

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“DISEÑO, DESARROLLO E IMPLEMENTACIÓN DE UN
SISTEMA PARA LA COORDINACIÓN DE EVALUACIONES
EN EL DEPARTAMENTO DE INFORMÁTICA.”

TOMÁS JAVIER ESCALONA YÁÑEZ

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Pedro Francisco Godoy Barrera
Profesor Correferente: Luz Mairet Chourio Acevedo

Julio - 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: **Diseño, desarrollo e implementación de un sistema para la coordinación de evaluaciones en el Departamento de Informática**

Nombre del candidato(a): **Tomás Javier Escalona Yáñez**

Carrera / Grado: **Ingeniería Civil Informática**

Campus: **Casa Central**

Departamento: **Departamento de Informática**

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, **Pedro Francisco Godoy Barrera**, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 17-08-2026

Firma: _____

Estudiante o Candidato(a):

Fecha: 17-08-2026

Firma: _____

DEDICATORIA

A todas las personas que hacen del mundo un lugar más cálido.

AGRADECIMIENTOS

Primero quiero agradecer a mi mamá, por aguantar todas las cosas que le pedía mientras trabajaba en esto, por motivarme constantemente, y siempre estar al lado mío apoyándome a pesar de todas mis mañas, ideas locas como irme a estudiar justo donde no tengo familia, y por cuidarme y quererme tanto.

A mi gata regalona, que aguantó que sus momentos de juego pasaran a ser menos frecuentes, que de vez en cuando me demorara más de lo normal en darle comida, y que soporta que la abrace tanto.

A toda mi familia extendida, que a pesar de que soy malo para mantenerme en contacto, sé que siempre van a estar ahí para mí.

También quiero agradecer a todas las personas que ya no están aquí y que fueron parte de mi vida, porque todos dejaron su marca en mí y por eso siempre los llevaré en mi corazón.

Agradecer a todos mis amigos con los que interactué a larga distancia, que muchas noches de juegos y risas me ayudaron tanto a relajarme en momentos de estrés.

Y finalmente a quienes me inspiran a ser mi mejor yo.

RESUMEN

Resumen— El aprendizaje de un estudiante es comúnmente medido a través de evaluaciones, por lo cual la planificación de estas afecta directamente el rendimiento académico y la carga temporal a la que se ve sometido el estudiante. Actualmente en el Departamento de Informática de la Universidad Técnica Federico Santa María el proceso de planificación se realiza de manera manual. En esta memoria se presenta el diseño, desarrollo e implementación de un sistema de coordinación de evaluaciones, con el cual se busca disminuir cargas académicas temporales, y así mejorar el rendimiento académico de los estudiantes.

Palabras Clave— Planificación, Rendimiento Académico, Sistema de Coordinación, Aplicación Web.

ABSTRACT

Abstract— Student learning is commonly measured through evaluations, so their planning directly affects both academic performance and student workload. Currently at the Computer Science Department of Universidad Técnica Federico Santa María, this planning is done manually. This work presents the design, development, and implementation of an evaluation coordination system, which aims to reduce concentrated evaluation periods and thus improve students' academic performance.

Keywords— Planning, Academic Performance, Coordination System, Web Application

GLOSARIO

ACID Atomicity, Consistency, Isolation, Durability.

API Application Programming Interface.

BD Base de datos.

CORS Cross-Origin Resource Sharing.

CRUD Create, Read, Update, Delete.

CSS Cascading Style Sheets.

DI Departamento de Informática.

ER Entidad-Relación.

FK Foreign Key (clave foránea).

HMAC Hash-based Message Authentication Code.

HMR Hot Module Replacement.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS HTTP Secure.

ICI Ingeniería Civil Informática.

IP Internet Protocol.

JSON JavaScript Object Notation.

JWT JSON Web Token.

LMS Learning Management System.

ORM Object-Relational Mapper.

REST Representational State Transfer.

RF Requerimiento Funcional.

RNF Requerimiento No Funcional.

SCT Sistema de Créditos Transferibles.

SIGA Sistema de Información y Gestión Académica.

SPA Single Page Application.

