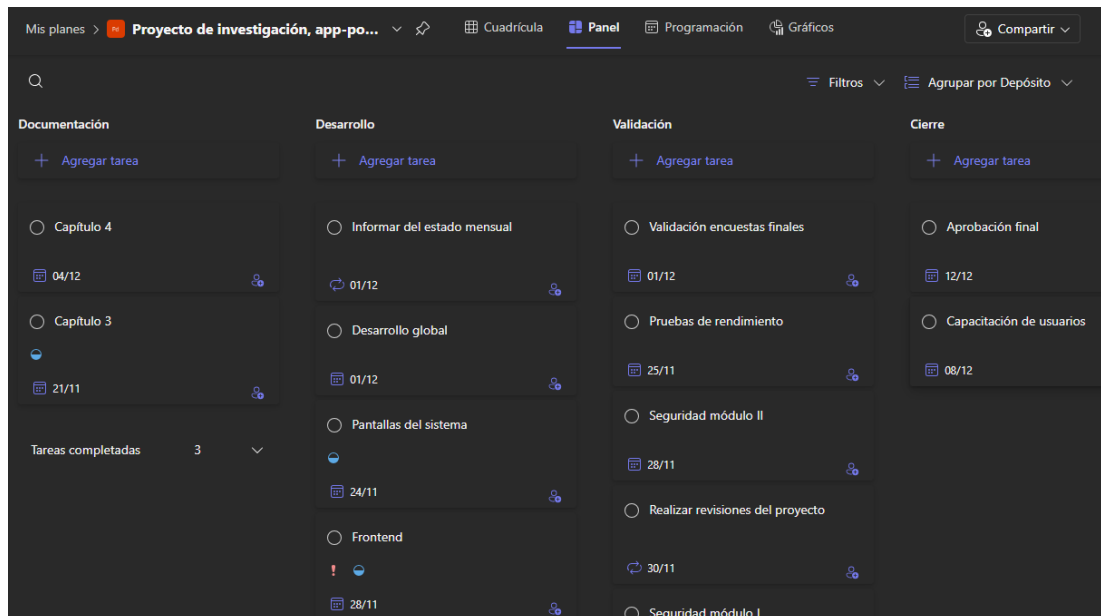


Figura 15. Actividades en Planner.

En el tablero, ilustrado de forma referencial en la Figura 15. se definieron columnas tales como:

- Documentación
- Desarrollo
- Validación
- Cierre

Las tareas se asignaron a estas columnas según su estado, permitiendo visualizar el flujo de trabajo y priorizar lo realmente importante en cada momento.

Para organizar el avance temporal, las actividades se agruparon en sprints de hasta dos semanas de duración. Cada sprint incluyó trabajo tanto de backend como de frontend, lo que permitió avanzar de forma uniforme y paralela en la construcción de la solución:

- Durante las etapas de diseño, se abordaron tareas de modelado del negocio y de la base de datos.
- En las etapas de desarrollo, se priorizó que cada módulo crítico (por ejemplo, pagos) quedara funcional de extremo a extremo: API + interfaz gráfica.
- En las etapas finales, se centró el esfuerzo en pruebas, ajustes y documentación, preparando la implementación en la academia.

Este enfoque facilitó la gestión del proyecto por parte de un solo desarrollador, manteniendo visibilidad sobre lo que se había hecho, lo que estaba en progreso y lo que quedaba pendiente.

3.5.4 Desarrollo iterativo

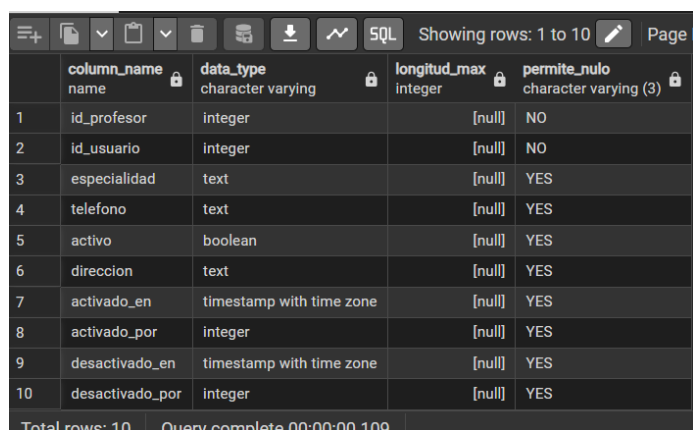
El desarrollo de la solución se ejecutó de manera iterativa a través de varios sprints. A continuación, se describen las principales actividades realizadas en cada uno de ellos.

a. Sprint 1

En el primer sprint se abordaron tareas fundamentales de análisis y diseño del sistema:

- Extracción de características

Se identificaron las entidades principales del dominio de la Academia Porto Ambato: estudiantes, representantes, profesores, categorías, subcategorías, matrículas, mensualidades, pagos y usuarios del sistema. Para cada entidad se definieron sus atributos clave construyendo un diccionario de datos similar al mostrado en la Figura 15, pero adaptado a tablas como mensualidades o pagos.



	column_name	data_type	longitud_max	permite_nulo
	name	character varying	integer	character varying (3)
1	id_profesor	integer	[null]	NO
2	id_usuario	integer	[null]	NO
3	especialidad	text	[null]	YES
4	telefono	text	[null]	YES
5	activo	boolean	[null]	YES
6	direccion	text	[null]	YES
7	activado_en	timestamp with time zone	[null]	YES
8	activado_por	integer	[null]	YES
9	desactivado_en	timestamp with time zone	[null]	YES
10	desactivado_por	integer	[null]	YES

Total rows: 10 Query complete 00:00:00.109

Figura 16. Diccionario de datos tabla “profesores”.

- Diseño de la base de datos

Con las entidades y sus relaciones definidas, se diseñó en PostgreSQL el núcleo de la base de datos encargado de la gestión de usuarios, roles y permisos. En este esquema la tabla usuarios actúa como eje principal, almacenando los datos de cada cuenta y vinculándose mediante la clave foránea id_rol con la tabla roles, donde se agrupan los perfiles funcionales del sistema. La tabla permisos define las operaciones que pueden ejecutarse y, a través de la tabla intermedia rol_permiso, se implementa una relación de muchos a muchos entre roles y permisos, lo que permite construir esquemas de autorización flexibles y fácilmente extensibles. Finalmente, la tabla auditoria registra

las acciones realizadas por cada usuario sobre las distintas tablas de la aplicación, almacenando la acción ejecutada, la tabla afectada, el identificador del registro y la fecha y hora del evento, con lo cual se refuerza la trazabilidad y la integridad de la información. El diagrama relacional resultante, con sus claves primarias y foráneas definidas, se muestra en la Figura 17.

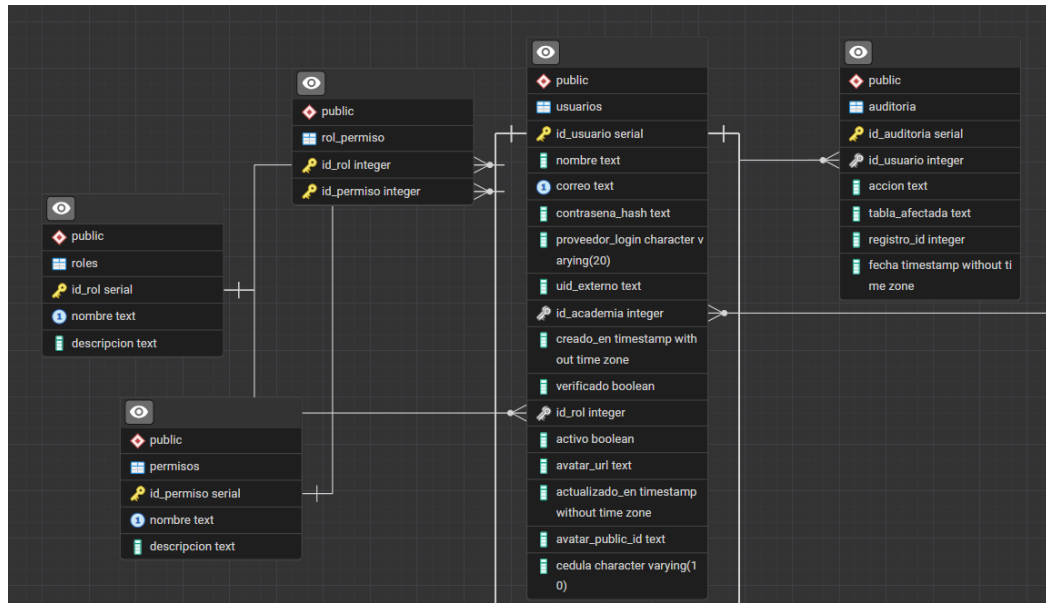


Figura 17. Diagrama de la estructura principal de los usuarios y roles.

Para la gestión académica y financiera de la academia se diseñó un conjunto de tablas que modelan categorías deportivas, estudiantes, profesores, mensualidades, pagos y asistencia. La tabla **categorías** almacena la información general de cada grupo (nombre, rango de edades, academia a la que pertenecen y estado activo), mientras que **subcategorías** detalla los equipos o niveles específicos, incluyendo un código único y la fecha de creación. La asignación de entrenadores se realiza mediante **profesores** y la tabla puente **profesor_categoria**, que vincula a cada profesor con una o varias categorías.

La tabla **estudiantes** concentra los datos personales de los alumnos (nombres, apellidos, fecha de nacimiento, dirección, teléfono, cédula, academia y estado), y la relación muchos a muchos entre estudiantes y subcategorías se resuelve con **subcategoria_estudiante**, donde se registra la fecha de unión y el orden en la lista. Sobre esta base se construye el módulo de asistencia: **sesiones_asistencia** define cada

sesión de entrenamiento (subcategoría, profesor, fecha, horario y observaciones) y **asistencias** registra, para cada estudiante, el estatus de asistencia en esa sesión.

En el ámbito financiero, la tabla **mensualidades** representa las obligaciones de pago derivadas de cada matrícula (mes, año, valor y estado), mientras que **estado_mensualidad** conserva el historial de cambios de estado a lo largo del tiempo. Finalmente, la tabla **pagos** registra cada cobro realizado (monto, fecha, método de pago, referencia, motivo de anulación, usuario responsable e identificador de idempotencia), lo que asegura trazabilidad y control sobre las transacciones. El esquema resultante se muestra en la Figura 18.

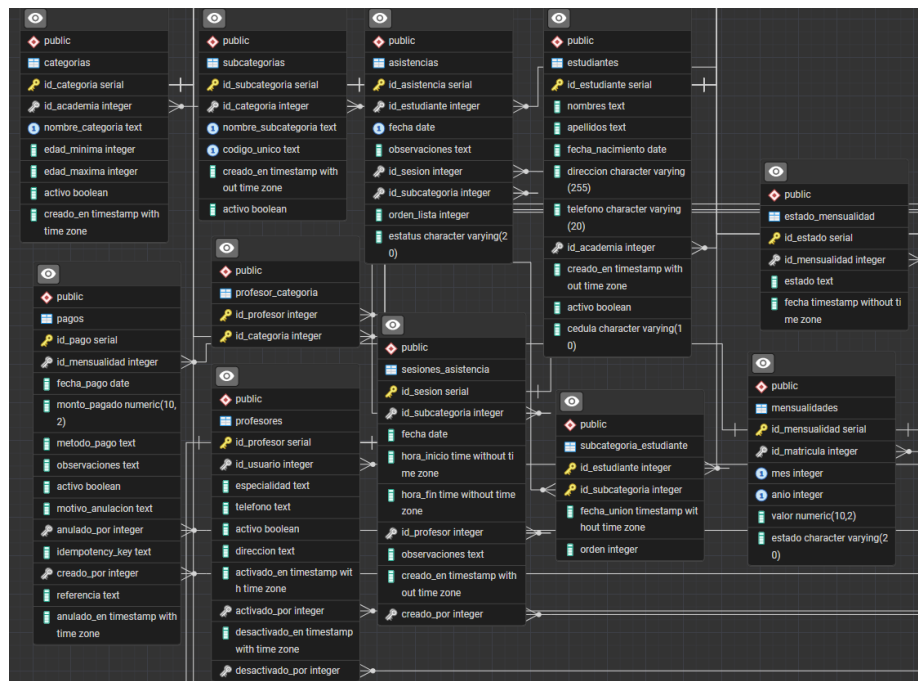


Figura 18. Diagrama de la estructura principal de los usuarios y roles.

Las tablas para utilizar, como se visualiza en la Figura 16 y la Figura 17, son las necesarias para el desarrollo del sistema, las mismas que son:

- usuarios
- roles
- permisos
- rol_permiso

- auditoria
- categorias
- subcategorias
- profesores
- profesor_categoria
- estudiantes
- subcategoria_estudiante
- sesiones_asistencia
- asistencias
- mensualidades
- estado_mensualidad
- pagos

Cabe señalar que el modelo completo de la base de datos incorpora además otras tablas complementarias, orientadas a configuración, trazabilidad y funcionalidades de apoyo, que enriquecen el sistema, pero no son esenciales para el núcleo funcional descrito en este apartado.

- Creación del proyecto backend (API REST)

El entorno de desarrollo basado en **Node.js** ofrece una gran variedad de herramientas y bibliotecas que facilitan el trabajo en proyectos de distintos niveles de complejidad. Su gestor de paquetes, **npm**, permite incorporar soluciones creadas por la comunidad o por el propio ecosistema de Node, lo que ahorra tiempo y evita tener que desarrollar desde cero funcionalidades que ya están resueltas y comprobadas. Esto convierte a Node.js en una plataforma flexible, práctica y adecuada para proyectos modernos.

Al iniciar un proyecto mediante el comando básico de npm, se genera automáticamente la estructura inicial del backend, incluyendo el archivo **package.json**, donde se concentran la configuración principal, las dependencias y los scripts más importantes. A partir de esta base, es posible construir APIs bajo el enfoque REST, logrando que aplicaciones móviles, sistemas externos o servicios internos se comuniquen de forma ordenada y segura con el servidor. Así, se establece un punto de partida claro y organizado para continuar con el desarrollo modular del sistema como se puede ver en la Figura 19.

```
PS C:\Users\roger\Desktop\tesis> cd .\backend-app-porto\  
PS C:\Users\roger\Desktop\tesis\backend-app-porto> npm init -y  
Wrote to C:\Users\roger\Desktop\tesis\backend-app-porto\package.json:  
  
{  
  "name": "backend-app-porto",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC"  
}
```

Figura 19. Proyecto Node.js

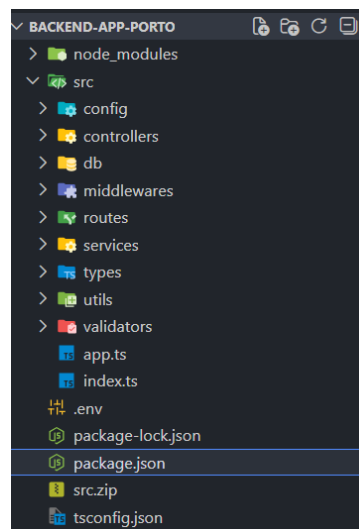


Figura 20. Estructura del backend.

En la Figura 20 se observa la estructura general del backend desarrollado con Node.js y TypeScript, organizada de manera modular para facilitar su mantenimiento y crecimiento. Toda la lógica del servidor se concentra dentro de la carpeta **src**, mientras que las dependencias, configuraciones principales y scripts del proyecto se administran desde el archivo *package.json*. Esta organización permite mantener separado lo esencial del proyecto respecto a los archivos temporales y del entorno.

Dentro de *src*, como se aprecia en la figura, se disponen carpetas especializadas según su función. Por ejemplo, **controllers** contiene la lógica que atiende las solicitudes entrantes, **services** gestiona la parte operativa y las reglas de negocio, y **routes** define los distintos endpoints de la API. Otras carpetas como **middlewares**, **validators**, **config** y **utils** complementan el funcionamiento al encargarse de procesos clave como seguridad, validaciones, configuración global y funciones auxiliares. Esta distribución clara permite que cada parte del sistema mantenga una responsabilidad concreta.

Finalmente, los archivos **app.ts** e **index.ts**, visibles también en la Figura 19, funcionan como el punto de arranque del servidor, integrando todas las rutas, servicios y configuraciones definidas previamente. Junto a ellos se encuentran archivos como **tsconfig.json** y *.env*, indispensables para la compilación y configuración del entorno. En conjunto, esta estructura ofrece un proyecto ordenado, comprensible y preparado para escalar conforme el sistema crezca.

- Dependencias requeridas

```
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
  "bcrypt": "^6.0.0",
  "cloudinary": "^2.7.0",
  "cors": "^2.8.5",
  "dayjs": "^1.11.13",
  "dotenv": "^17.2.1",
  "express": "^5.1.0",
  "jsonwebtoken": "^9.0.2",
  "multer": "^2.0.2",
  "pg": "^8.16.3",
  "zod": "^4.0.14"
},
```

Figura 21. Dependencias requeridas

En la Figura 21 se muestran las principales dependencias incluidas en el archivo `package.json`, las cuales conforman el núcleo funcional del backend. Cada una de estas librerías cumple un rol específico dentro de la aplicación y permite extender las capacidades del servidor sin necesidad de implementar manualmente funciones ya estandarizadas dentro del ecosistema de Node.js.

Entre estas dependencias se incluyen herramientas fundamentales como **express**, que provee la estructura necesaria para construir la API; **cors**, que habilita el intercambio de datos entre diferentes dominios; y **dotenv**, utilizada para manejar de forma segura las variables de entorno del proyecto. A esto se suma **pg**, el cliente oficial para interactuar con PostgreSQL, lo que facilita la comunicación con la base de datos de manera eficiente.

Además, la Figura 21 muestra otras librerías orientadas a la seguridad, validación y manejo de datos, como **bcrypt** para el cifrado de contraseñas, **jsonwebtoken** para la gestión de autenticación por tokens, **multer** para la carga de archivos y **zod**, empleado para validar la estructura de los datos recibidos. También se incorpora **cloudinary** para la gestión de archivos multimedia y **dayjs** para el manejo de fechas. Con este conjunto de dependencias, el backend obtiene una base robusta y flexible para cubrir las necesidades del sistema.

- Archivos de configuración

```
src > index.ts > ...
1  import app from './app';
2  import dotenv from 'dotenv';
3
4  dotenv.config();
5
6  const PORT = process.env.PORT || 3000;
7
8  app.listen(PORT, () => {
9    console.log(`✅ Servidor corriendo en http://localhost:${PORT}`);
10 });
11
```

Figura 22. Archivo de configuración `index.ts`.

En la Figura 22 se muestra el archivo `index.ts`, el cual actúa como punto de entrada principal del servidor. En este módulo se realiza la carga inicial de las variables de entorno mediante la librería **dotenv**, se define el puerto sobre el cual se ejecutará la aplicación y finalmente se inicializa el servidor utilizando `app.listen()`. Este archivo cumple la función de enlazar toda la configuración previa y poner en marcha la API, generando el mensaje de confirmación que indica que el servicio está activo y disponible en la dirección correspondiente.

```
src > app.ts > ...
1 > import express from 'express'; ...
28
29 const app = express();
30 const allowLocal = /^https?:\/\/(localhost|127\.0\.0\.1|:\d+)?$/i;
31 const corsOptions = {
32   origin(origin: string | undefined, callback: (err: Error | null, allow?: boolean) => void) {
33     if (!origin || allowLocal.test(origin)) {
34       return callback(null, true);
35     }
36     return callback(new Error('CORS: Origin no permitido'));
37   },
38   credentials: true,
39   methods: ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'OPTIONS'],
40   allowedHeaders: ['Content-Type', 'Authorization', 'Idempotency-Key', 'Accept'],
41   exposedHeaders: ['X-Total-Count', 'Authorization'],
42   optionsSuccessStatus: 204,
43 };
44 app.use(cors(corsOptions));
45 app.use(express.json());
46
47 /* ====== Routes ====== */
48 app.use('/auth', authRoutes);
49 app.use('/protegida', protegidaRoutes);
50
51 app.use('/usuarios', usuariosRoutes);
52 app.use('/profesores', profesoresRoutes);
53 app.use('/estudiantes', estudiantesRoutes);
```

Figura 23. Archivo de configuración `app.ts`.

La Figura 23 presenta el contenido del archivo `app.ts`, donde se configura la aplicación de Express antes de exponer cualquier ruta. En este archivo se define un conjunto de reglas para el manejo de CORS, permitiendo únicamente solicitudes provenientes de orígenes autorizados y estableciendo los métodos, encabezados y credenciales que la API puede recibir. Además, se habilita el uso de `express.json()` para procesar solicitudes con cuerpo en formato JSON, garantizando que los datos enviados desde el cliente puedan interpretarse correctamente.

En este mismo archivo se observa la declaración y carga de las rutas principales del sistema, organizadas según los módulos funcionales del proyecto, como autenticación, usuarios, profesores y estudiantes. Esta estructura modular permite que cada conjunto de endpoints esté agrupado y sea administrado de forma independiente, asegurando un desarrollo más ordenado y mantenible. En conjunto, estas configuraciones establecen el funcionamiento base del servidor y preparan el entorno para la comunicación segura y estructurada entre el backend y los diferentes clientes del sistema.

- Mockups de interfaces

Paralelamente, se elaboraron mockups de las principales pantallas del sistema: login, panel principal del administrador, módulos de estudiantes, mensualidades, pagos y reportes. Estos diseños, mostrados en figuras análogas a las Figuras 23 a 31, sirvieron como guía visual para el desarrollo del frontend.

The mockup shows a login interface with the title "Iniciar Sesión" in blue. Below the title are two input fields: "Correo Electrónico" with the placeholder "ejemplo@correo.com" and "Contraseña" with the placeholder "Ingresa tu contraseña". A blue "Entrar" button is positioned below the password field. At the bottom, there is a link that says "¿Olvidaste tu contraseña?".

Figura 24. Mockup del inicio de sesión.

En la Figura 24 se presenta el mockup correspondiente a la pantalla de Inicio de Sesión del sistema. Esta interfaz fue diseñada con un enfoque minimalista y centrado en la facilidad de uso, permitiendo que el usuario acceda rápidamente a su cuenta mediante un formulario claro y directo. En la parte superior se muestra el título principal, seguido de los campos para ingresar el correo electrónico y la contraseña, ambos acompañados de ejemplos y textos guía que facilitan la comprensión del usuario.



Mi Perfil

Nombre

Correo Electrónico

Teléfono

Guardar

Figura 25. Mockup del perfil.

En la Figura 25 se observa el mockup de la pantalla Mi Perfil, donde el usuario puede visualizar y actualizar su información personal registrada en el sistema. El diseño presenta una vista limpia y centrada en la facilidad de edición, mostrando en la parte superior un ícono representativo del usuario seguido de los campos principales: nombre, correo electrónico y número telefónico. Cada campo está dispuesto de forma ordenada y utiliza placeholders para guiar al usuario sobre el contenido esperado. Al final del formulario se encuentra el botón Guardar, que permite confirmar los cambios realizados. Esta pantalla busca ofrecer una experiencia clara y directa para la gestión de datos personales.

Gestión de Roles

+ Agregar Rol

Rol	Descripción	Acciones
Administrador	Acceso completo al sistema	<div>EditarEliminar</div>
Usuario	Acceso básico	<div>EditarEliminar</div>
Editor	Puede editar contenido	<div>EditarEliminar</div>
Supervisor	Puede supervisar y aprobar	<div>EditarEliminar</div>

Figura 26. Mockup de la pantalla roles.

La Figura 26 presenta el mockup correspondiente a la sección Gestión de Roles. En esta interfaz se muestra una tabla que agrupa los roles existentes dentro del sistema, junto con su descripción y las acciones disponibles para cada uno. En la parte superior se incluye el botón Agregar Rol, que permite crear nuevos perfiles de acceso. La tabla organiza la información de manera sencilla, destacando los botones de acción Editar y Eliminar, los cuales facilitan la administración de permisos y responsabilidades. Este diseño permite a los administradores gestionar de forma eficiente la estructura de roles que regula el acceso a las distintas funcionalidades del sistema

Gestión de Usuarios

+ Agregar Usuario

Usuario	Correo	Rol	Estado	Acciones
<div>JP</div> Juan Pérez	juan@email.com	Administrador	Activo	<div>EditarEliminar</div>
<div>MG</div> María García	maria@email.com	Editor	Activo	<div>EditarEliminar</div>
<div>CL</div> Carlos López	carlos@email.com	Usuario	Inactivo	<div>EditarEliminar</div>
<div>AS</div> Ana Silva	ana@email.com	Supervisor	Activo	<div>EditarEliminar</div>

Figura 27. Mockup de la pantalla de usuarios.

En la Figura 27 se ilustra la pantalla de Gestión de Usuarios, donde se listan los usuarios registrados en el sistema junto con datos relevantes como nombre, correo

electrónico, rol asignado y estado actual. La interfaz incorpora un botón de Agregar Usuario para permitir la creación de nuevos registros. La tabla presenta un diseño visualmente claro que facilita la identificación del estado de cada usuario mediante etiquetas de color, como Activo o Inactivo, lo que mejora la comprensión rápida de la información. Cada fila incluye además botones para editar o eliminar el registro, proporcionando un control completo sobre la administración de cuentas dentro del sistema.

Gestión de Profesores

+ Agregar Profesor

Profesor	Correo	Teléfono	Acciones	
<div>LM</div> <div>Luis Martínez</div>	luis.martinez@academia.com	+593 98 765 4321	<div>Editar</div>	<div>Eliminar</div>
<div>SR</div> <div>Sofía Rodríguez</div>	sofia.rodriguez@academia.com	+593 99 123 4567	<div>Editar</div>	<div>Eliminar</div>
<div>PC</div> <div>Pedro Castro</div>	pedro.castro@academia.com	+593 97 654 3210	<div>Editar</div>	<div>Eliminar</div>
<div>MV</div> <div>María Vega</div>	maria.vega@academia.com	+593 96 789 0123	<div>Editar</div>	<div>Eliminar</div>

Figura 28. Mockup de la pantalla de profesores.

En la Figura 28 se presenta la pantalla de Gestión de Profesores, donde se visualiza el listado de entrenadores registrados en la academia. La interfaz permite observar información relevante como nombre, correo institucional y número telefónico, organizada en una tabla de fácil lectura. En la parte superior se incluye el botón Agregar Profesor, el cual habilita la creación de un nuevo registro. Asimismo, cada profesor cuenta con opciones de Editar y Eliminar, facilitando la administración completa del personal docente. El diseño se orienta a ofrecer una gestión ágil y ordenada del cuerpo de entrenadores.

Gestión de Categorías

+ Agregar Categoría		
Categoría	Descripción	Acciones
Sub-8	Categoría de niños de 8 años o menos	<button>Editar</button> <button>Eliminar</button>
Sub-10	Categoría de niños de 10 años o menos	<button>Editar</button> <button>Eliminar</button>
Sub-12	Categoría de niños de 12 años o menos	<button>Editar</button> <button>Eliminar</button>
Sub-14	Categoría de adolescentes de 14 años o menos	<button>Editar</button> <button>Eliminar</button>

Figura 29. Mockup de la pantalla de categorías.

La Figura 29 muestra el módulo de Gestión de Categorías, donde se administran los distintos grupos o niveles de formación de la academia, tales como Sub-8, Sub-10, Sub-12 y Sub-14. Cada categoría se acompaña de una breve descripción para aclarar el rango de edades correspondiente. Desde el botón Agregar Categoría es posible añadir nuevos niveles según las necesidades de la institución. Además, cada fila incorpora botones de Editar y Eliminar que permiten mantener actualizada la estructura organizativa del sistema. La presentación es clara y facilita la toma de decisiones administrativas.

Gestión de Subcategorías

+ Agregar Subcategoría			
Subcategoría	Categoría	Descripción	Acciones
Sub-8 Mañana	Sub-8	Entrenamiento matutino 8:00 - 10:00	<button>Editar</button> <button>Eliminar</button>
Sub-8 Tarde	Sub-8	Entrenamiento vespertino 15:00 - 17:00	<button>Editar</button> <button>Eliminar</button>
Sub-10 Mañana	Sub-10	Entrenamiento matutino 8:00 - 10:00	<button>Editar</button> <button>Eliminar</button>
Sub-10 Tarde	Sub-10	Entrenamiento vespertino 15:00 - 17:00	<button>Editar</button> <button>Eliminar</button>
Sub-12 Tarde	Sub-12	Entrenamiento vespertino 16:00 - 18:00	<button>Editar</button> <button>Eliminar</button>

Figura 30. Mockup de la pantalla de subcategorías.

En la Figura 30 se observa la interfaz de Gestión de Subcategorías, que detalla los grupos internos dentro de cada categoría principal, por ejemplo, horarios matutinos o vespertinos. La tabla muestra la subcategoría, la categoría a la que pertenece y una descripción que especifica el horario o modalidad de entrenamiento. El sistema permite agregar nuevas subcategorías mediante el botón correspondiente, así como editar o eliminar las existentes. Esta vista proporciona una estructura precisa de la organización interna de los grupos de trabajo de la academia.

Gestión de Estudiantes

+ Agregar Estudiante

Estudiante	Matricula	Curso	Subcategoría	Teléfono	Acciones	
AG Andrea González	EST-001	Sub-8	Sub-8 Mañana	+593 98 111 2222	Editar	Eliminar
DT Diego Torres	EST-002	Sub-10	Sub-10 Tarde	+593 99 333 4444	Editar	Eliminar
LR Laura Reyes	EST-003	Sub-12	Sub-12 Tarde	+593 97 555 6666	Editar	Eliminar
CM Carlos Morales	EST-004	Sub-8	Sub-8 Tarde	+593 96 777 8888	Editar	Eliminar

Figura 31. Mockup de la pantalla de estudiantes.

La Figura 31 presenta el módulo de Gestión de Estudiantes. En esta pantalla se listan los alumnos registrados, mostrando su nombre, número de matrícula, categoría asignada, subcategoría correspondiente y contacto telefónico. A través del botón Agregar Estudiante se pueden incorporar nuevos registros al sistema. Cada estudiante dispone de opciones de Editar y Eliminar, lo que permite una administración dinámica y actualizada de la información académica. La tabla está diseñada para ser intuitiva y permitir una rápida identificación del estado de los estudiantes.

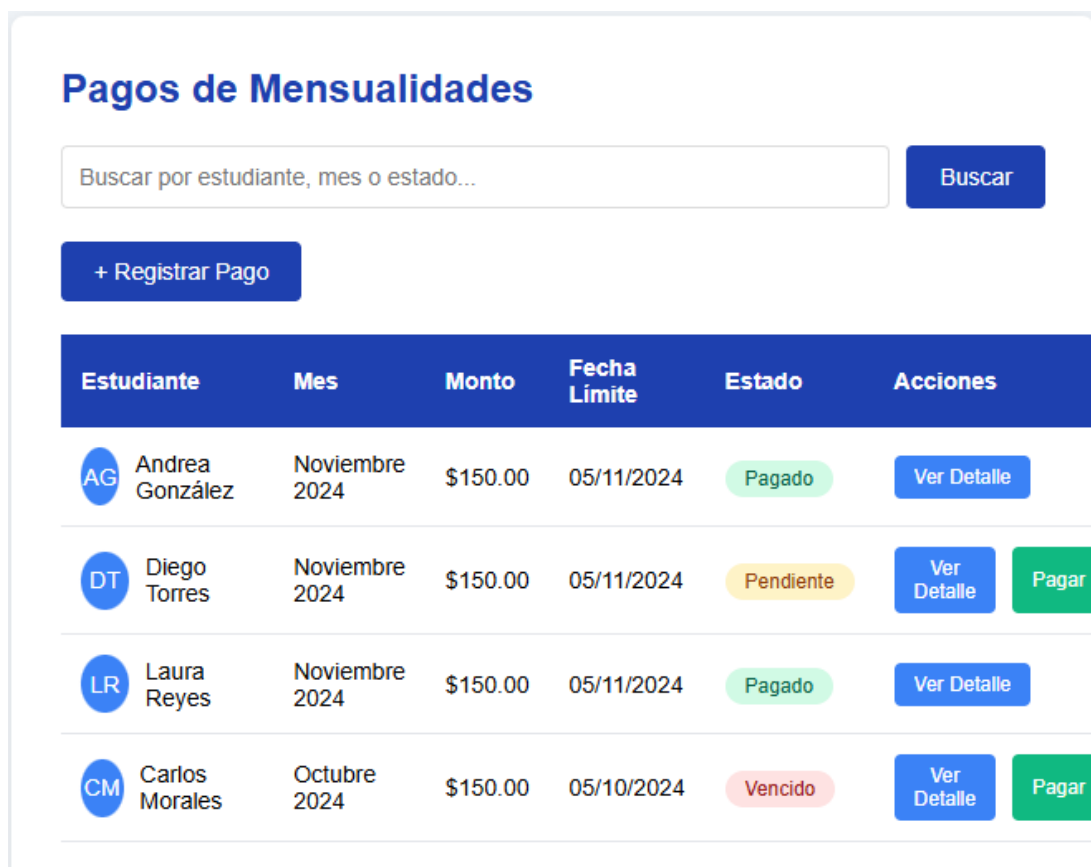


Figura 32. Mockup de la pantalla de pagos de mensualidades.

La Figura 32 representa la pantalla de Pagos de Mensualidades, uno de los módulos más importantes del sistema. Aquí se despliega una lista de estudiantes junto con el mes correspondiente, el monto a pagar, la fecha límite y el estado actual del pago, el cual se distingue mediante etiquetas visuales como Pagado, Pendiente o Vencido. La interfaz incluye una barra de búsqueda para filtrar por estudiante, mes o estado, así como un botón Registrar Pago que permite añadir nuevos registros. Cada fila incorpora acciones como Ver Detalle o Pagar, facilitando un seguimiento claro y ordenado de las obligaciones económicas de cada alumno.

Es importante señalar que las interfaces presentadas corresponden a mockups preliminares, desarrollados con el objetivo de visualizar la estructura funcional del sistema y definir el flujo general de uso. Estos diseños no representan la versión final de la aplicación, ya que en etapas posteriores pueden sufrir ajustes estéticos, cambios en la disposición de elementos o incorporación y eliminación de funcionalidades, conforme avance el desarrollo y se realicen validaciones con los usuarios. Por tanto,

los mockups funcionan como una guía conceptual que orienta la construcción del sistema, permitiendo identificar mejoras y asegurar que la experiencia de uso sea coherente con las necesidades de la academia.

- Creación del proyecto Flutter (Frontend)

Con los mockups definidos, se inició la creación del proyecto en **Flutter**, configurando la estructura de carpetas, la navegación y los componentes base (formularios, tablas, botones).

La Figura 33 representa el proyecto frontend y su organización inicial.

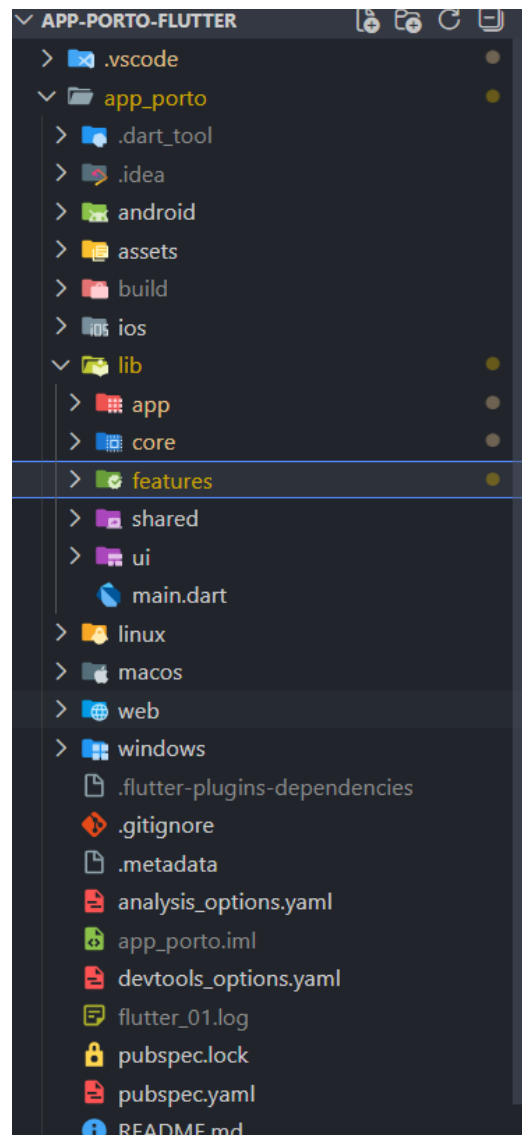


Figura 33. Estructura del proyecto en Flutter.

b. Sprint 2

En el segundo sprint se inició el desarrollo funcional del backend y del frontend, enfocándose en la autenticación y los módulos base:

- Módulo de autenticación – Backend

Se implementó el flujo de **login**: recepción de credenciales, verificación de la contraseña cifrada, generación de token JWT y respuesta con la información de usuario y rol.

```
export const loginUser = async (correo: string, contrasena: string) => {
  const r = await pool.query(`SELECT * FROM usuarios WHERE correo = $1 LIMIT 1`, [correo]);

  if (r.rows.length === 0) {
    throw { message: 'Correo o contraseña incorrectos', code: 401 };
  }

  const usuario = r.rows[0];

  if (usuario.activo === false) {
    throw { message: 'Usuario inactivo. Contacta al administrador.', code: 403 };
  }

  const ok = await comparePassword(contrasena, usuario.contrasena_hash);
  if (!ok) {
    throw { message: 'Correo o contraseña incorrectos', code: 401 };
  }

  const token = generarToken({
    id_usuario: usuario.id_usuario,
    nombre: usuario.nombre,
    correo: usuario.correo,
    id_rol: usuario.id_rol
  });

  const creado_en = dayjs().toDate();
  const expira_en = dayjs().add(1, 'day').toDate();

  await pool.query(
    `INSERT INTO tokens (id_usuario, creado_en, expira_en, token) VALUES ($1, $2, $3, $4)`,
    [usuario.id_usuario, creado_en, expira_en, token]
  );

  await registrarAuditoria(usuario.id_usuario, 'Inicio de sesión', 'tokens');
}
```

Figura 34. Función de login en el backend.