SQL Structured Query Language.

SUS System Usability Scale.

TLS Transport Layer Security.

UI User Interface.

URL Uniform Resource Locator.

UTFSM Universidad Técnica Federico Santa María.

UUID Universally Unique Identifier.

XSS Cross-Site Scripting.

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	X
ÍNDICE DE TABLAS	XI
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Contexto	3
1.2 Proceso actual de planificación de evaluaciones	3
1.2.1 Evaluaciones coordinadas	3
1.2.2 Evaluaciones no coordinadas	4
1.3 Problemas encontrados	5
1.4 Objetivos	6
CAPÍTULO 2: MARCO CONCEPTUAL	7
2.1 Planificación académica y rendimiento estudiantil	7
2.1.1 Práctica distribuida y carga académica	7
2.1.2 Enfoques existentes y la brecha que dejan	8
2.2 Aplicaciones web	10
2.2.1 Aplicación web	10
2.2.2 Arquitectura cliente-servidor	11
2.2.3 Single Page Application (SPA)	11
2.2.4 API REST	12
2.2.5 CORS	12
2.3 Arquitectura del sistema	12
2.3.1 Arquitectura monolítica	13
2.3.2 Contenedores y Docker	13
2.3.3 Nginx	13
2.4 Bases de datos relacionales	13
2.4.1 Modelo relacional	13
2.4.2 PostgreSQL	14
2.4.3 Migraciones de base de datos	14
2.5 Back-end	14
2.5.1 Go	15
2.5.2 Autenticación con JWT y cookies httpOnly	15
2.5.3 HMAC	15
2.5.4 ORM	15

2.5.5	bcrypt	16
2.6	Front-end	16
2.6.1	Vue 3	16
2.6.2	TypeScript	16
2.6.3	Vite	17
2.6.4	PrimeVue	17
2.6.5	vue-router	17
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN		18
3.1	Actores y casos de uso	18
3.2	Requerimientos del sistema	19
3.3	Trazabilidad	20
3.4	Arquitectura general	21
3.5	Diseño de base de datos	23
3.5.1	Usuarios y permisos	23
3.5.2	Catálogo académico	25
3.5.3	Oferta académica y evaluaciones	26
3.5.4	Importación masiva	26
3.5.5	Programas y malla curricular	27
3.5.6	Decisiones de diseño	28
3.5.7	Normalización, índices e integridad	28
3.6	Diseño back-end	29
3.6.1	Decisiones tecnológicas	29
3.6.2	Estructura de paquetes	33
3.6.3	Autenticación y autorización	35
3.7	Diseño de la API	35
3.7.1	Flujos de solicitud representativos	37
3.8	Infraestructura y despliegue	41
3.8.1	Docker	41
3.8.2	Docker Compose	41
3.8.3	Nginx	41
3.8.4	Topología de despliegue	42
3.9	Diseño front-end	42
3.9.1	Decisiones tecnológicas	42
3.9.2	Estructura del SPA	45
3.10	Diseño interfaz de usuario	46
3.10.1	Calendario de evaluaciones	46
3.10.2	Gestión de evaluaciones	48
3.10.3	Flujo de importación masiva	49
3.10.4	Panel de administración	52
3.11	Consideraciones de seguridad	52
3.12	Nuevo proceso de planificación de evaluaciones	55
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		57

4.1	Metodología de validación	57
4.2	Pruebas automatizadas	58
4.3	Importación masiva de evaluaciones	59
4.3.1	Carga del archivo	59
4.3.2	Revisión de filas	59
4.3.3	Commit y validación previa	62
4.4	Gestión individual de evaluaciones	63
4.5	Calendario público y detección de conflictos	66
4.6	Prueba de carga y concurrencia	68
4.7	Evaluación de usabilidad (SUS)	71
4.7.1	Protocolo de prueba	72
4.7.2	Resultados: Jefes de Carrera	72
4.7.3	Resultados: Personal Académico (ayudantes)	73
4.7.4	Resultados: Personal Académico (profesores)	74
4.7.5	Lectura conjunta de los resultados	74
4.8	Hallazgos de la validación	75
4.8.1	Un defecto expuesto por el uso real	75
4.8.2	Mejoras de usabilidad detectadas	76
CAPÍTULO 5: CONCLUSIONES		78
5.1	Resumen del trabajo realizado	78
5.2	Objetivos realizados	78
5.3	Límites	79
5.4	Trabajo futuro	80
5.5	Reflexión final	81
REFERENCIAS BIBLIOGRÁFICAS		82
ANEXO A: MANUAL DE USUARIO		85