En la Figura 34 se muestra un fragmento del código correspondiente al proceso de autenticación de usuarios dentro del sistema. En este ejemplo se evidencia el uso de consultas parametrizadas mediante el método `pool.query`, donde los valores proporcionados por el usuario son enviados como parámetros seguros, evitando así

vulnerabilidades como la inyección SQL. La consulta obtiene únicamente el registro asociado al correo proporcionado, aplicando un límite de una fila para optimizar él se valida con los siguientes pasos: primero se verifica que el usuario exista y esté activo dentro de la plataforma. Posteriormente, la contraseña ingresada se compara con el hash almacenado en la base de datos utilizando una función segura de verificación. Si cualquiera de estas comprobaciones falla, se devuelve un mensaje de error adecuado, garantizando que el sistema no exponga información sensible.

Una vez validadas las credenciales, se genera un token JWT con los datos esenciales del usuario, el cual será utilizado para mantener la sesión activa en futuras solicitudes. Adicionalmente, se registra dicho token en la tabla correspondiente utilizando otra consulta parametrizada y se almacena un registro en la tabla de auditoría, lo que permite mantener trazabilidad del evento de inicio de sesión. Este fragmento refleja la integración entre la capa de acceso a datos, la seguridad del sistema y los mecanismos de control interno.

- Módulo de autenticación – Frontend

```
Future<({String token, Map<String, dynamic> user})> login({
  required String correo,
  required String contrasena,
}) async {
  final res = await _http.post(Endpoints.authLogin, body: {
    'correo': correo,
    'contrasena': contrasena,
  }, headers: {});

  String? _findJwt(dynamic v) {
    final re = RegExp(r'^[A-Za-z0-9\\-\\_]=+\\.([A-Za-z0-9\\-\\_]=+\\.([A-Za-z0-9\\-\\_]=+\\.)+$)');
    if (v is String && re.hasMatch(v)) return v;
    if (v is Map) for (final e in v.entries) { final t = _findJwt(e.value); if (t != null) return t; }
    if (v is Iterable) for (final e in v) { final t = _findJwt(e); if (t != null) return t; }
    return null;
  }

  String? token;
  Map<String, dynamic>? user;

  if (res is Map) {
    token = (res['token'] ?? res['access_token']) as String?;
    user = (res['usuario'] ?? res['user']) as Map<String, dynamic>;

    final data = res['data'];
    if (token == null && data is Map) {
      token ??= (data['token'] is String)
        ? data['token'] as String
        : (data['token'] is Map && data['token']['token'] is String)
          ? data['token']['token'] as String
          : null;
      user ??= (data['usuario'] ?? data['user']) as Map<String, dynamic>;
    }
    token ??= _findJwt(res);
  }
}
```

Figura 35. Función de login en el frontend.

En la Figura 35 se muestra el fragmento de código correspondiente al proceso de autenticación implementado en el cliente desarrollado con Flutter. En esta función se realiza la solicitud al endpoint de inicio de sesión del backend, enviando las credenciales del usuario (correo electrónico y contraseña) mediante una petición POST. La respuesta del servidor es procesada de forma segura y flexible, permitiendo identificar diferentes estructuras posibles en el objeto devuelto por la API.

El código incluye un método auxiliar encargado de localizar el token JWT dentro de la respuesta, independientemente de si este se encuentra directamente en el nivel principal o anidado dentro de otras estructuras. Esto garantiza una mayor compatibilidad frente a variaciones en el formato de respuesta, convirtiendo el proceso de lectura del token en una tarea robusta. Además, se verifica la presencia de la información del usuario autenticado, extrayendo sus datos de forma tipada y manteniendo la coherencia del estado interno de la aplicación.

El sistema combina tanto la búsqueda directa como la exploración recursiva para asegurar que el token sea encontrado correctamente, incluso en casos donde el backend pueda retornar respuestas más complejas. De esta manera, el cliente logra gestionar de forma confiable la recepción del token y los datos del usuario, sentando las bases para la administración de sesiones y la protección de rutas dentro de la aplicación.

c. *Sprint 3*

Una vez consolidado el inicio de sesión, se avanzó hacia los módulos centrales del sistema:

- Controladores y servicios para estudiantes y profesores (backend)

```
export async function crearEstudiante({
  nombres: string;
  apellidos: string;
  cedula?: string | null;
  fecha_nacimiento?: string | null;
  telefono?: string | null;
  direccion?: string | null;
  id_academia?: number | null;
}, {
  id_usuario: number
}) {
  const {
    nombres,
    apellidos,
    cedula,
    fecha_nacimiento,
    telefono,
    direccion,
    id_academia,
  } = data;
```

Figura 36. Servicio de crear un estudiante en el backend.

```

export async function postEstudiante(req: Request, res: Response) {
  try {
    const uid = (req as any).user?.id_usuario ?? 0;
    const body = req.body || {};

    if (body.estudiante || body.matricula) {
      const estudiante = body.estudiante || {};
      const matricula = body.matricula || {};

      if (!matricula.id_categoria) {
        return res.status(400).json({ error: 'id_categoria es requerido en matricula' });
      }
      const out = await MatSvc.inscribirEstudianteConMatricula(estudiante, matricula, uid);
      return res.status(201).json(out);
    }

    const row = await Svc.crearEstudiante(body, uid);
    return res.status(201).json(row);
  } catch (e: any) {
    const msg = mapPgCedulaMessage(e);
    const code = HttpStatusFromError(e);
    return res.status(code).json({ error: msg || e.message || 'Error' });
  }
}

```

Figura 37. Controller del servicio crear un estudiante.

En la Figura 36 se aprecia uno de los módulos centrales del backend encargado de la creación de estudiantes, donde se procesa la información enviada desde el cliente, se realiza una desestructuración clara de los datos recibidos y se aplican validaciones iniciales, como la verificación del formato de la cédula antes de proceder con el registro. Este módulo emplea consultas SQL parametrizadas para insertar de forma segura los datos en la base de datos, garantizando integridad, prevención de inyecciones SQL y coherencia con las reglas definidas en el modelo. Del mismo modo, en el controlador correspondiente también mostrado en la Figura 37 se identifica la lógica que decide si el estudiante debe crearse de manera individual o junto con una matrícula inicial, validando que se proporcionen los datos indispensables y delegando la operación a los servicios especializados. Ambos componentes trabajan de forma conjunta para asegurar que el registro de estudiantes se realice de manera robusta, consistente y alineada con el flujo administrativo de la academia.

```

export const crearProfesor = async (data: {
  id_usuario: number,
  especialidad?: string,
  telefono?: string,
  direccion?: string,
  activo?: boolean
}, actorId?: number) => {
  const { id_usuario, especialidad, telefono, direccion, activo = true } = data;

  const client = await pool.connect();
  try {
    await client.query('BEGIN');
    await client.query(`UPDATE usuarios SET id_rol = 2 WHERE id_usuario = $1 AND id_rol <> 2`, [id_usuario]);

    const up = await client.query(`
      INSERT INTO profesores (id_usuario, especialidad, telefono, direccion, activo, activado_en, activado_por)
      VALUES ($1,$2,$3,$4,$5, CASE WHEN $5 THEN NOW() END, CASE WHEN $5 THEN $6 END)
      ON CONFLICT (id_usuario) DO UPDATE
      SET activo
        = EXCLUDED.activo,
        activado_en = CASE WHEN EXCLUDED.activo THEN NOW() ELSE profesores.activado_en END,
        activado_por = CASE WHEN EXCLUDED.activo THEN $6 ELSE profesores.activado_por END,
        especialidad = COALESCE(EXCLUDED.especialidad, profesores.especialidad),
        telefono = COALESCE(EXCLUDED.telefono, profesores.telefono),
        direccion = COALESCE(EXCLUDED.direccion, profesores.direccion)
      RETURNING id_profesor;
    `, [id_usuario, especialidad ?? null, telefono ?? null, direccion ?? null, activo, actorId ?? null]);

    await registrarAuditoria(actorId ?? id_usuario, 'Registro/Upsert de profesor', 'profesores');

    const row = await client.query(`
      SELECT p.*, u.nombre AS nombre_usuario, u.correo, u.id_rol, u.cedula
      FROM profesores p
      JOIN usuarios u ON p.id_usuario = u.id_usuario
      WHERE p.id_profesor = $1
    `, [up.rows[0].id_profesor]);
  }

```

Figura 38. Servicio para crear un profesor.

```

export const postProfessor = async (req: Request, res: Response) => {
  const actorId = (req as any).user?.id_usuario as number | undefined;
  const row = await svc.crearProfesor(req.body, actorId);
  res.status(201).json(row);
};

```

Figura 39. Controller de crear un profesor.

En la Figura 38 se muestra el módulo encargado del registro y actualización de profesores dentro del sistema, el cual constituye una de las operaciones centrales del backend. Este servicio implementa un proceso transaccional que actualiza el rol del usuario, realiza una inserción o actualización en la tabla de profesores mediante una sentencia SQL parametrizada y registra la acción en el módulo de auditoría, garantizando integridad y trazabilidad en todas las operaciones. Así mismo, en la Figura 39, se observa el controlador correspondiente, que recibe la solicitud desde el cliente, obtiene la identidad del usuario que ejecuta la acción y delega la lógica principal al servicio especializado, devolviendo finalmente una respuesta HTTP con el resultado del proceso. En conjunto, estos componentes aseguran un manejo

consistente, seguro y estructurado en la gestión del personal docente dentro del sistema.

- Módulo de categorías, subcategorías y matrículas

```
export const crearCategoria = async (data: {  
  nombre_categoria: string,  
  edad_minima?: number,  
  edad_maxima?: number,  
  id_usuario: number  
) => {  
  const { nombre_categoria, edad_minima, edad_maxima, id_usuario } = data;  
  
  const nombre = String(nombre_categoria ?? '').trim();  
  if (!nombre) throw { status: 400, message: 'nombre_categoria es requerido' };  
  
  await assertNombreUnico(nombre);  
  
  const result = await pool.query(`  
    INSERT INTO categorias (nombre_categoria, edad_minima, edad_maxima, activo, creado_en)  
    VALUES ($1, $2, $3, TRUE, NOW())  
    RETURNING *  
  `,  
    [nombre, edad_minima ?? null, edad_maxima ?? null]  
  );  
  
  await registrarAuditoria(id_usuario, 'Registro de categoría', 'categorias', result.rows[0].id_categoria);  
  return result.rows[0];  
};
```

Figura 40. Servio de crear categoría.

```
export const crearSubcategoria = async (  
  data: { id_categoria: number, nombre_subcategoria: string, codigo_unico: string },  
  id_usuario: number  
) => {  
  const { id_categoria, nombre_subcategoria, codigo_unico } = data;  
  const result = await pool.query(`  
    INSERT INTO subcategorias (id_categoria, nombre_subcategoria, codigo_unico)  
    VALUES ($1, $2, $3)  
    RETURNING *  
  `, [id_categoria, nombre_subcategoria, codigo_unico]);  
  
  await registrarAuditoria(id_usuario, 'Registro de subcategoría', 'subcategorias');  
  return result.rows[0];  
};
```

Figura 41. Servicio de crear subcategoría.

```

export const inscribirEstudianteConMatricula = async (
  est: EstudianteInput,
  mat: MatriculaInput,
  id_usuario: number
) => {
  const client = await pool.connect();
  try {
    await client.query('BEGIN');

    // Buscar si ya existe el estudiante (heurística básica)
    const ex = await client.query(
      `
      SELECT *
      FROM estudiantes
      WHERE lower(nombres)=lower($1)
         AND lower(apellidos)=lower($2)
         AND COALESCE(fecha_nacimiento::text,'') = COALESCE($3,'')
         AND COALESCE(id_academia,0) = COALESCE($4,0)
      LIMIT 1
      `,
      [est.nombres?.trim(), est.apellidos?.trim(), est.fecha_nacimiento || null, est.id_academia || null]
    );

    let id_estudiante: number;

    if (ex.rows.length > 0) {
      id_estudiante = ex.rows[0].id_estudiante;

      // no permitir 2da matricula
      const ya = await client.query(
        `SELECT 1 FROM matriculas WHERE id_estudiante = $1 LIMIT 1`,
        [id_estudiante]
      );
      if (ya.rows.length > 0) {
        await client.query('ROLLBACK');
        throw { code: 409, message: 'El estudiante ya posee una matrícula' };
      }
    }
  }
}

```

Figura 42. Servicio de crear matrícula.

En las Figuras 40 y 41 se observa la implementación de los módulos encargados de la creación de categorías y subcategorías, elementos fundamentales para la organización interna del sistema académico. En el caso de la creación de categorías, el backend valida previamente que el nombre de la categoría no esté vacío y verifica su unicidad antes de ejecutar la inserción en la base de datos mediante una consulta SQL parametrizada. De igual manera, el módulo para registrar subcategorías recibe el identificador de la categoría principal y el nombre de la subcategoría, asegurando que la información se almacene correctamente y registrando cada acción realizada en el sistema a través del módulo de auditoría. Estos componentes permiten mantener una estructura ordenada, evitando duplicidades y garantizando la integridad de los datos esenciales para la organización de los estudiantes.

Por su parte, la Figura 42 muestra el proceso completo de inscripción de estudiantes junto con su matrícula, uno de los módulos más completos dentro del backend.

Este procedimiento incluye la apertura de una transacción para garantizar la consistencia de los datos, una verificación heurística para determinar si el estudiante ya existe en el sistema y la comprobación de que no posea una matrícula previa. Solo después de superar estas validaciones se procede al registro del estudiante y su correspondiente matrícula, asegurando que no se generen duplicados ni inconsistencias. Gracias a esta lógica cuidadosamente implementada, el sistema

garantiza un manejo seguro y preciso de los datos académicos, preservando la coherencia entre las distintas entidades que conforman la base de datos.

- Interfaces de administración en Flutter

En el frontend se crearon vistas para gestionar estudiantes y matrículas, reflejando lo definido en los mockups. Se implementaron tablas con filtros y formularios de alta/edición de registros, consumiendo las APIs a través de servicios especializados.

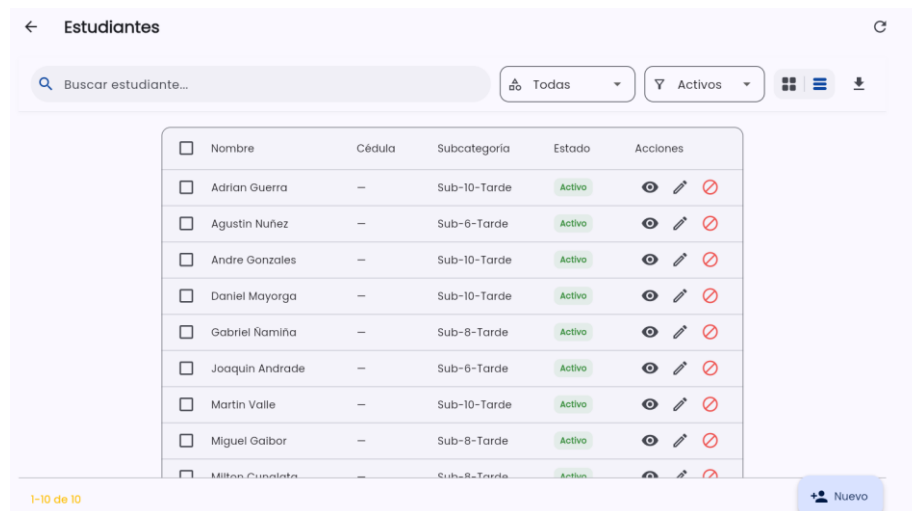


Figura 43. Pantalla de estudiantes.

The screenshot shows a form titled 'Registrar Nuevo Estudiante' with the instruction 'Complete el formulario con los datos del estudiante'. The form is divided into two main sections: 'Datos Personales' and 'Datos de Matrícula'. The 'Datos Personales' section includes fields for 'Nombres *', 'Apellidos *', 'Cédula (Ecuador)' with a checkmark, 'Fecha de Nacimiento' with a calendar icon, 'Dirección', and 'Teléfono'. The 'Datos de Matrícula' section includes a 'Categoría *' dropdown, a 'Subcategoría' dropdown with the text 'Primero seleccione categoría', 'Ciclo', and 'Fecha de Matrícula' with a calendar icon. At the bottom right, there are two buttons: 'Cancelar' and 'Guardar Inscripción'.

Figura 44. Pantalla para crear un estudiante con su matrícula.

En la Figura 43 se muestra la interfaz principal del módulo de *Estudiantes*, donde se puede visualizar de manera organizada el listado completo de alumnos registrados en

la academia. Esta vista permite filtrar por nombre, categoría y estado, facilitando la búsqueda y administración de la información. La tabla presenta datos relevantes como nombre, categoría asignada, teléfono, fecha de creación y estado, acompañado de acciones rápidas para editar o visualizar detalles del estudiante. Además, la interfaz incorpora opciones como exportación de datos y control del número de registros por página, ofreciendo una experiencia más eficiente y adaptable a las necesidades del usuario.

Por su parte, la Figura 44 presenta el formulario de *Registro de Nuevo Estudiante*, diseñado bajo un enfoque intuitivo y dividido en dos secciones: *Datos Personales* y *Datos de Matrícula*. En la primera sección se recopilan los datos básicos del estudiante, como nombres, apellidos, cédula, dirección y teléfono; mientras que la segunda sección permite seleccionar la categoría, subcategoría, ciclo y fecha de matrícula correspondiente. Cada campo está estructurado con controles claros, validaciones visuales y guías de uso, lo que facilita un registro rápido, ordenado y sin inconsistencias. El diseño moderno y limpio refuerza la experiencia del usuario, permitiéndole gestionar inscripciones de manera sencilla y eficiente.

d. Sprint 4

En este sprint se abordaron los módulos más sensibles desde el punto de vista financiero: mensualidades y pagos, así como las primeras capacidades de reporte.

- Módulo de mensualidades – Backend

Se desarrollaron funciones para generar mensualidades de forma automática para las matrículas activas, considerando el valor establecido por la academia y la fecha de vencimiento correspondiente. Las mensualidades se crean inicialmente con estado “pendiente”.

```

export const crearMensualidad = async (
  data: { id_matricula: number; mes: number; anio: number; valor: number; estado: EstadoMensualidad },
  id_usuario: number
) => {
  const { id_matricula, mes, anio, valor, estado } = data;

  const estadosValidos: EstadoMensualidad[] = ['pendiente', 'pagado', 'anulado'];
  if (!estadosValidos.includes(estado)) {
    throw { message: 'Estado inválido. Debe ser pendiente, pagado o anulado', code: 400 };
  }

  try {
    const q = `
      INSERT INTO mensualidades (id_matricula, mes, anio, valor, estado)
      VALUES ($1, $2, $3, $4, $5)
      RETURNING *
    `;
    const r = await pool.query(q, [id_matricula, mes, anio, valor, estado]);

    // Historial de estado (opcional)
    await EstadoMensualidadService.registrarEstadoMensualidad(
      r.rows[0].id_mensualidad,
      estado,
      id_usuario
    );

    await registrarAuditoria(id_usuario, 'Registro de mensualidad', 'mensualidades', r.rows[0].id_mensualidad);
    return r.rows[0];
  } catch (error: any) {
    // Único por (id_matricula, mes, anio)
    if (error?.code === '23505') {
      throw { message: 'Ya existe una mensualidad para ese mes y año', code: 409 };
    }
    throw error;
  }
};

```

Figura 45. Servicio de crear una mensualidad.

```

export async function actualizarPago(input: PagoUpdateInput, userId: number) {
  const client = await pool.connect();
  try {
    await client.query('BEGIN');

    const { rows: pagoRows } = await client.query(
      `SELECT * FROM public.pagos
      WHERE id_pago = $1
      FOR UPDATE`,
      [input.id_pago]
    );
    const pago = pagoRows[0];
    if (!pago) {
      await client.query('ROLLBACK');
      return { status: 404, error: 'Pago no encontrado' };
    }
    if (!pago.activo) {
      await client.query('ROLLBACK');
      return { status: 409, error: 'No se puede editar un pago anulado' };
    }

    const m = await lockMensualidad(client, pago.id_mensualidad);
    if (!m) {
      await client.query('ROLLBACK');
      return { status: 404, error: 'Mensualidad no encontrada' };
    }
    if (m.estado === 'anulado') {
      await client.query('ROLLBACK');
      return { status: 409, error: 'No se pueden editar pagos de una mensualidad anulada' };
    }

    const nuevoMonto = input.monto_pagado ?? pago.monto_pagado;
    const validacion = await validarSobrepago(client, m, nuevoMonto, pago.id_pago);
    if (!validacion.valido) {
      await client.query('ROLLBACK');
    }
  }
}

```

Figura 46. Servicio de actualizar pago.

```

export async function anularPago(id_pago: number, motivo: string, userId: number) {
  const client = await pool.connect();
  try {
    await client.query('BEGIN');

    const { rows: pagoRows } = await client.query(
      `SELECT * FROM public.pagos
       WHERE id_pago = $1
       FOR UPDATE`,
      [id_pago]
    );
    const pago = pagoRows[0];
    if (!pago) {
      await client.query('ROLLBACK');
      return { status: 404, error: 'Pago no encontrado' };
    }
    if (!pago.activo) {
      await client.query('ROLLBACK');
      return { status: 409, error: 'El pago ya está anulado' };
    }

    await client.query(
      `UPDATE public.pagos
       SET activo = FALSE,
           motivo_anulacion = $2,
           anulado_por = $3,
           anulado_en = NOW()
       WHERE id_pago = $1`,
      [id_pago, motivo || 'Anulado por usuario', userId]
    );

    const m = await lockMensualidad(client, pago.id_mensualidad);
    await recalcularEstado(client, m); // acepta null sin error TS

    await registrarAuditoria(userId, `Anulación de pago: ${motivo}`, 'pagos', id_pago);
  }
}

```

Figura 47. Servicio de anular pago.

- Módulo de pagos - Backend

Se implementaron las APIs necesarias para registrar pagos totales y parciales, distribuyendo el valor abonado entre las mensualidades abiertas y actualizando su estado a “pagado” cuando el saldo llega a cero. También se incluyó la posibilidad de anular pagos en casos excepcionales, registrando la acción en una tabla de auditoría (Figuras 45, 46 y 47 análogas para registrar pago, actualizar estado y anular).

```
export async function crearPago(input: PagoCreateInput, userId: number, idempotencyKey?: string) {
  const client = await pool.connect();
  try {
    await client.query('BEGIN');

    const m = await lockMensualidad(client, input.id_mensualidad);
    if (!m) {
      await client.query('ROLLBACK');
      return { status: 404, error: 'Mensualidad no encontrada' };
    }

    // ✗ typo corregido: era "óL"
    if (m.estado === 'anulado') {
      await client.query('ROLLBACK');
      return { status: 409, error: 'No se pueden registrar pagos en una mensualidad anulada' };
    }

    // Idempotencia
    if (idempotencyKey) {
      const { rows: existente } = await client.query(
        `SELECT * FROM public.pagos
         WHERE id_mensualidad = $1
            AND idempotency_key = $2
            AND activo = TRUE
         LIMIT 1`,
        [input.id_mensualidad, idempotencyKey]
      );
      if (existente[0]) {
        await client.query('COMMIT');
        return { status: 200, data: existente[0], message: 'Pago ya registrado (idempotente)' };
      }
    }

    await client.query('COMMIT');
    return { status: 200, data: { id: input.id_mensualidad, estado: m.estado }, message: 'Pago registrado exitosamente' };
  } catch (error) {
    await client.query('ROLLBACK');
    return { status: 500, error: 'Error al registrar pago' };
  }
}
```

Figura 48. Servicio de crear pago.

- Módulo de pagos y estado de cuenta-Frontend

En Flutter se construyeron pantallas para que el administrador pueda buscar un estudiante, visualizar sus mensualidades, registrar pagos y consultar el estado de cuenta. Se incluyeron alertas de confirmación antes de operaciones críticas y mensajes claros de éxito o error.

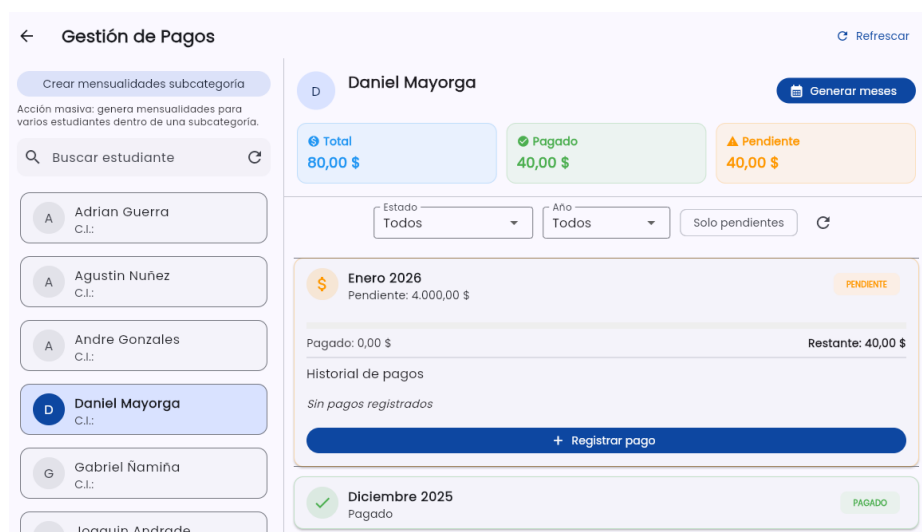


Figura 49. Pantalla de pagos.

- Implementación de seguridad aplicativa – Backend y Frontend

Backend:

Durante este sprint se integró el sistema completo de seguridad en el backend. Se configuró el uso de tokens JWT con expiración, incorporando la verificación automática en cada solicitud mediante middleware de autenticación. Se implementó el cifrado de contraseñas con bcrypt y se establecieron reglas de validación de contraseñas fuertes en el endpoint de registro de usuarios.

Se configuró CORS de manera restrictiva, permitiendo únicamente solicitudes desde los dominios autorizados (<https://portoambato.ec> y <http://localhost:3000> para desarrollo), bloqueando cualquier otro origen.

Se implementaron **validaciones de entrada**, utilizando la librería Zod, que valida la estructura y tipos de datos de cada petición antes de procesarla, evitando errores y posibles ataques de inyección.

Frontend:

En la aplicación Flutter se implementaron los siguientes controles de seguridad:

Control de intentos fallidos: Se incorporó un contador que limita intentos de inicio de sesión. Tras superar este límite, la cuenta queda bloqueada temporalmente durante 15 minutos, mostrando un mensaje claro al usuario sobre el tiempo restante.

Temporizadores en operaciones sensibles: Para el envío de correos de recuperación de contraseña y códigos de verificación, se implementó un temporizador de 60 segundos que desactiva el botón de envío durante ese periodo, evitando el spam de solicitudes.

Autenticación con Google OAuth 2.0: Se completó la integración del inicio de sesión mediante Google, utilizando el paquete `google_sign_in`. El flujo verifica el token de Google en el backend y crea o autentica al usuario de manera segura sin exponer credenciales.

Validación continua de sesión: Se implementó un interceptor HTTP que verifica en cada respuesta del servidor si el token sigue siendo válido. Ante respuestas 401 (No autorizado) o tokens expirados, se cierra automáticamente la sesión y se redirige al usuario a la pantalla de login.

Validaciones de formularios: Todos los campos de entrada incorporan validación en tiempo real (formato de correo electrónico, longitud mínima de contraseñas, caracteres permitidos en nombres), mostrando mensajes de error claros antes de permitir el envío de datos.

- Pruebas de seguridad

Al finalizar el sprint se ejecutaron pruebas específicas de seguridad:

- **Pruebas de autenticación:** Verificación de rechazo de tokens inválidos, expirados o manipulados.
- **Pruebas de autorización:** Confirmación de que cada rol solo puede acceder a sus endpoints permitidos.
- **Pruebas de límite de intentos:** Validación del bloqueo temporal tras intentos fallidos.
- **Pruebas de CORS:** Verificación de rechazo de solicitudes desde orígenes no autorizados.

Como resultado de este sprint, el sistema quedó protegido mediante un conjunto integral de medidas de seguridad que cubren autenticación, autorización, trazabilidad, validación de datos y protección contra ataques comunes, cumpliendo con los requisitos no funcionales de seguridad establecidos en la sección 3.5.1.

e. *Sprint 5*

En el último sprint se enfocó el trabajo en la estabilización, pruebas y preparación de la implantación:

- Pruebas y optimización

Se llevaron a cabo pruebas de rendimiento en las operaciones más utilizadas (listados, generación de mensualidades, registro de pagos). Se ajustaron consultas SQL, índices y validaciones para mejorar tiempos de respuesta. También se verificó el correcto manejo de sesiones expiradas y tokens inválidos.

- Integración y despliegue

Se integraron todos los módulos en una versión unificada del sistema y se configuró un entorno de despliegue en la plataforma Railway, donde se alojó tanto la base de datos PostgreSQL como el backend desarrollado en Node.js. Esta infraestructura en la nube permitió gestionar los servicios de forma centralizada, automatizar la puesta en línea del proyecto y garantizar su disponibilidad. Además, se documentaron los procedimientos necesarios para el despliegue, la actualización y la administración del sistema en producción, asegurando que el proceso pueda replicarse de manera consistente.

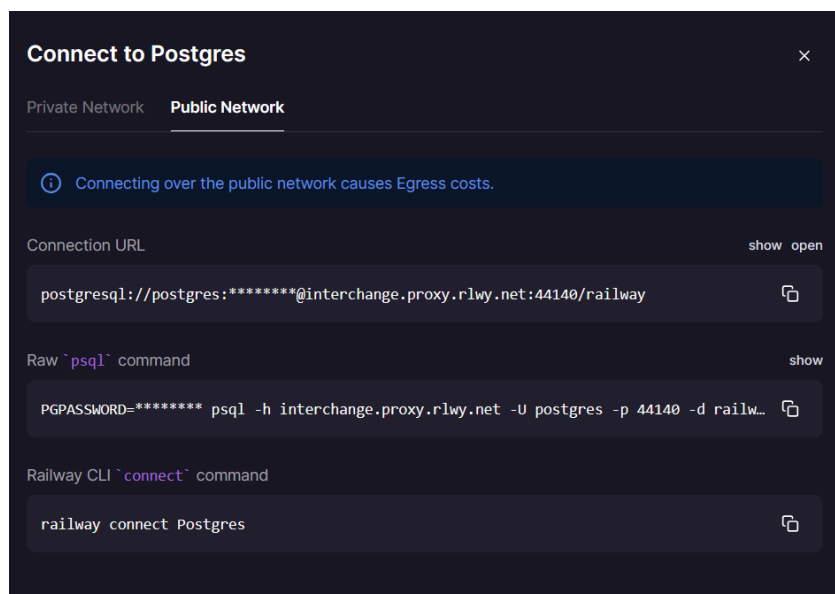


Figura 50. Datos de conexión para la base de datos.

En la Figura 50 se presenta la interfaz de Railway utilizada para obtener los datos de conexión a la base de datos PostgreSQL desplegada en la plataforma. Desde esta sección se genera automáticamente la URL de conexión. Esta información permite conectarse de forma segura al motor de base de datos alojado en la nube, gestionar tablas, ejecutar migraciones y verificar el estado del servicio. Gracias a esta infraestructura, el sistema puede operar con una base de datos accesible desde cualquier entorno autorizado, garantizando disponibilidad y facilitando el proceso de despliegue en producción.

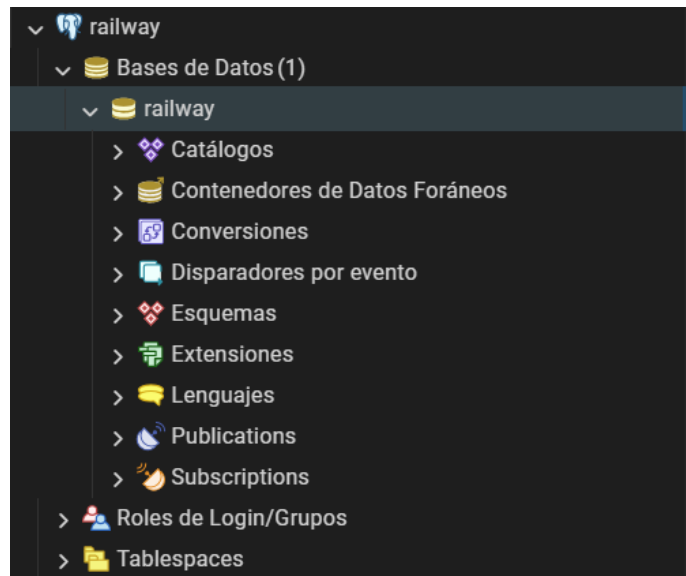


Figura 51. Conexión al servidor de Railway.

En la Figura 51 se muestra la conexión realizada desde PgAdmin hacia la instancia de PostgreSQL alojada en Railway. Esta conexión permitió gestionar la base de datos desde el entorno local, facilitando la importación del archivo .sql con toda la estructura y datos iniciales del sistema. A través de esta interfaz fue posible visualizar los catálogos, esquemas, extensiones y demás componentes internos de la base de datos, asegurando que la restauración se ejecutara correctamente. Este procedimiento resultó fundamental para cargar la base de datos completa y dejarla operativa en el entorno de producción de Railway.

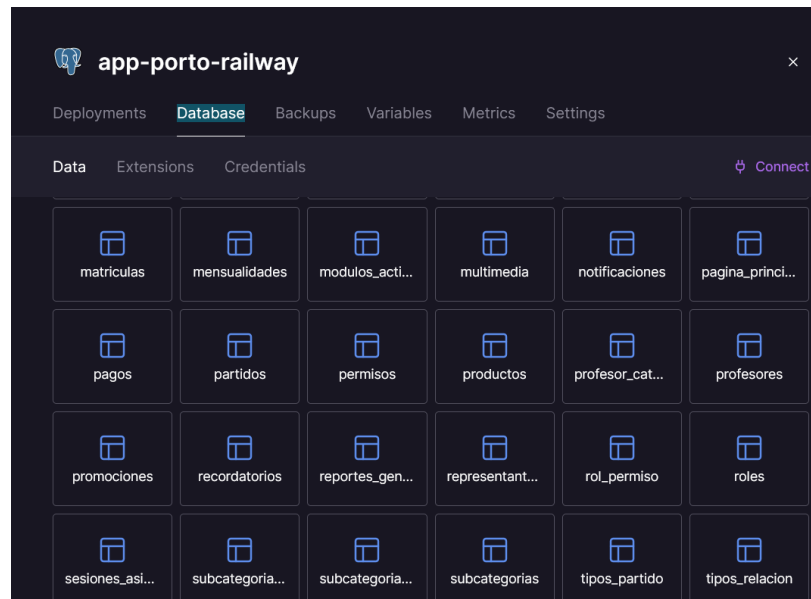


Figura 52. Tablas en Railway.

En la Figura 52 se visualizan todas las tablas que fueron correctamente cargadas y desplegadas en la instancia de PostgreSQL dentro de Railway. Este panel permite comprobar que la estructura completa de la base de datos, incluyendo módulos como matrículas, mensualidades, subcategorías, pagos, profesores, roles, reportes, entre otros, fue importada y se encuentra operativa en el entorno de producción. Gracias a esta visualización, se pudo verificar que cada tabla y relación quedó registrada de manera adecuada en el servidor, garantizando así que el sistema disponga de todos los componentes necesarios para su funcionamiento integral.

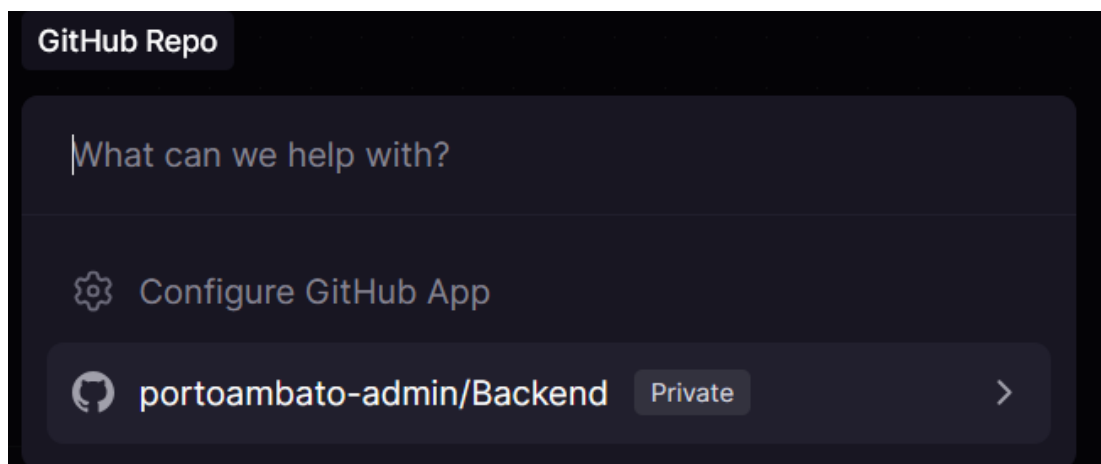


Figura 53. Creación del backend en Railway.

En la Figura 53 se observa la vinculación del proyecto con el repositorio privado de GitHub denominado portoambato-admin/Backend. A través de esta integración, Railway puede obtener automáticamente el código fuente del backend, permitiendo desplegar nuevas versiones del sistema cada vez que se realice un commit o actualización en el repositorio. Esta conexión garantiza un flujo de desarrollo continuo, ordenado y versionado, facilitando la automatización del despliegue, la sincronización del código y el mantenimiento del entorno de producción. Gracias a ello, el sistema puede mantenerse siempre actualizado sin necesidad de realizar cargas manuales del backend.