ÍNDICE DE FIGURAS

1	Proceso actual de planificación de evaluaciones	4
2	Diagrama de casos de uso del sistema	19
3	Arquitectura general del sistema en capas	22
4	Modelo entidad-relación completo del esquema	24
5	Tabla de <i>staging</i> exam_imports	27
6	Diagrama de componentes del back-end	34
7	Niveles de autorización, anidados según el alcance de datos visible.	36
8	Diagrama de secuencia del flujo de autenticación (login).	39
9	Diagrama de secuencia de la creación de una evaluación con la restricción de exclusión.	39
10	Diagrama de secuencia de la importación masiva desde Excel, en sus cuatro fases.	40
11	Topología de despliegue en contenedores Docker	43
12	Escala de severidad usada para clasificar las semanas con conflictos.	46
13	Calendario público de evaluaciones con detección automática de conflictos.	48
14	Vista de gestión de evaluaciones para usuarios autenticados.	49
15	Ciclo de vida de una fila de importación.	51
16	Revisión de filas en el flujo de importación masiva desde Excel.	51
17	Panel de administración del sistema.	52
18	Nuevo proceso de planificación de evaluaciones	56
20	Confirmación de carga exitosa con el número de filas reconocidas.	59
19	Selección del archivo Excel y semestre de destino para la carga.	60
21	Filas de importación agrupadas por asignatura, con su estado de resolución.	61
22	Resolución manual de una fila bloqueada por falta de coincidencia de asignatura.	61
23	Resumen de validación previa al commit, con filas que se importarán y filas que fallarán.	62
24	Resultado de un commit exitoso.	62
25	Resultado de un commit parcialmente exitoso.	63
26	Resultado de un commit fallido.	63
27	Listado filtrable de evaluaciones en la página de gestión.	64
28	Formulario de creación de una nueva evaluación.	64
29	Rechazo de una evaluación que se cruza en el horario con otra existente.	65
30	Confirmación previa al borrado suave de una evaluación.	65
31	Listado con evaluaciones eliminadas visibles y opción de restauración.	66
32	Calendario público de evaluaciones, accesible sin autenticación.	67
33	Selector de semestre del calendario público.	67
34	Vista semanal del calendario con el detalle de horario y sala de cada evaluación.	67
35	Selección de asignaturas por malla y semestre curricular.	68
36	Detalle de conflictos de la semana, agrupados por severidad.	68
37	Formulario de inicio de sesión.	85
38	Creación de una oferta de asignatura y de sus paralelos.	87

39	Administración de usuarios y asignación de departamentos.	89
40	Gestión de semestres.	90
41	Registro de departamentos y de sus códigos.	90
42	Listado de mallas curriculares registradas.	90
43	Asignación de asignaturas a los semestres curriculares de una malla.	91

ÍNDICE DE TABLAS

1	Enfoques existentes frente a las condiciones del problema de coordinación. .	11
2	Requerimientos funcionales del sistema.	20
3	Requerimientos no funcionales del sistema.	20
4	Matriz de trazabilidad: problema → requerimiento → funcionalidad.	21
5	Resumen de grupos de endpoints de la API REST.	38
6	Modelo de amenazas: amenazas consideradas y controles que las mitigan. . .	54
7	Requerimientos y nivel de validación que aporta su evidencia.	58
8	Resultados de la prueba de carga sobre la consulta del calendario público, en el equipo de desarrollo.	69
9	Prueba de carga sobre el servidor pequeño simulado	70
10	Columnas de la plantilla de importación	88

INTRODUCCIÓN

La educación superior requiere de sus estudiantes un alto nivel de compromiso, por lo cual son continuamente evaluados a través de exámenes y trabajos. Estas evaluaciones son el mecanismo principal por el cual el aprendizaje y el avance académico de los estudiantes son determinados. La manera en que estas evaluaciones son programadas a través del semestre puede tener efectos profundos en el rendimiento del estudiante [Richardson *et al.*, 2012, Vázquez *et al.*, 2012]. Cuando múltiples exámenes están concentrados en un rango de tiempo corto, los estudiantes se ven enfrentados a una carga académica que puede limitar su capacidad de estudiar efectivamente, llevando a un aumento de estrés y disminuyendo sus resultados académicos.

La práctica espaciada consiste en distribuir las sesiones de estudio a lo largo del tiempo, en lugar de concentrarlas inmediatamente antes de una evaluación. Estudios en ciencia cognitiva han demostrado constantemente que este enfoque lleva a una retención y comprensión del material aprendido significativamente más efectivas [Cepeda *et al.*, 2006]. En cambio, periodos con alta concentración de evaluaciones generan estrés académico y carga cognitiva elevada al exigir preparación simultánea para múltiples asignaturas, condiciones que afectan negativamente el rendimiento [Vázquez *et al.*, 2012, Richardson *et al.*, 2012]. Una distribución adecuada de evaluaciones a lo largo del semestre permite a los estudiantes organizar mejor su tiempo de estudio, aprovechando los beneficios de la práctica espaciada. Este problema se agrava cuando las asignaturas son gestionadas independientemente y los profesores no tienen una visión global del calendario de evaluaciones de otras asignaturas, llevando a situaciones donde evitar coincidencias de fechas de exámenes se vuelve imposible coordinando manualmente.

En el Departamento de Informática (DI) de la Universidad Técnica Federico Santa María (UTFSM), el proceso de programación de exámenes se realiza generalmente de manera manual y descentralizada. Cada profesor determina de manera independiente las fechas de las evaluaciones de su asignatura, sin tener una vista unificada de las evaluaciones del semestre que afectarían a sus estudiantes. Esto lleva a periodos recurrentes de presión académica concentrada que afecta negativamente el bienestar del estudiante y su rendimiento académico.

Esta memoria presenta el diseño, desarrollo e implementación de un sistema de coordinación de evaluaciones para el DI, con el objetivo de resolver este problema. Este sistema proporciona a los profesores, estudiantes y administradores una vista semestral unificada de todas las evaluaciones programadas, y genera alertas cuando detecta periodos de alta carga académica. Al habilitar la planificación coordinada e informada de evaluaciones, el sistema busca reducir estos *peaks* de alta concentración de carga académica, creando así condiciones más conducentes para un aprendizaje efectivo.

En el capítulo 1 se describe el problema en detalle y se establecen los objetivos del trabajo. El

capítulo 2 presenta el marco conceptual, abarcando la literatura relevante y las tecnologías utilizadas. El capítulo 3 describe la solución propuesta, incluyendo la arquitectura del sistema y su diseño. El capítulo 4 presenta la validación de la solución. Finalmente, las conclusiones y el trabajo futuro son discutidos en el capítulo 5.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contexto

La Universidad Técnica Federico Santa María (UTFSM) ofrece múltiples programas de estudio que cubren varios aspectos técnicos y científicos en el ámbito de la ingeniería. Como parte de su estructura académica, los estudiantes cursan múltiples asignaturas cada semestre, en las cuales son evaluados periódicamente a través de tareas, pruebas y certámenes.

Estas evaluaciones son el principal mecanismo por el cual el aprendizaje del estudiante es medido y su progreso académico es determinado. Los estudios muestran de manera sistemática que el rendimiento académico está determinado por un amplio rango de factores interconectados [Richardson *et al.*, 2012, Vázquez *et al.*, 2012], entre los cuales la forma en que los estudiantes se preparan para sus evaluaciones desempeña un rol importante [Credé y Kuncel, 2008]. Estados de ánimo como el estrés y la ansiedad, influenciados por cómo las evaluaciones están distribuidas a lo largo del semestre, también contribuyen a estos resultados [Vázquez *et al.*, 2012, Richardson *et al.*, 2012].