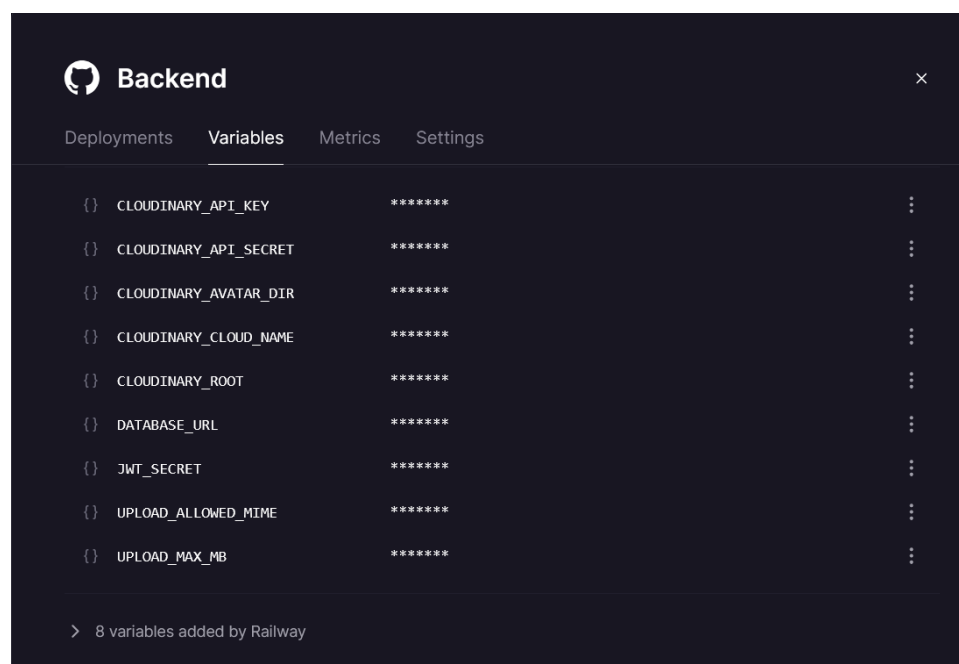


Figura 54. Variables de configuración del backend.

En la Figura 54 se presentan las variables de entorno configuradas en el panel de Railway para el despliegue del backend. Estas variables permiten gestionar de forma segura información sensible utilizada por la aplicación, como las credenciales de Cloudinary, el secreto JWT para la autenticación y los parámetros relacionados con la carga de archivos. Entre ellas destaca **DATABASE_URL**, que contiene la cadena de conexión hacia la base de datos PostgreSQL alojada en Railway, permitiendo que el backend se comunique correctamente con el motor de datos en producción. La correcta configuración de estas variables garantiza un despliegue estable, seguro y plenamente funcional del sistema en el entorno remoto.

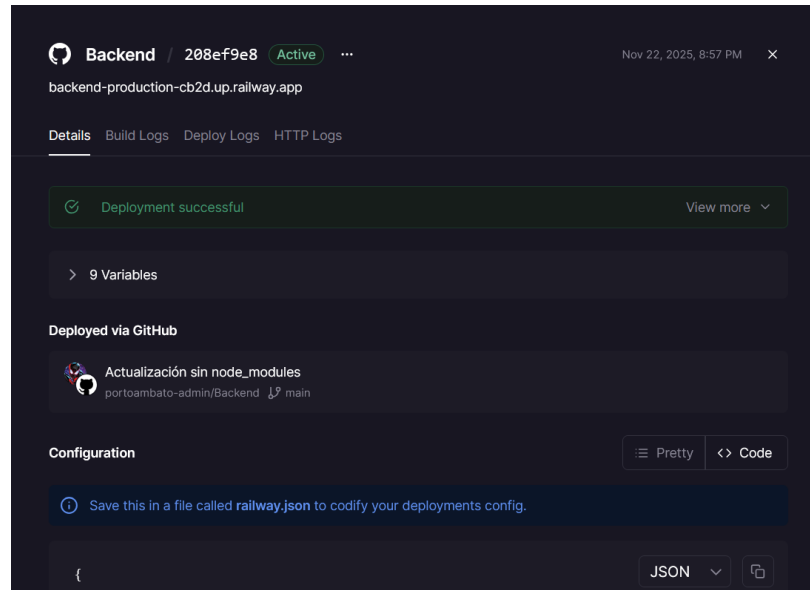


Figura 55. Backend desplegado.

En la Figura 55 se observa el panel de despliegue del backend dentro de Railway, donde se confirma que la última versión del sistema fue implementada exitosamente a partir del commit más reciente proveniente del repositorio privado en GitHub. Esta vista permite visualizar el estado del servicio, las variables cargadas, los registros de compilación y ejecución, y la configuración generada automáticamente para el entorno de producción. Gracias a este mecanismo de integración continua, cada actualización enviada al repositorio se despliega de manera ordenada y automatizada, garantizando que el sistema se mantenga siempre actualizado y operativo en el servidor remoto.

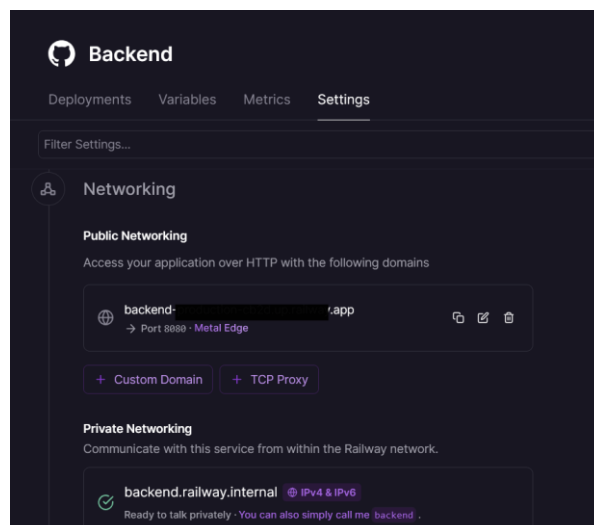


Figura 56. Configuración de enlace para poder usar el backend.

En la Figura 56 se muestra la sección de configuración de red del servicio backend dentro de Railway, donde se gestionan los dominios públicos y privados asignados a la aplicación. Desde este apartado es posible visualizar la URL pública generada por la plataforma la cual permite acceder al backend desde internet y habilitar funciones adicionales como la asignación de un dominio personalizado, el uso de TCP Proxy o la interacción a través del dominio interno provisto por Railway para comunicaciones privadas entre servicios. Esta configuración es esencial para que el backend pueda ser consumido por el frontend, aplicaciones móviles u otros clientes externos, ya que define la ruta oficial mediante la cual el sistema expone sus endpoints en producción.

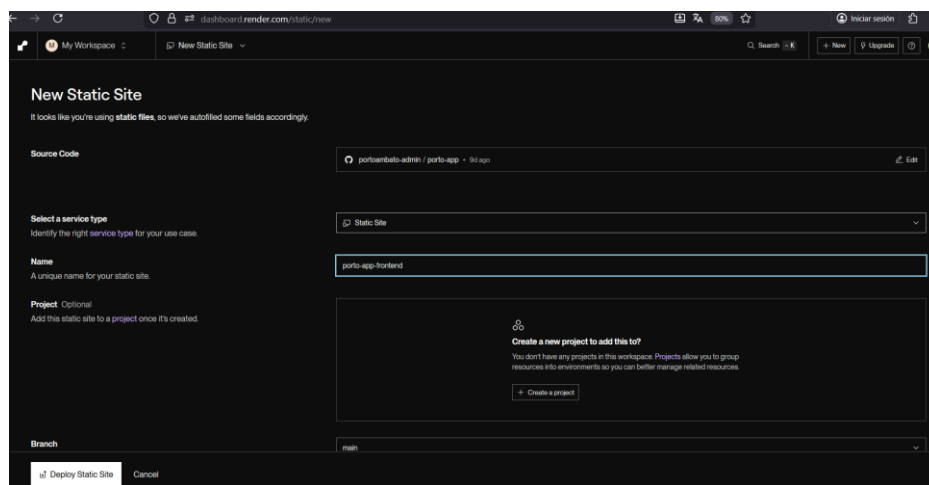


Figura 57. Creación del sitio en Render.

Render es una plataforma moderna de despliegue en la nube que permite publicar aplicaciones y sitios web de forma sencilla, automatizada y sin configuraciones complejas. Su uso resulta especialmente práctico para proyectos frontend que requieren un hosting rápido y estable.

En la Figura 57 se observa el proceso de creación de un nuevo sitio estático en Render, donde se selecciona el repositorio correspondiente del frontend y se define el nombre del servicio antes del despliegue. Este formulario permite configurar desde el tipo de servicio hasta la rama del repositorio que será usada, dejando todo listo para que la plataforma procese los archivos, compile el proyecto si es necesario y finalmente publique el sitio. Con solo presionar “Deploy Static Site”, Render inicia automáticamente el proceso de construcción y puesta en producción, facilitando un flujo de trabajo continuo y adaptable para el desarrollo del sistema.

The image shows a dark-themed configuration interface for a service. On the left, there are five sections with descriptive text and links: 'Included Paths', 'Ignored Paths', 'Build Command', 'Publish Directory', 'Auto-Deploy', and 'Deploy Hook'. On the right, there are corresponding input fields and controls. 'Included Paths' and 'Ignored Paths' each have a '+ Add' button. 'Build Command' has a text input with 'app_porto/ \$' and an 'Edit' link. 'Publish Directory' has a text input with 'app_porto/ build/web' and an 'Edit' link. 'Auto-Deploy' has a dropdown menu set to 'Off' and an 'Edit' link. 'Deploy Hook' has a long, empty text input field, an eye icon for toggling visibility, a copy icon, and a 'Regenerate hook' button.

Include or ignore specific paths in your repo when determining whether to trigger an auto-deploy. Paths are relative to your repo's root directory. [Learn more.](#)

Included Paths
Changes that match these paths will trigger a new build.

+ Add Included Path

Ignored Paths
Changes that match these paths will not trigger a new build.

+ Add Ignored Path

Build Command
Render runs this command to build your app before each deploy.

app_porto/ \$ [Edit](#)

Publish Directory
The relative path of the directory containing built assets to publish. Examples: `./build`, `dist` and `frontend/build`.

app_porto/ build/web [Edit](#)

Auto-Deploy
By default, Render automatically deploys your service whenever you update its code or configuration. Disable to handle deploys manually. [Learn more.](#)

Off [Edit](#)

Deploy Hook
Your private URL to trigger a deploy for this server. Remember to keep this a secret.

[Regenerate hook](#)

Figura 58. Parámetros iniciales.

En la Figura 58 se observan los parámetros de configuración esenciales para el despliegue del frontend dentro de la plataforma Render. En este apartado se definen las rutas y comandos necesarios para que la aplicación pueda construirse correctamente y ser publicada en el servidor. Uno de los elementos más importantes es el campo Build Command, donde se especifica la instrucción encargada de compilar el proyecto antes de cada despliegue. De igual manera, en Publish Directory se indica la carpeta resultada del proceso de construcción en este caso build/web que Render utilizará como origen de los archivos estáticos que se servirán públicamente.

Además, se muestra el control Auto-Deploy, que permite habilitar o deshabilitar los despliegues automáticos cada vez que se registre un nuevo commit en el repositorio vinculado. Esta opción es útil para mantener un flujo de integración continua, aunque también permite administrar los despliegues manualmente cuando se requiere un mayor control. Finalmente, se incluye el Deploy Hook, una URL privada que permite disparar un despliegue de manera remota, útil para automatizaciones o integraciones externas. Todo este conjunto de configuraciones garantiza que el proceso de publicación del frontend sea consistente, repetible y totalmente alineado con el ciclo de desarrollo del proyecto.

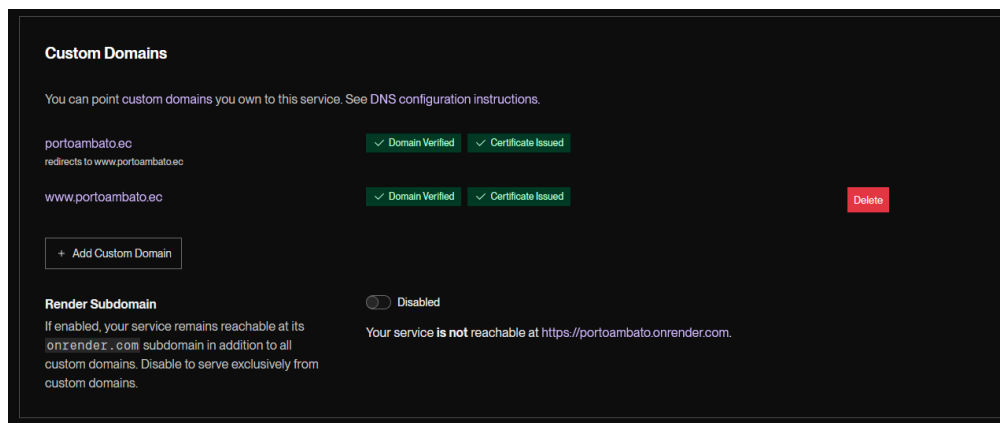


Figura 59, Configuración de dominio.

En la Figura 59 se presenta la configuración de dominios personalizados dentro de Render, un paso fundamental para publicar el sistema con un dominio propio y profesional. En este caso, se agregaron los dominios `portoambato.ec` y `www.portoambato.ec`, los cuales fueron verificados correctamente por la plataforma, tal como lo indican los estados `Domain Verified` y `Certificate Issued`. Esto significa que Render pudo validar la propiedad del dominio y generar automáticamente los certificados SSL necesarios para habilitar una conexión segura mediante HTTPS.

Además, se observa que la opción `Render Subdomain` se encuentra deshabilitada, lo que implica que el proyecto deja de ser accesible desde el subdominio predeterminado de Render y pasa a operar exclusivamente bajo los dominios personalizados configurados. Esta configuración permite que la aplicación se presente con una identidad propia, profesional y acorde a los requerimientos de despliegue del sistema de la Academia de Fútbol Porto Ambato.

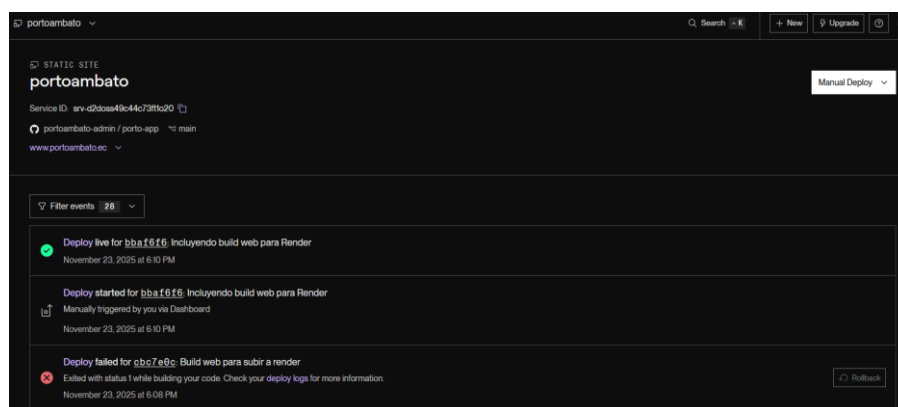


Figura 60. Sitio desplegado.

En la Figura 60 se observa el panel principal de despliegues de Render correspondiente al frontend del sistema, donde se registran cada uno de los intentos de publicación realizados. En este espacio se muestran tanto los despliegues exitosos como aquellos que fallaron, permitiendo identificar rápidamente el estado actual del sitio y los cambios aplicados en cada actualización. Además, Render registra de forma automática el commit asociado a cada despliegue, lo que facilita la trazabilidad entre las versiones del código y el comportamiento del sistema en producción.

Esta vista también permite ejecutar despliegues manuales, revisar logs detallados en caso de errores y realizar rollbacks cuando es necesario volver a una versión anterior estable. Gracias a este panel, el desarrollador tiene un control claro del historial de despliegues y del funcionamiento del servicio en tiempo real, garantizando un proceso de publicación ordenado y totalmente documentado dentro del flujo de trabajo del proyecto.

- Retroalimentación y mejora continua

Se llevaron a cabo pruebas constantes en cada uno de los entornos desplegados. Desde la base de datos alojada en Railway se monitorearon los registros y eventos mediante los logs del servidor, lo que permitió validar consultas, detectar inconsistencias y optimizar tiempos de respuesta. De manera paralela, el backend en Railway fue evaluado a través de sus propios logs y panel de actividad, verificando el correcto funcionamiento de las rutas, el uso de variables de entorno y el rendimiento del servicio en producción. Finalmente, el frontend desplegado en Render se sometió a múltiples ciclos de revisión, observando los resultados de los despliegues, errores de compilación y comportamientos en tiempo real. Este proceso continuo de monitoreo, ajuste y verificación garantizó una evolución progresiva del sistema, fortaleciendo su estabilidad, seguridad y confiabilidad en cada iteración.

3.6 Eficiencia en la gestión de cobros en la Academia

Para valorar el impacto del sistema multiplataforma desarrollado en la Academia Oficial de Fútbol Porto Ambato, se analizó la eficiencia en la gestión de cobros antes y después de la implantación de la aplicación.

Para ello, se tomaron como referencia tareas clave del proceso de cobro (registro de matrículas, generación de mensualidades, registro de pagos, consulta de estado de cuenta y elaboración de reportes). Los tiempos fueron estimados con base en la información proporcionada por el administrador y el personal docente responsable de estas actividades.

En la Tabla 29 se muestran los tiempos promedio dedicados a cada actividad, comparando el trabajo manual previo con el uso del sistema multiplataforma.

Tabla 29. Período de ejecución de actividades en la gestión de cobros.

Aspecto	Antes de la implantación del sistema multiplataforma	Después de la implantación del sistema multiplataforma
Registro de matrícula y generación de la primera mensualidad por estudiante	25 minutos	3 minutos
Generación de mensualidades para todos los estudiantes de una categoría	60 minutos	5 minutos
Registro de pago mensual y emisión de comprobante	15 minutos	1 minuto
Búsqueda del estado de cuenta de un estudiante	20 minutos	10 segundos
Elaboración de reporte mensual de morosidad	45 minutos	4 minutos

A partir de estos datos se calculó el porcentaje de mejora en cada aspecto, utilizando la siguiente expresión:

$$MP = \left(\frac{TM - TA}{TM} \right) * 100$$

Donde:

MP = Mejora porcentual

TM = Tiempo medio del proceso antes del sistema (gestión manual).

TP = Tiempo medio del proceso después del sistema (gestión automatizada).

Para los cálculos, los tiempos expresados en segundos se convirtieron a minutos.

Tabla 30. Resultados de mejora porcentual en la gestión de cobros.

Aspecto	TM(min)	TA(min)	MP
Registro de matrícula y generación de primera mensualidad	25,00	3,00	88,00 %
Generación de mensualidades por subcategoría	60,00	5,00	91,67 %
Registro de pago y emisión de comprobante	15,00	1,00	93,33 %
Búsqueda de estado de cuenta de un estudiante	20,00	0,17	99,17 %
Elaboración de reporte mensual de morosidad	45,00	4,00	91,11 %

Con la Tabla 30 y los resultados se observa una reducción muy significativa de los tiempos de atención en todas las actividades analizadas. Las tareas que antes requerían varios minutos de búsqueda en cuadernos, hojas sueltas o archivos de cálculo, ahora se ejecutan en cuestión de segundos o pocos minutos desde la aplicación web y móvil.

En términos prácticos, esto se traduce en:

- Menor carga operativa para el administrador y los profesores.
- Mayor rapidez para responder a los representantes sobre el estado de pagos.
- Generación casi inmediata de reportes de morosidad y control de ingresos.

En conjunto, la automatización de la gestión de cobros mediante el sistema multiplataforma de la Academia Oficial de Fútbol Porto Ambato incrementa de forma notable la eficiencia del proceso y aporta una base más sólida para la toma de decisiones administrativas.

CAPÍTULO IV. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- El análisis del proceso actual de cobros en la Academia Porto Ambato determinó que el uso de herramientas manuales y dispersas como cuadernos, Excel y WhatsApp, que genera errores de registro, pérdida de información, cálculos incorrectos y baja transparencia para los representantes. Se evidencia plenamente en el análisis del proceso manual descrito en el Capítulo III, donde se identifican fallas en la organización de datos, validación y trazabilidad de pagos.
- Para el desarrollo de la propuesta, se utilizó Flutter, para la capa de presentación, Node.js con Express para la lógica de negocio y PostgreSQL para la base de datos resultó ser técnicamente viable y eficiente. Esta combinación permitió desarrollar una solución multiplataforma escalable que unifica la experiencia web y móvil con una sola base de código, cumpliendo con los requisitos de rendimiento y consistencia visual entre dispositivos.
- La implementación de la metodología híbrida Scrumban facilitó la organización y el control del desarrollo del sistema en un contexto de recursos limitados. Se concluye que la división del trabajo en Sprints permitió entregar módulos funcionales de manera progresiva, asegurando que funcionalidades críticas como la generación de mensualidades y el registro de pagos operen correctamente antes del cierre del proyecto.
- El sistema desarrollado permite tener un mejor flujo de cobros, transformando un proceso basado en la memoria y notas físicas en un sistema con reglas de negocio validadas y notificaciones automáticas. La comparación entre el proceso manual y el automatizado demuestra que la nueva plataforma garantiza la integridad de los datos, permite la consulta de historial en tiempo real y reduce significativamente la dependencia del personal administrativo para conocer el estado de deuda de un estudiante de la academia.

4.2 Recomendaciones

- Se aconseja ejecutar un plan de capacitación dirigido a los padres de familia ya que las encuestas mostraron un alto índice de "neutralidad" respecto a la adopción tecnológica, es vital demostrar prácticamente los beneficios del sistema para asegurar una transición exitosa y reducir la brecha de aceptación inicial.
- Aunque el sistema utiliza plataformas en la nube como Railway y Render, se puede establecer una política institucional en el futuro de copias de seguridad (backups) periódicas de la base de datos PostgreSQL y auditorías de los registros de actividad. Esto garantizará la sostenibilidad operativa de la academia a largo plazo y protegerá la información sensible de estudiantes y pagos ante posibles fallos externos.
- Actualmente, el sistema registra pagos realizados por transferencia o efectivo de forma manual o validada por el administrador. Se delega, como una siguiente fase evolutiva del software, integrar una pasarela de pagos digital (gateways) directamente en la aplicación. Esto automatizaría completamente el ciclo, eliminando la necesidad de validación manual de comprobantes y reduciendo aún más la carga operativa administrativa.
- Aprovechando la flexibilidad de Flutter, se deben recolectar comentarios sobre interfaz de los módulos de “Gestión de Pagos” y “Consulta de estado de cuenta” por lo que se sugiere realizar evaluaciones periódicas de usabilidad durante los primeros meses de despliegue en producción para realizar ajustes estéticos o funcionales que faciliten el uso a usuarios con menor destreza digital.

REFERENCIAS BIBLIOGRÁFICAS

- [1] F. C. Bull *et al.*, "World Health Organization 2020 guidelines on physical activity and sedentary behaviour," *Br J Sports Med*, vol. 54, p. 20, 1451, doi: 10.1136/bjsports-2020-102955.
- [2] K. P. F. Tacan, "La actividad física aeróbica y su relación con la depresión en los adolescentes: Revisión Sistemática," *MENTOR revista de investigación educativa y deportiva*, vol. 4, no. 10, pp. 698–714, Jan. 2025, doi: 10.56200/mried.v4i10.9123.
- [3] D. Carranza-Bautista, A. J. G. Hincapié, C. F. R. Núñez, and M. C. S. Rodríguez, "Sports and physical activity management challenges and issues in Latin America: Strategic diagnosis through the P + SEPCES = R Tools and the SWOT matrix," *Retos*, vol. 63, pp. 291–305, Feb. 2025, doi: 10.47197/RETOS.V63.109697.
- [4] Leonardo José Paredes Reyes, "APLICACIÓN MÓVIL PARA LA GESTIÓN DEPORTIVA," UNIVERSIDAD TÉCNICA DE AMBATO, Ambato, 2021.
- [5] N. Kuzmin, K. Ignatiev, and D. Grafov, "Experience of Developing a Mobile Application Using Flutter," *Lecture Notes in Electrical Engineering*, vol. 621, pp. 571–575, 2020, doi: 10.1007/978-981-15-1465-4_56.
- [6] S. Boukhary and E. Colmenares, "A clean approach to flutter development through the flutter clean architecture package," *Proceedings - 6th Annual Conference on Computational Science and Computational Intelligence, CSCI 2019*, pp. 1115–1120, Dec. 2019, doi: 10.1109/CSCI49370.2019.00211.
- [7] C. Özkurt, "MANAGING CORPORATE CLEANING OPERATIONS WITH A FLUTTER-DEVELOPED MOBILE APP: MAXIMIZING EFFICIENCY WITH THE CLEANER APP," *International Journal of Engineering and Innovative Research*, vol. 6, no. 3, pp. 135–150, Oct. 2024, doi: 10.47933/IJEIR.1508477.
- [8] C. Cariño, "360Player, la plataforma usada por Barça Academy y Rafa Nadal Academy, acelera su expansión - AS.com." Accessed: May 17, 2025. [Online]. Available: <https://as.com/futbol/360player-la-app-de-referencia-de-barca-academy-y-rafa-nadal-academy-acelera-su-expansion-n/>
- [9] E. Ramos Méndez, G. Arceo Moheno, and M. A. Almeida Aguilar, "Las TIC's en la innovación de los procesos organizacionales de las pequeñas y medianas empresas," *Vinculatégica*, vol. 7, no. 1, Dec. 2021, doi: 10.29105/VTGA7.1-144.
- [10] R. Isaac and B. Rivera, "ANÁLISIS COMPARATIVO ENTRE FRAMEWORKS, PARA EL DESARROLLO DE APLICACIONES MÓVILES MULTIPLATAFORMAS," UNIVERSIDAD TÉCNICA DE BABAHOYO, Babahoyo, 2021. Accessed: May 17, 2025. [Online]. Available: <https://dspace.utb.edu.ec/bitstream/handle/49000/9473/E-UTB-FAFI-SIST-000153.pdf?sequence=1&isAllowed=y>
- [11] Rodríguez Mayón and Steward Daniel, "Desarrollo de una aplicación web para la evaluación de calidad del software basado en el modelo de calidad iso 9126,"

- Universidad Técnica de Machala, Machala, 2020. Accessed: May 17, 2025. [Online]. Available: <http://repositorio.utmachala.edu.ec/handle/48000/16048>
- [12] F. Rini and A. Yulio Pernanda, "INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH AND ANALYSIS New Lms Mobile Framework Based on Multiplatform: A Literature Review of Mobile Lms Theory, Design and Implementation", doi: 10.47191/ijmra/v6-i6-54.
 - [13] N. Torres Flores, "OPTIMIZACIÓN DE LA GESTIÓN DE COBRANZAS PARA MEJORAR LA LIQUIDEZ EN LA EMPRESA GRUPO ASEGA SAC," Universidad San Ignacio de Loyola, Lima, 2024. Accessed: Oct. 14, 2025. [Online]. Available: www.gob.pe
 - [14] "(PDF) The Adoption and Use of Distributed System Architecture on Cross-Multiple Platforms." Accessed: Dec. 03, 2025. [Online]. Available: https://www.researchgate.net/publication/391732257_The_Adoption_and_Use_of_Distributed_System_Architecture_on_Cross-Multiple_Platforms
 - [15] G. Jošt and V. Taneski, "State-of-the-Art Cross-Platform Mobile Application Development Frameworks: A Comparative Study of Market and Developer Trends," *Informatics 2025, Vol. 12, Page 45*, vol. 12, no. 2, p. 45, Apr. 2025, doi: 10.3390/INFORMATICS12020045.
 - [16] "Cross Platform Mobile Development Frameworks to Use in 2025." Accessed: Dec. 03, 2025. [Online]. Available: https://www.wearedevelopers.com/en/magazine/404/cross-platform-mobile-development-frameworks?utm_source=chatgpt.com
 - [17] Arjun Santhosh, Anu Tiji, Anuraj T R, Naithan Thomas, Aswathy Anoop, and Tintu Varghese, "Cross-Platform Innovation: The Rise and Impact of Flutter in Modern App Development," *International Research Journal on Advanced Engineering and Management (IRJAEM)*, vol. 2, no. 12, pp. 3560–3569, Dec. 2024, doi: 10.47392/IRJAEM.2024.0525.
 - [18] "Flutter architectural overview." Accessed: Dec. 03, 2025. [Online]. Available: https://docs.flutter.dev/resources/architectural-overview?utm_source=chatgpt.com
 - [19] "What's new in Flutter 3.27. Elevating App Development with... | by Kevin Chisholm | Flutter." Accessed: Dec. 03, 2025. [Online]. Available: <https://medium.com/flutter/whats-new-in-flutter-3-27-28341129570c>
 - [20] "What's New with Flutter 3.29 and Real Use Cases | by Selin Namak | Stackademic." Accessed: Dec. 03, 2025. [Online]. Available: <https://medium.com/stackademic/whats-new-with-flutter-3-29-and-real-use-cases-5a31f302f0b0>
 - [21] "Flutter 3.35 — What's New and Why It Matters | by Pranav Dave | Medium." Accessed: Dec. 03, 2025. [Online]. Available: <https://medium.com/@pranavdave.code/flutter-3-35-whats-new-and-why-it-matters-90e969483ed9>

- [22] "What's new in Flutter 3.38. Write less, see more, build faster | by Kevin Chisholm | Nov, 2025 | Flutter." Accessed: Dec. 03, 2025. [Online]. Available: <https://medium.com/flutter/whats-new-in-flutter-3-38-3f7b258f7228>
- [23] "Apps Built with Flutter - 12 Must-Know Examples in 2025 (December 2025)." Accessed: Dec. 03, 2025. [Online]. Available: https://litslink.com/blog/apps-built-with-flutter?utm_source=chatgpt.com
- [24] "Top 15 Mobile Applications Built with Flutter Framework - DesignersX." Accessed: Dec. 03, 2025. [Online]. Available: https://www.designersx.us/top-12-mobile-apps-build-flutter-framework/?utm_source=chatgpt.com
- [25] "Amazing Apps Built with Flutter." Accessed: Dec. 03, 2025. [Online]. Available: https://tms-outsource.com/blog/posts/apps-built-with-flutter/?utm_source=chatgpt.com
- [26] "Top 20 Apps Built with Flutter Framework - TopDevelopers.co." Accessed: Dec. 03, 2025. [Online]. Available: https://www.topdevelopers.co/blog/top-apps-built-using-flutter/?utm_source=chatgpt.com
- [27] M. Wicha and B. Pańczyk, "Performance analysis of REST API technologies using Spring and Express.js examples," *Journal of Computer Sciences Institute*, vol. 29, pp. 352–359, Dec. 2023, doi: 10.35784/JCSI.3796.

ANEXOS

Anexo A. Manual de usuario.

En la Figura A1 se muestra la portada del manual de usuario.

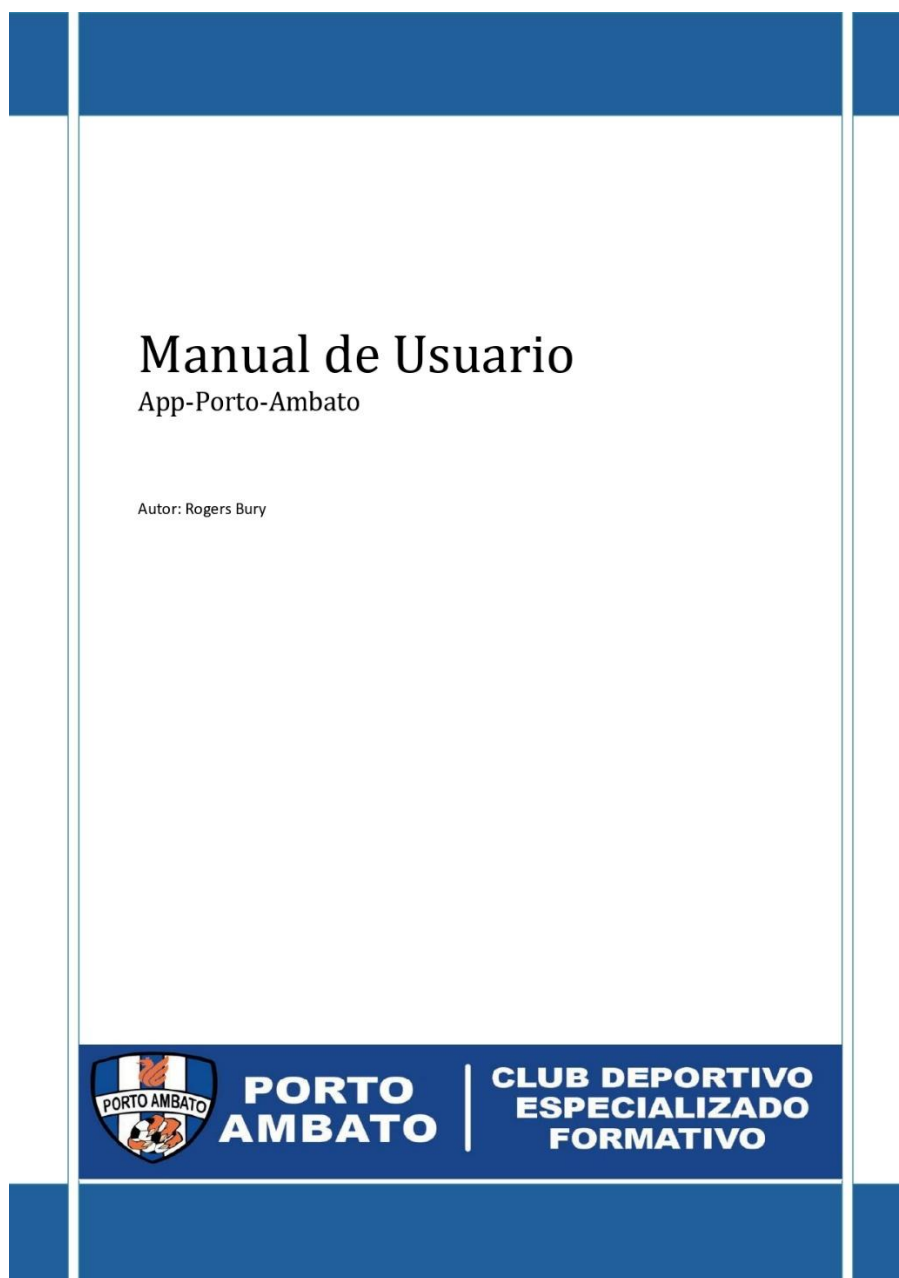


Figura A1. Portada.

En la Figura A2 se muestra la descripción breve.



Descripción

El manual describe la interfaz gráfica, los componentes principales y las funciones más importantes de la plataforma, con el objetivo de garantizar una experiencia satisfactoria y eficiente durante su uso. Incluye instrucciones paso a paso para realizar las operaciones más frecuentes, explicaciones claras de botones, menús e iconos, así como ejemplos prácticos de navegación en cada sección. Además, ofrece recomendaciones de uso, buenas prácticas y una guía básica para la resolución de problemas comunes, de manera que el usuario pueda aprovechar al máximo todas las funcionalidades disponibles.

Adicionalmente, la documentación y la descripción de la arquitectura se elaboraron tomando como referencia los lineamientos de ISO/IEC/IEEE 42010 y el modelo de calidad de ISO/IEC 25010, priorizando atributos relevantes para el contexto de la academia como seguridad, usabilidad y mantenibilidad. De igual forma, se incorporaron criterios básicos y verificables de accesibilidad basados en WCAG 2.2, aplicados de manera selectiva, sin declarar conformidad total.

Figura A2. Descripción.

En la Figura A3 se muestra la tabla de contenido.

	Manual de Usuario App-Porto-Ambato
Tabla de contenido	
Introducción	4
Contenido	4
Introducción al Sistema	4
<i>Ingreso al Sistema.</i>	<i>5</i>
<i>Componentes de Interfaz</i>	<i>6</i>
<i>Notificaciones.....</i>	<i>7</i>
<i>Menú Lateral</i>	<i>7</i>
<i>Dashboard.....</i>	<i>9</i>
<i>Perfil</i>	<i>9</i>
Uso del Sistema	11
<i>Módulo de Usuarios</i>	<i>11</i>
<i>Módulo de Profesores</i>	<i>12</i>
<i>Módulo de Roles.....</i>	<i>12</i>
<i>Módulo Categorías.....</i>	<i>13</i>
<i>Módulo de Subcategorías.....</i>	<i>14</i>
<i>Módulo de Estudiantes.....</i>	<i>15</i>
<i>Módulo de asistencia</i>	<i>18</i>
<i>Módulo de pagos.....</i>	<i>19</i>
<i>Registro masivo de mensualidades.....</i>	<i>20</i>
<i>Módulo de Reportes.....</i>	<i>20</i>
<i>Configuración General del Sistema</i>	<i>21</i>
Anexo	23
<i>Guía rápida de Botones.....</i>	<i>23</i>
Rogers Bury	Página 3 de 23

Figura A3. Tabla de contenido.

En la Figura A4 se muestra la introducción del manual, contenido e introducción al sistema.