1.2. Proceso actual de planificación de evaluaciones

La programación de evaluaciones en el Departamento de Informática (DI) se lleva a cabo de manera descentralizada, sin un sistema unificado para coordinar fechas entre asignaturas. Este proceso puede dividirse en dos categorías según el tipo de evaluación.

1.2.1. Evaluaciones coordinadas

Una parte significativa de las asignaturas que los estudiantes de Ingeniería deben tomar son externas al Departamento de Informática, por ejemplo, las asignaturas de los departamentos de Matemáticas y Física. Las asignaturas de los distintos niveles de Cálculo y Física son tomadas simultáneamente por estudiantes de múltiples ingenierías, por lo cual sus evaluaciones son coordinadas por estos departamentos y publicadas con fechas fijas. El DI no tiene mayor injerencia en definir estas fechas y debe programar sus propias evaluaciones considerándolas.

También existen asignaturas coordinadas dentro del DI, es decir, asignaturas con múltiples paralelos dictados por distintos profesores pero que comparten evaluaciones. Estas también requieren coordinación entre los profesores, proceso que igualmente se realiza de forma manual.

1.2.2. Evaluaciones no coordinadas

Para asignaturas con menor cantidad de estudiantes, el proceso típico es que cada profesor define las fechas de sus evaluaciones al inicio del semestre, las cuales luego son anunciadas a sus estudiantes junto al programa de la asignatura, generalmente durante la primera clase. En esta instancia el profesor le pregunta informalmente a sus estudiantes si existe algún conflicto de fechas con otras asignaturas.

Esta forma de programación presenta limitaciones fundamentales. El profesor de una asignatura no coordinada no tiene ninguna forma de conocer las fechas que otras asignaturas ya fijaron, de modo que por sí mismo no puede saber si sus evaluaciones se topan con las de otros certámenes. La detección de un tope queda supeditada por completo a que algún estudiante lo advierta y se lo informe, lo que muchas veces no ocurre. La pregunta informal de la primera clase tampoco lo resuelve, porque al inicio del semestre es frecuente que los propios estudiantes aún no conozcan todas sus fechas, dado que el resto de sus profesores tampoco las ha anunciado todavía. Como consecuencia, el tope suele salir a la luz recién a pocos días de las evaluaciones, cuando resolverlo ya se ha vuelto complejo: mover una fecha comprometida depende de la disposición del profesor y, sobre todo, de que exista una sala disponible para la nueva fecha, algo difícil de asegurar con tan poco margen. Esto significa que la coordinación depende de la memoria y la disponibilidad de los involucrados, en vez de una visión sistémica del calendario académico.