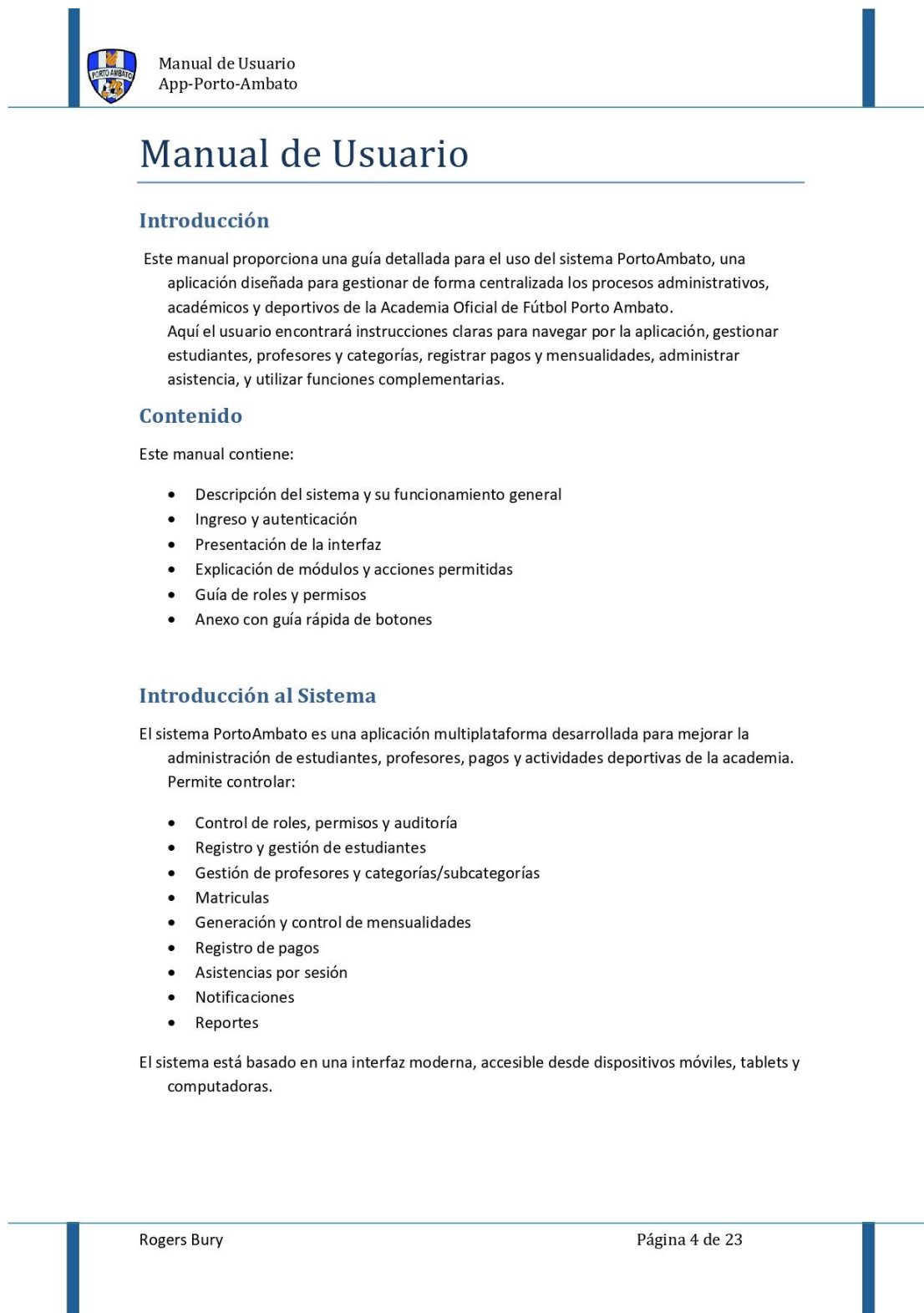


Figura A4. Introducción y contenido del sistema.

En la Figura A5 se muestra el login y el módulo de olvidaste tu contraseña.



Ingreso al Sistema.

Figura 1. Pantalla de acceso al sistema.

Figura 2. Proceso de recuperación de contraseña.

En la Figura A6 Pasos para usar el módulo y componentes de interfaz.



Para ingresar correctamente al sistema, siga los siguientes pasos:

Paso 1. Abra la aplicación

Desde su dispositivo móvil o navegador web, acceda al sistema PortoAmbato.

Paso 2. Ingrese sus credenciales

En la pantalla de inicio:

Escriba su **correo electrónico**.

Ingrese su **contraseña**.

Pulse el botón **INICIAR SESIÓN**.

Paso 3. Ingreso con Google

Si su cuenta está habilitada, también podrá acceder presionando **Continuar con Google**.

Paso 4. Si olvidó su contraseña

Seleccione **¿Olvidaste tu contraseña?**, ingrese su correo y revise su bandeja para restaurar el acceso.

Componentes de Interfaz

La interfaz del sistema se divide en tres áreas principales que permiten una navegación

Incluye:

- Nombre y correo electrónico
- Menú de perfil
- Panel administrativo

Acciones que puede realizar:

- Editar perfil
- Acceso a la parte administrativa
- Cerrar sesión

Figura A6. Pasos e interfaz.

En la Figura A7 se muestra notificaciones y menú lateral.

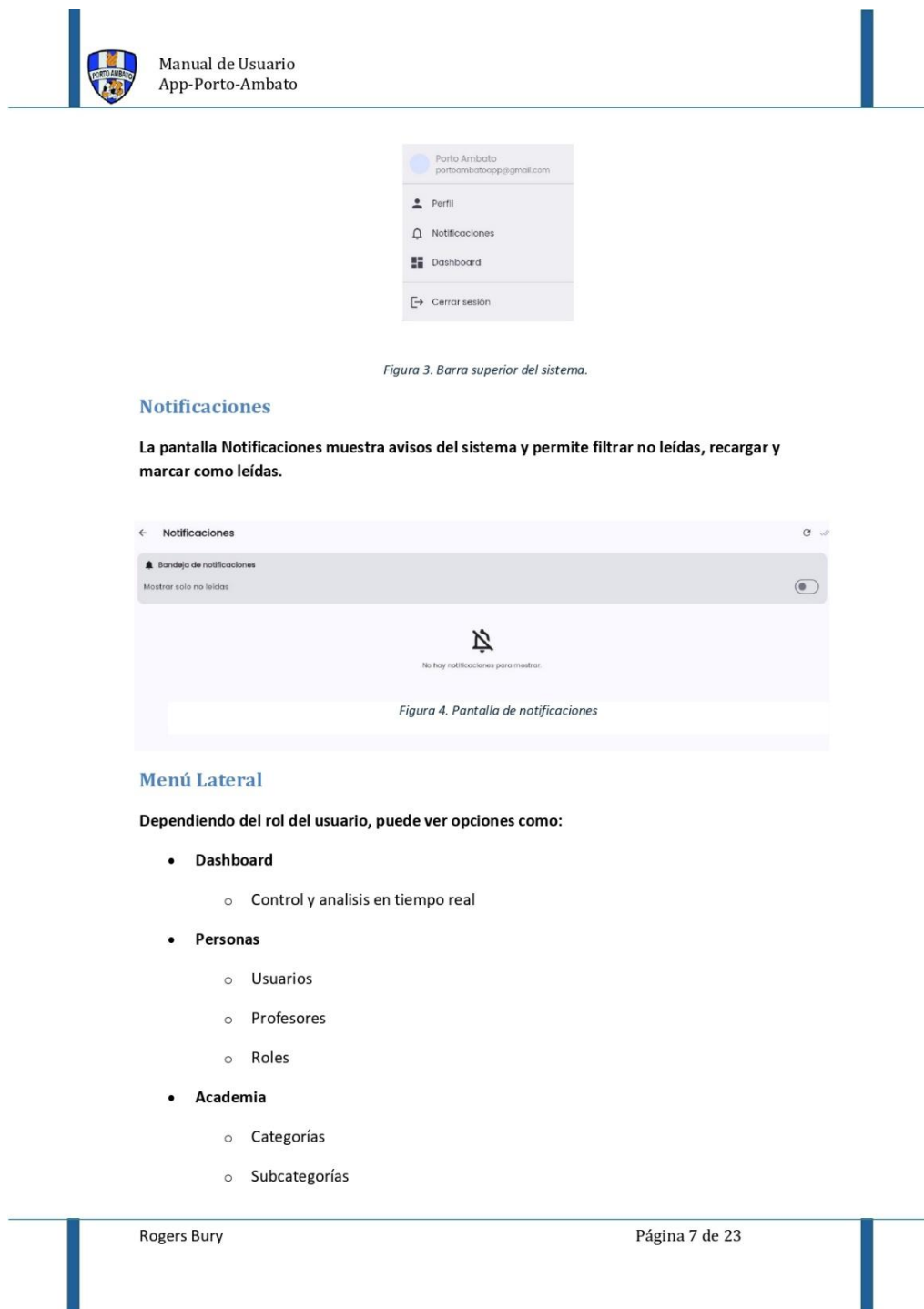


Figura A7. Módulo de notificaciones.

En la Figura A8 se muestra el menú lateral.

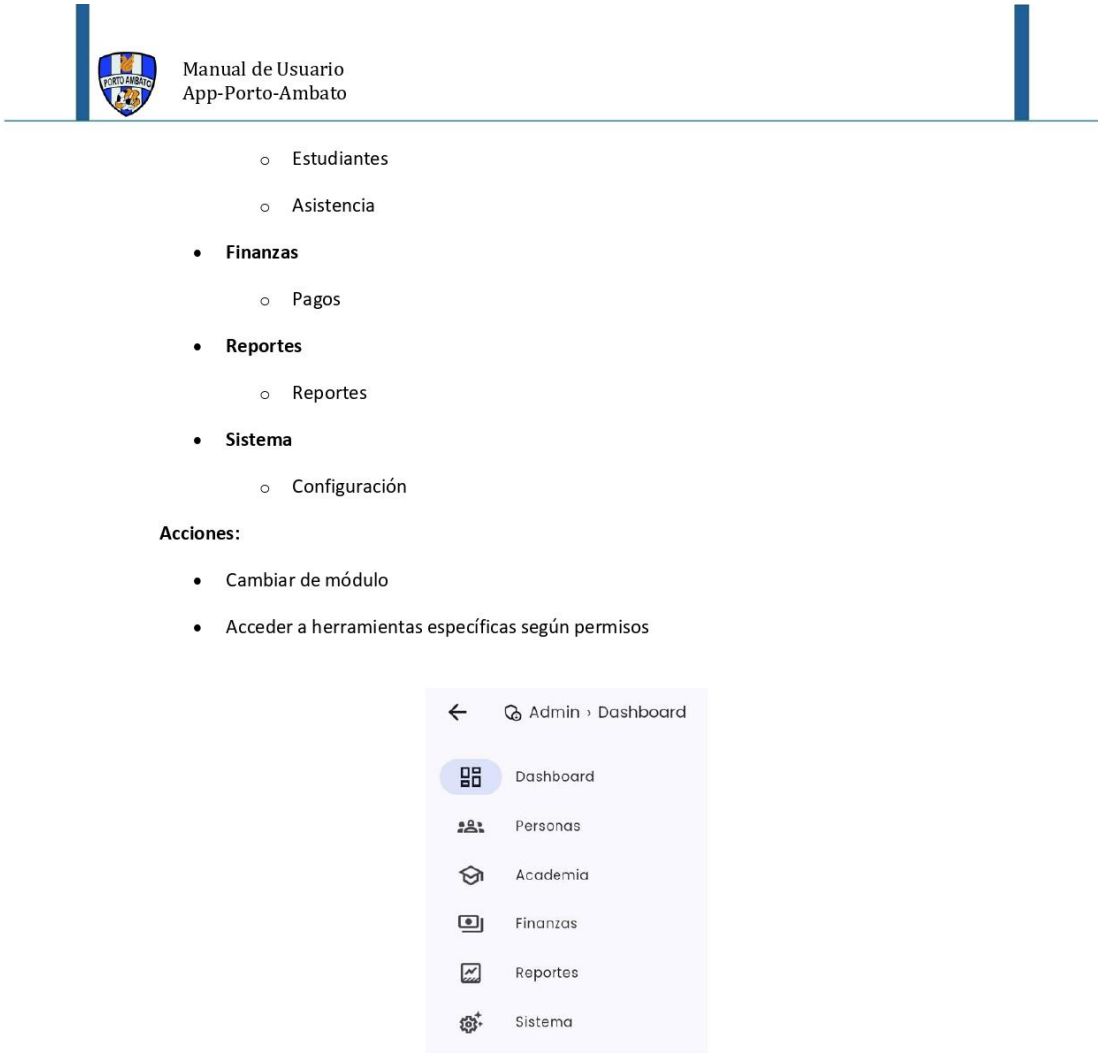


Figura 5. Menú lateral con los módulos principales.

Figura A8. Menú lateral.

En la Figura A9 se muestra el dashboard.



Dashboard

El Dashboard presenta indicadores clave del sistema en tiempo real y permite analizar la información mediante filtros de fecha y secciones General, Financiero y Académico.

Acciones principales

- Seleccionar rango de tiempo (7 días, 30 días, 3 meses, año).
- Cambiar de sección (General / Financiero / Académico).
- Actualizar los datos (recargar).
- Visualizar KPIs (estudiantes, profesores, categorías/subcategorías, facturado y recaudado).

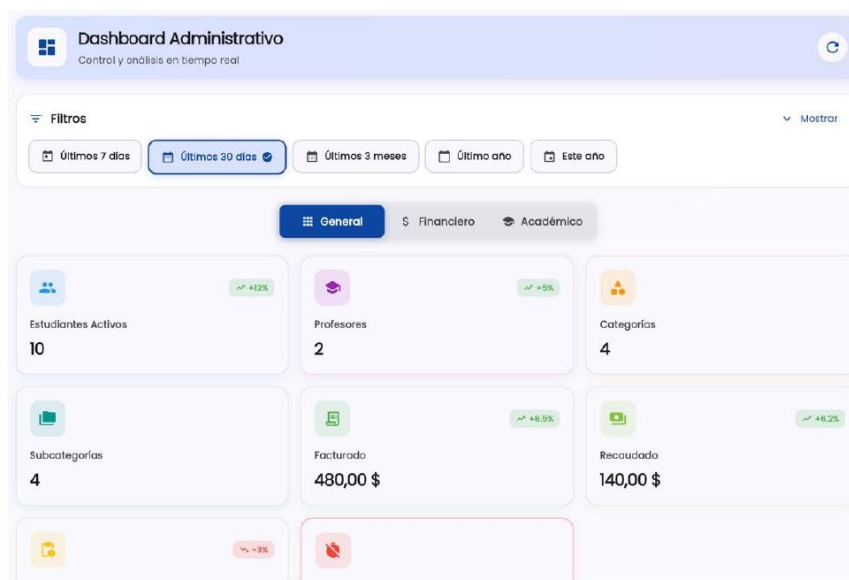


Figura 6. Dashboard administrativo.

En la Figura A10 se muestra el perfil.



Perfil

Acciones:

- Actualizar foto.
- Cambiar contraseña.
- Editar datos básicos.
- Verificar cedula ec.



Rogers Bury
rogersbury@gmail.com

Información Personal

Nombre completo
 Rogers Bury

Cédula (Ecuador)
 1850554450 

Cédula verificada ✓

Correo electrónico
 rogersbury@gmail.com

GUARDAR CAMBIOS

Seguridad

 Cambiar Contraseña

Figura 7. Perfil del usuario.

Figura A10. Módulo de perfil.

En la Figura A11 se muestra el módulo de usuarios.



Uso del Sistema

Módulo de Usuarios

Este módulo permite administrar todas las cuentas del sistema.

Acciones disponibles

1. Crear un usuario

1. Presione el botón **Crear usuario**.
2. Complete los datos requeridos.
3. Seleccione el **rol** del usuario.
4. Pulse **Guardar**.

2. Editar usuario

1. En la tabla, seleccione el ícono **Editar**.
2. Actualice los campos necesarios.
3. Pulse **Guardar**.

3. Desactivar usuario

1. Seleccione el ícono **Desactivar**.
2. Confirme la acción.

Nota: la eliminación es lógica. El usuario puede reactivarse.

Cédula	Nombre	Correo	Rol	Registro	Acciones
1803981479	Christian Llerena	portoambato@gmail.com	ADMIN	2025-11-24	
1850554450	Rogers Bury	rogersbury@gmail.com	ADMIN	2025-08-01	
—	Liseth Vargas	vargasliseth243@gmail.com	USUARIO	2025-11-25	
—	Kevin	kevoandrade019@gmail.com	PROFESOR	2025-11-25	
—	Kevin Escalante	kescalante_12@gmail.com	USUARIO	2025-11-24	

Figura 8. Gestión de usuarios del sistema.

En la Figura A12 se muestra módulo de profesores y roles.



Módulo de Profesores

Acciones disponibles:

1. Editar profesor

- Ícono **Editar** en la tabla.

2. Activar / Desactivar

- Ícono **Desactivar**.

Nota: la eliminación es lógica. El profesor puede reactivarse.

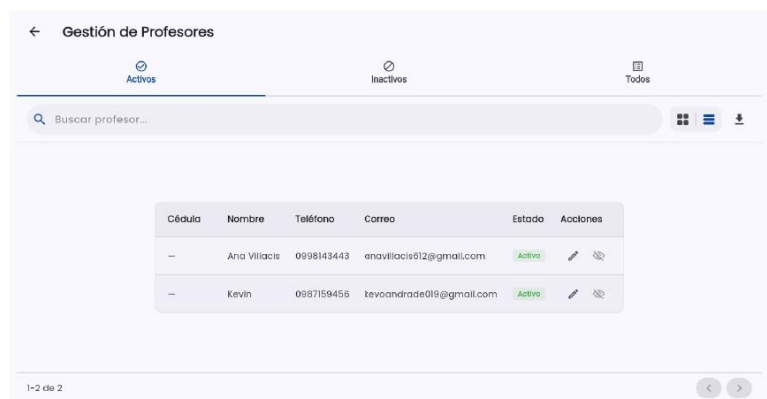


Figura 9. Módulo de profesores.

Módulo de Roles

Acciones disponibles

1. Crear un Rol

5. Presione el botón **Crear**.
6. Complete los datos requeridos.
7. Seleccione lo permisos del rol.
8. Pulse **Guardar**.

2. Editar Rol

4. En la tabla, seleccione el ícono **Editar**.
5. Actualice los campos necesarios.

Figura A12. Módulo de profesores y roles.

En la Figura A13 se muestra el módulo de categorías.



6. Pulse **Guardar**.

3. Eliminar Rol

3. Seleccione el ícono **eliminar**.

4. Confirme la acción.

Roles y permisos
Gestiona y controla el acceso al sistema

Buscar rol... + Nuevo

ID	Rol	Descripción	Permisos	Acciones
1	admin	Puede administrar toda la aplicación	100	
2	profesor	Profesor/a: gestiona evaluaciones y asistencia de su grupo	14	
3	representante	Representante: gestiona pagos y datos de su representado	8	
4	usuario	No tiene permisos	0	

Figura 10. Módulo de roles.

Módulo Categorías

Acciones:

Crear categoría

1. Presione **Nueva categoría**.
2. Ingrese nombre y rango de edades.
3. Pulse **Guardar**.

Editar categoría

1. Presione en el icono **Editar**.
2. Actualice los campos necesarios.

Figura A13. Módulo de categorías.

En la Figura A14 se muestra el módulo de subcategorías.



3. Pulse **Guardar**.

Desactivar/activar categoría

4. Presione en el icono **Desactivar/Activar**.

← Categorías					
Buscar por nombre...					
+ Nueva					
Activas (4) Inactivas (0) Todas (0)					
ID	Categoría	Edad	Estado	Creado	Acciones
3	Sub-6	5 - 6	Activo	2025-11-24	
2	Sub-8	7 - 8	Activo	2025-11-24	
1	Sub-10	9 - 10	Activo	2025-11-24	
4	Walkiki	2 - 10	Activo	2025-11-24	

1-4 de 4

Figura 11. Módulo de categorías.

Módulo de Subcategorías

Acciones:

Crear subcategoría

1. Presione **Nueva categoría**.
2. Ingrese los datos necesarios.
3. Pulse **Guardar**.

Editar subcategoría

1. Presione en el icono **Editar**.
2. Actualice los campos necesarios.
3. Pulse **Guardar**.

Desactivar/activar subcategoría

Figura A14. Módulo de subcategorías.