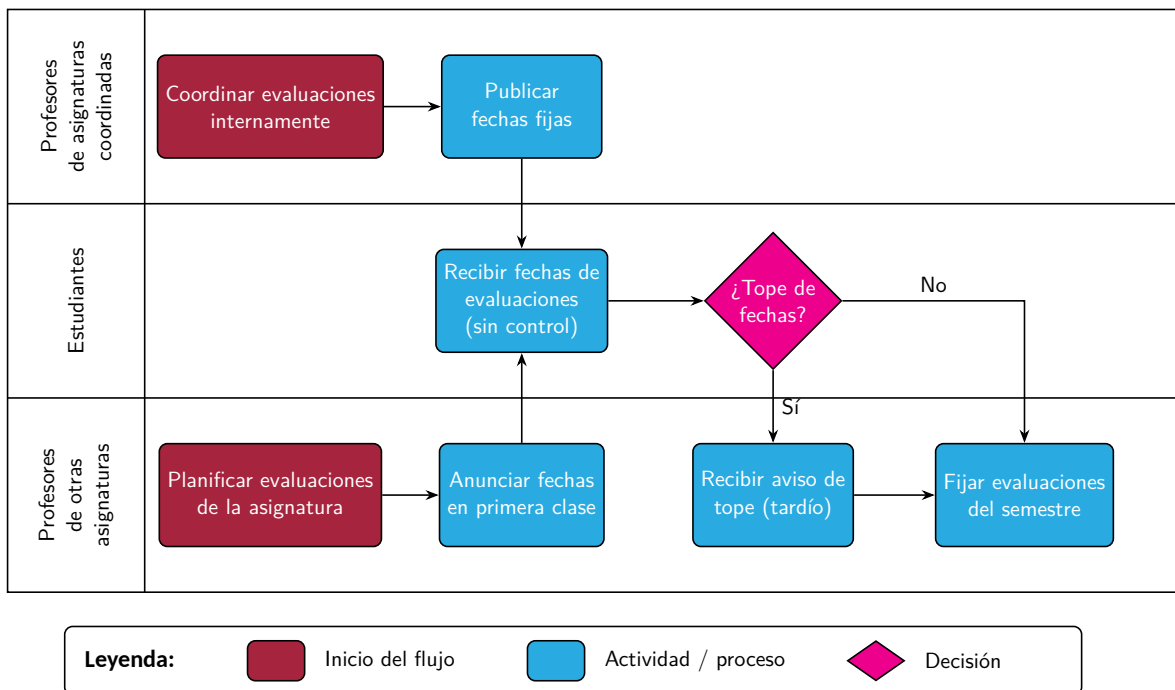


Figura 1: Proceso actual de planificación de evaluaciones en el DI. Cada carril representa un actor; las cajas son actividades y los rombos, decisiones.

Fuente: Elaboración propia.

Es importante notar que este ajuste informal de fechas no constituye una solución al problema de fondo: su éxito depende completamente de que el estudiante tome la iniciativa de informar el tope, de la disposición del profesor para modificar la fecha y de que exista una sala disponible para hacerlo, sin que ningún mecanismo garantice o facilite ese resultado. Cuando el ajuste no es posible o no se intenta a tiempo, el estudiante no tiene más alternativa que rendir ambas evaluaciones, aceptar un peor desempeño en alguna de ellas, o en casos extremos abandonar la asignatura. El problema de la planificación descoordinada persiste a nivel de proceso incluso en los casos individuales donde un ajuste informal tiene éxito.

1.3. Problemas encontrados

La ausencia de un sistema integrado de programación genera varios problemas recurrentes que afectan tanto a los estudiantes como a los profesores.

- **Falta de visibilidad entre asignaturas:** Al momento de definir las fechas de sus evaluaciones, los profesores no tienen información acerca de otras evaluaciones programadas en otras asignaturas. Esto lleva a semanas con alta densidad de evaluaciones simultáneas, lo que expone a los estudiantes a una carga académica concentrada que dificulta la preparación para cada evaluación.
- **Imposibilidad de coordinación universal:** Como los estudiantes de un mismo nivel no necesariamente toman las mismas asignaturas, o no siguen la malla académica por diversas razones, resulta inviable encontrar una combinación de fechas que evite conflictos para cada estudiante. La escala del problema descarta además la coordinación manual: aunque el DI imparte principalmente un programa, las evaluaciones que enfrenta cada uno de sus estudiantes provienen de asignaturas dictadas por numerosos departamentos de la universidad, muchas de ellas cursadas simultáneamente por estudiantes de distintos programas de ingeniería. Coordinar las fechas de modo que ningún estudiante quede con un tope supondría consolidar la programación de una parte considerable de la oferta de la institución, con cada asignatura dictada en varios paralelos y con varias evaluaciones por semestre. En la práctica, esas fechas no están consolidadas en ningún registro común al que los actores del proceso puedan recurrir.
- **Semanas con carga crítica de evaluaciones:** Un síntoma de este problema es la *semana de receso*, un periodo de descanso oficial a mitad del semestre que los estudiantes coloquialmente denominan “*semana anti-suicidio*”. Este nombre refleja la percepción de que la concentración de evaluaciones alrededor de esta semana llega a niveles críticos. La institución ha llegado a regular explícitamente la programación de evaluaciones en los días inmediatamente anteriores y posteriores a esta semana, reconociendo implícitamente la existencia del fenómeno. Incluso con este tipo de reglas, la concentración de evaluaciones persiste sin una solución sistemática.

- **Falta de registro histórico:** Como la gran mayoría de la programación de evaluaciones es informal, no existe un registro histórico de las evaluaciones y la carga académica real a la que los estudiantes se ven enfrentados. Esto significa que al momento de elegir las asignaturas que cursarán, se guían principalmente por las medidas oficiales de supuesta carga horaria, que históricamente ha sido cuestionada por los estudiantes.
- **Rendimiento académico y bienestar:** Una planificación deficiente deteriora el bienestar de los estudiantes y, con ello, su rendimiento académico. El efecto trasciende el plano individual: incide en la retención estudiantil y compromete así la misión de la institución educativa.
- **Dificultad de intervención por parte de los jefes de carrera:** Incluso si los jefes de carrera se enteran de la alta carga académica de los estudiantes, resulta difícil intervenir cuando el semestre ya está en curso y las fechas de las evaluaciones fueron determinadas semanas atrás. No existe un mecanismo que los alerte con anticipación para intervenir de manera proactiva en vez de reactiva.

1.4. Objetivos

Objetivo general: Diseñar, desarrollar e implementar un sistema de coordinación que permita una vista global semestral unificada de la carga académica de los estudiantes en el Departamento de Informática de la Universidad Técnica Federico Santa María.

Objetivos específicos:

- Investigar y caracterizar el proceso actual de planificación de evaluaciones y las necesidades de los actores involucrados.
- Establecer un mecanismo que permita cargar las evaluaciones de las asignaturas impartidas cada semestre y que alerte cuando una evaluación tenga tope con otra del mismo semestre.
- Disponibilizar un mecanismo que permita a estudiantes y profesores visualizar, en una vista unificada, las evaluaciones de las asignaturas que cursan o dictan.

CAPÍTULO 2

MARCO CONCEPTUAL

Este capítulo presenta los fundamentos conceptuales y tecnológicos sobre los que se construye la solución propuesta. Se parte del fenómeno que motiva el trabajo, la relación entre la planificación de evaluaciones y el rendimiento académico, y de las herramientas actualmente disponibles para gestionar calendarios académicos; a partir de sus limitaciones se introducen los conceptos de ingeniería de software necesarios para comprender el diseño del sistema: las aplicaciones web y su arquitectura, la arquitectura y el despliegue del sistema, las bases de datos relacionales, y las tecnologías de back-end y front-end empleadas. Cada concepto se presenta en función de su rol dentro del problema de coordinación de evaluaciones, y no como una definición aislada.

2.1. Planificación académica y rendimiento estudiantil

El rendimiento académico en la educación superior está determinado por un amplio rango de factores interconectados, entre los cuales las condiciones bajo las cuales los estudiantes se preparan para sus evaluaciones desempeñan un rol particularmente relevante [Richardson *et al.*, 2012, Vázquez *et al.*, 2012].

2.1.1. Práctica distribuida y carga académica

La *práctica espaciada* (o práctica distribuida) es la estrategia de estudio que consiste en distribuir las sesiones de preparación a lo largo del tiempo, en lugar de concentrarlas inmediatamente antes de una evaluación. Décadas de investigación en ciencias cognitivas han demostrado de manera sistemática que la práctica espaciada produce una retención y comprensión del material significativamente superior a la *práctica masiva*, en la cual el estudio se comprime en un periodo corto justo antes de la evaluación. La síntesis cuantitativa de Cepeda *et al.* abarca 839 mediciones del efecto en 317 experimentos publicados en 184 artículos, lo que sitúa el fenómeno entre los resultados mejor establecidos del área [Cepeda *et al.*, 2006]. El mecanismo detrás del efecto involucra la consolidación de la memoria: la recuperación repetida del material en intervalos espaciados fortalece la retención de manera más efectiva que una única sesión intensiva de estudio [Credé y Kuncel, 2008].