En la Figura A15 se muestra el módulo de estudiantes.



1. Presione en el icono **Desactivar/Activar**.

Ver estudiantes

1. Presione en el icono de ver **Estudiantes**.
2. Lista los estudiantes de esa subcategoría.

Asignación

1. Presione en el icono de **Asignación**.
2. Lista los estudiantes sin asignación a una subcategoría.
3. Presione los estudiantes que desea asignar.
4. Presione **Guardar**.

ID	Subcategoría	Código	Categoría	Estado	Creado	Acciones
4	Wakiki	Wakiki	Wakiki	Activo	2025-12-02	[Icons]
2	Sub-8-Tarde	Sub-8-Tarde	Sub-8	Activo	2025-12-02	[Icons]
3	Sub-8-Tarde	Sub-8-Tarde	Sub-8	Activo	2025-12-02	[Icons]
1	Sub-10-Tarde	Sub-10-Tarde	Sub-10	Activo	2025-8-24	[Icons]

Figura 12. Módulo de subcategorías.

Módulo de Estudiantes

Acciones:

Crear estudiante

1. Presione **Nuevo**.
2. Ingrese los datos necesarios.
3. Pulse **Guardar**.

Editar estudiante

1. Presione en el icono **Editar**.
2. Actualice los campos necesarios.
3. Pulse **Guardar**.

Desactivar/activar estudiante

Figura A15. Módulos módulo estudiantes.

En la Figura A16 se muestra el módulo de estudiantes y inscripción.



1. Presione en el icono **Desactivar/Activar**.

Ver estudiante

1. Presione en el icono de ver **Estudiante**.

2. Lista los datos del estudiante y además se puede ingresar a un resumen de sus mensualidades.

☐	Nombre	Cédula	Subcategoría	Estado	Acciones
☐	Adrian Guerra	—	Sub-10-Tarde	Activo	
☐	Agustin Nuñez	—	Sub-6-Tarde	Activo	
☐	Andre Gonzales	—	Sub-10-Tarde	Activo	
☐	Daniel Mayorga	—	Sub-10-Tarde	Activo	
☐	Gabriel Naniña	—	Sub-8-Tarde	Activo	
☐	Joaquin Andrade	—	Sub-6-Tarde	Activo	
☐	Martin Valle	—	Sub-10-Tarde	Activo	

1-10 de 10

Nuevo

Figura 13. Módulo estudiantes.

Nueva Inscripción

Datos Personales

Nombres * Apellidos *

Cédula Fecha Nacimiento

Dirección

Teléfono

Datos Académicos

Categoría * Subcategoría Seleccione Categoría

Ciclo Fecha Matricula 2025-12-02

Cancelar GUARDAR INSCRIPCIÓN

Figura 14. Pantalla para inscripción del estudiante.

Figura A16. Módulo de inscripción.

En la Figura A17 se muestra el módulo el detalle del estudiante y vinculación con el representante.

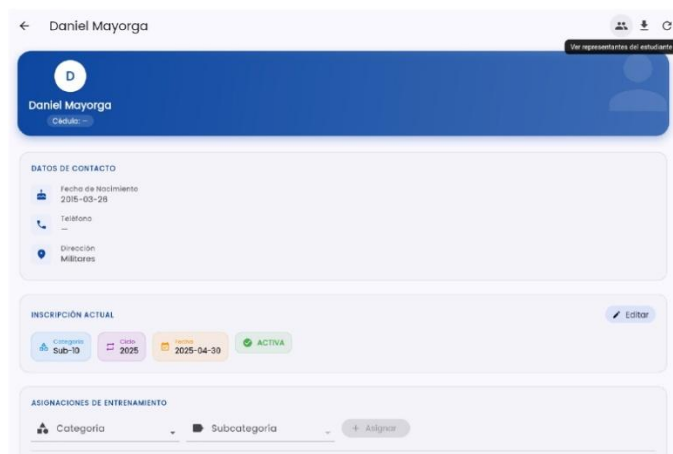


Figura 15. Pantalla de información del estudiante.

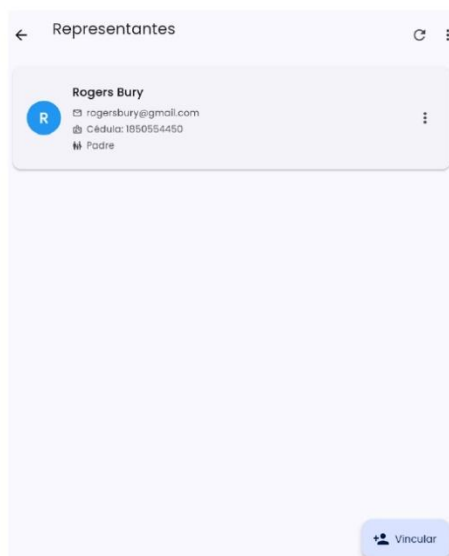


Figura 16. Pantalla de vinculación del representante.

Figura A17. detalle del estudiante y vinculación con el representante.

En la Figura A18 se muestra el módulo asistencia.



Módulo de asistencia

Acceder a la asistencia

1. Presione en la subcategoría que desea tomar la lista.

Registrar asistencia

1. Seleccione la fecha.
2. Marque a cada estudiante como:
 - Presente
 - Tarde
 - Ausente
 - Justificado
5. Pulse Guardar asistencia.

Consulta de asistencia anterior

1. Presione en historial, calendario, por alumno.
2. podrá ver un historial de acuerdo con la opción seleccionada.

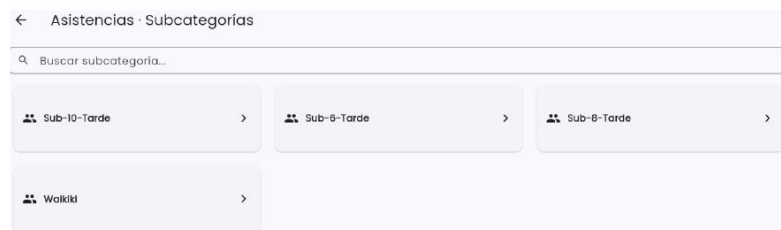


Figura 17. Módulo de Asistencia.

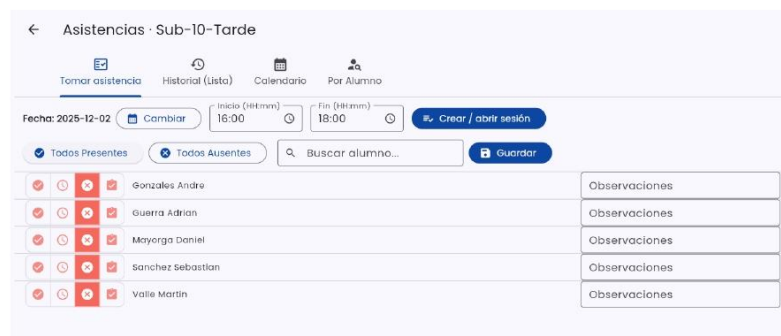


Figura 17. Registro de asistencias.

Figura A18. Módulo de asistencia.

En la Figura A19 se muestra módulo de pagos.



Módulo de pagos

Pasos para registrar un pago

1. Seleccione el estudiante.
2. Abra el módulo **Pagos**.
3. Seleccione la mensualidad pendiente.
4. Ingrese monto y método de pago.
5. Pulse **Registrar pago**.

Generar meses

1. Seleccione **Generar meses**.
2. Ingrese el valor de cada mensualidad y seleccione los meses a generar.
3. Pulse en guardar.

The screenshot shows the 'Gestión de Pagos' (Payment Management) screen. On the left, there's a list of students: Daniel Mayorga, Gabriel Namiña, Joaquín Andrade, Martín Valle, Miguel Galbar, Milton Cunalata, and Sebastián Sánchez. The main area displays the selected student, Daniel Mayorga, with a 'Generar meses' button. Below this, there are summary cards for 'Total' (80,00 \$), 'Pagado' (40,00 \$), and 'Pendiente' (40,00 \$). There are filters for 'Estado' (Todos) and 'Año' (Todos), and a 'Solo pendientes' button. The 'Historial de pagos' section shows 'Enero 2025' with a 'Pendiente: 4.000,00 \$' and 'Pagado: 0,00 \$', and 'Diciembre 2025' as 'Pagado'. A '+ Registrar pago' button is at the bottom.

Figura 18. Registro de pagos.

Figura A19. Modulo pagos.

En la Figura A20 se muestra parte de la generación masiva de mensualidades.



Registro masivo de mensualidades

La ventana **Generación Masiva por Subcategoría** permite registrar mensualidades en grupo mediante un asistente por pasos: selección de subcategoría, carga de estudiantes, definición de mes/valor, resumen y confirmación.

Acciones principales

- Seleccionar **subcategoría**.
- Avanzar con **Siguiente** o salir con **Cancelar**.
- Revisar **resumen y confirmar** la generación masiva.

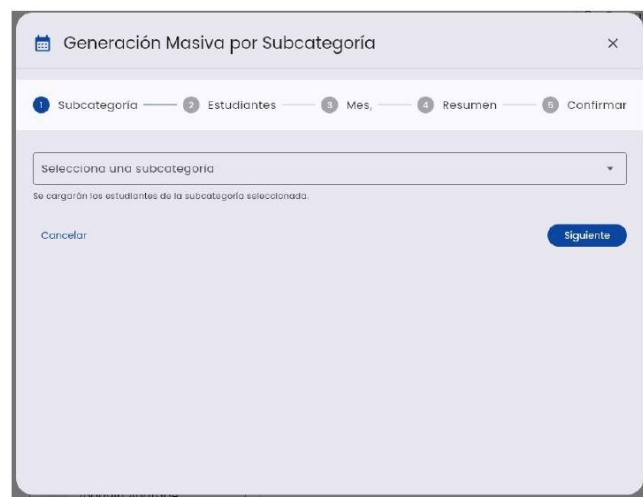


Figura 19. Registro masivo de Mensualidades.

Módulo de Reportes

Componentes principales

1. Filtros

- **Desde / Hasta:** Permite seleccionar un rango de fechas.
- **Método de pago:** Efectivo, transferencia, depósito o todos.
- **Estado:** Pagado, pendiente, anulado o todos.

Figura A20. Inicio de módulo de generación masiva de mensualidades.

En la Figura A21 se muestra el módulo de reportes.



2. Botón “Generar PDF”

Crea un archivo PDF con los resultados según los filtros seleccionados.

Pasos para generar el reporte

1. Seleccione el rango de fechas (opcional).
2. Elija el método de pago (opcional).
3. Seleccione el estado (opcional).
4. Presione **Generar PDF**.
5. Descargue y revise el archivo generado.

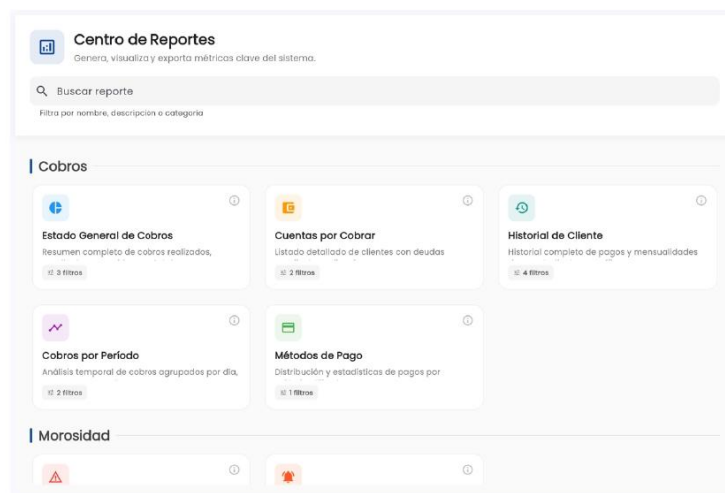


Figura 20. Módulo de Reporte del Sistema.

Configuración General del Sistema

La sección Configuración permite personalizar la aplicación y administrar preferencias como tema, tamaño de texto, modo compacto, reducción de animaciones y notificaciones.

Acciones principales

- Cambiar tema (Auto / Claro / Oscuro).
- Ajustar tamaño de texto.
- Activar modo compacto.

Figura A21. Módulo de reportes.

En la Figura A22 se muestra el módulo de configuración.



- Activar reducir animaciones.
- Acceder y gestionar notificaciones.

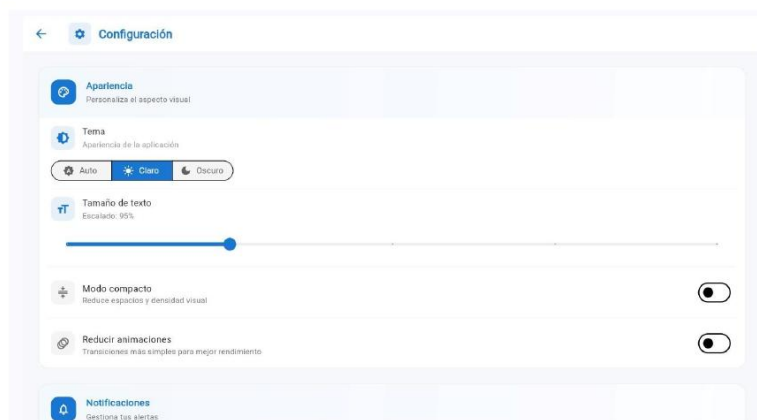


Figura 20. Pantalla de configuración.

En la Figura A23 se muestra la guía rápida de botones.



Anexo

Guía rápida de Botones

Ícono / Botón	Nombre	Función
Crear / Nuevo	Crear registro	Abre un formulario para crear usuarios, estudiantes, categorías, subcategorías, mensualidades o roles.
Editar	Editar registro	Permite actualizar los datos de un elemento seleccionado.
Desactivar / Eliminar	Eliminación lógica	Desactiva el registro sin borrarlo definitivamente (aplicable a usuarios, profesores, estudiantes, categorías y subcategorías).
Ver / Detalle	Ver información	Abre la vista detallada de estudiantes, incluyendo datos personales, mensualidades y asignaciones.
Asignar	Asignación	Permite asignar estudiantes a subcategorías o cambiar su subcategoría.
PDF	Generar PDF	Reportes en formato PDF.
Actualizar / Recargar	Recargar datos	Refresca la tabla o la pantalla actual.
Buscar	Buscar registro	Filtra información en tablas mediante texto.
Guardar	Guardar cambios	Confirma modificaciones o creación de un nuevo registro.
Cancelar	Cancelar	Cierra un formulario o descarta los cambios realizados.
Historial	Ver historial	Visualiza listas anteriores (por ejemplo, historial de asistencia).
Calendario	Selección de fecha	Permite ver asistencias por días o seleccionar rango de fechas en reportes.
Cambiar Contraseña	Seguridad	Abre el módulo de edición de contraseña del usuario.
Configuración	Configuración	Acceso al módulo de ajustes del sistema.
Cerrar Sesión	Salir	Cierra la sesión actual de usuario.

Tabla 1. Guía Rápida de Botones.

Figura A23. Guía rápida de botones.

Anexo B. Resultado de la fiabilidad de la encuesta en Spss.

En la Figura A20 se muestra el resultado del Alfa de Cronbach en Spss.

Estadísticas de fiabilidad	
Alfa de Cronbach	N de elementos
,768	10

Figura A20. Resultado del Alfa de Cronbach en Spss.

Anexo C. Repositorio del frontend.

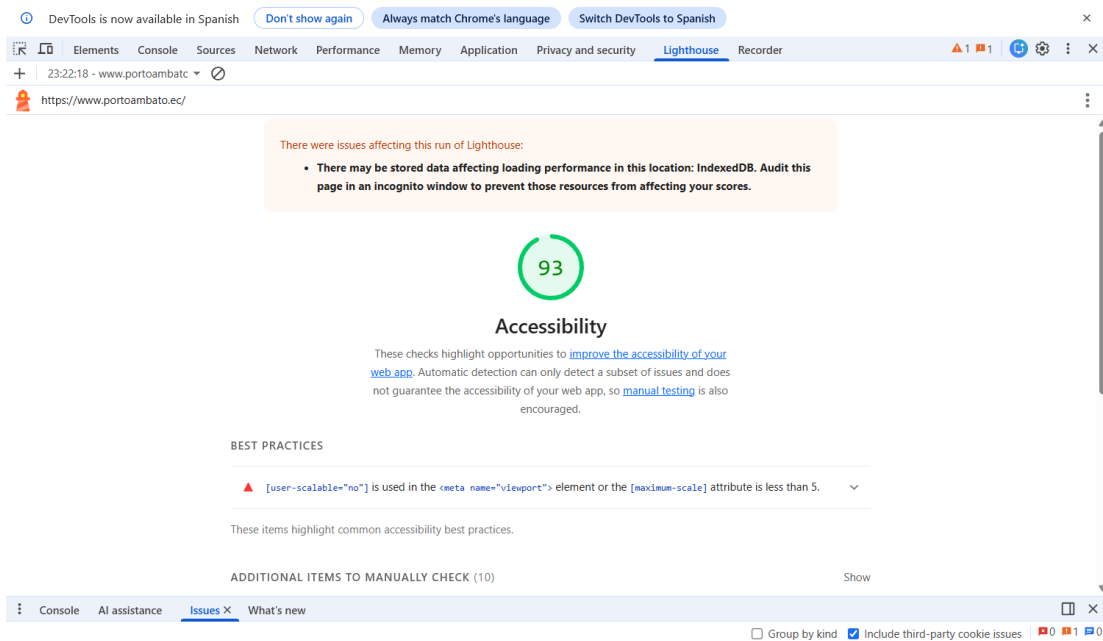


Figura A21. Resultado del análisis de accesibilidad.

En la Figura A21 se muestra el repositorio del frontend.

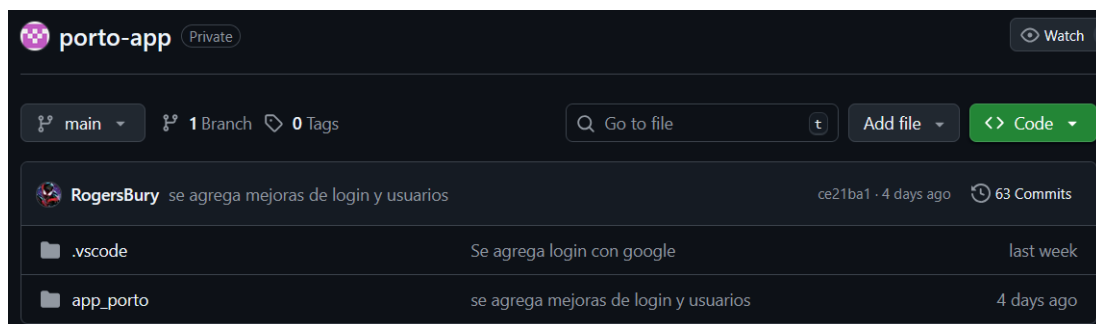


Figura A21. Repositorio del front-end.

Anexo D. Repositorio del backend.

En la Figura A22 se muestra el repositorio del backend.



Figura A22. Repositorio del backend.