De ese cuerpo de evidencia interesa aquí un hallazgo estructural en particular, y es que el efecto no depende solo de espaciar, sino de cuánto. El intervalo entre sesiones de estudio que maximiza la retención crece a medida que crece el intervalo hasta la evaluación [Cepeda *et al.*, 2006]. La implicancia para la planificación académica es directa: como el material de una asignatura debe retenerse durante todo un semestre, el espaciamiento que lo

favorece se mide en semanas, y es precisamente esa escala la que una evaluación desplazada a última hora destruye. El daño de la concentración de evaluaciones no se limita entonces a las horas de estudio disponibles en la semana afectada, sino que alcanza a la estructura temporal de la preparación completa.

Cuando múltiples evaluaciones se concentran en la misma semana, los estudiantes deben prepararse simultáneamente para varias asignaturas bajo presión de tiempo, lo que genera estrés académico y carga cognitiva elevada. Estos estados afectan negativamente las condiciones bajo las cuales se produce el aprendizaje [Vázquez *et al.*, 2012, Richardson *et al.*, 2012]. Adicionalmente, una distribución adecuada de evaluaciones a lo largo del semestre permite a los estudiantes organizar su tiempo de estudio de manera más autónoma, creando las condiciones que la investigación identifica como óptimas: sesiones de estudio distribuidas en el tiempo en lugar de concentradas en un único periodo de alta presión.

2.1.2. Enfoques existentes y la brecha que dejan

La coordinación de evaluaciones no es un problema inédito, y existen al menos cuatro familias de herramientas y enfoques que lo abordan de forma total o parcial. Compararlas exige fijar primero contra qué se las compara, es decir, qué condiciones impone el problema descrito en el capítulo anterior.

El profesor que programa un certamen es a la vez quien toma la decisión y quien carece de la información sobre la carga ya existente. Una herramienta que entregue esa información a otro actor, o que la entregue una vez fijada la fecha, llega donde ya no se decide, de modo que la exigencia determinante es que la información alcance al profesor en el momento mismo en que elige. Esa información debe estar organizada por programa de estudio, porque la carga que importa es la que enfrenta un estudiante que sigue una malla curricular en un semestre determinado. El contexto impone además dos restricciones sobre las soluciones viables. Las evaluaciones son eventos presenciales que cada profesor decide de manera autónoma, así que su registro depende siempre de una acción deliberada de alguien. Y en la práctica imponer a un profesor las fechas de sus evaluaciones rara vez resulta viable, de modo que el resultado tiende a construirse por acuerdo.

Las aplicaciones de calendario genéricas, como Google Calendar o Microsoft Outlook, permiten registrar eventos y recibir recordatorios, pero los datos los ingresa cada persona para sí misma. No existe una vista institucional compartida ni un mecanismo por el cual un departamento publique un calendario oficial que todos los estudiantes consulten, de modo que no satisfacen ninguna de las cuatro condiciones.

Las plataformas de gestión del aprendizaje (LMS) organizan el material y las actividades de cada asignatura. Moodle, adoptado por la UTFSM bajo el portal `aula.usm.cl`, incluye un calendario que maneja eventos de sitio, categoría, asignatura, grupo y usuario, y que reúne

en una sola vista los eventos de todas las asignaturas en las que un estudiante está inscrito [Moodle Pty Ltd, 2026]. Esa agregación atiende la organización por estudiante solo en apariencia, y deja intacta la exigencia determinante. El calendario se arma del lado de quien padece la sobrecarga, mientras que el profesor que fija la fecha ve únicamente el calendario de su propia asignatura y carece de toda señal sobre las demás. La visibilidad existe, entonces, exactamente donde no sirve para evitar el problema. A esto se suma que los eventos del calendario derivan de actividades de la propia plataforma, como el cierre de un cuestionario o el vencimiento de una entrega, por lo que un certamen presencial solo figura si alguien lo registra a mano. El calendario tampoco advierte cuando dos eventos coinciden, ya que los dibuja y deja que el estudiante lo note. Su alcance, por último, se define por la inscripción en la plataforma, de modo que no puede responder qué carga enfrenta un estudiante de tercer semestre de una carrera determinada.

Los sistemas de información universitarios ocupan el lugar opuesto. SIGA es autoritativo sobre la oferta de asignaturas, la inscripción y los horarios de clases [Universidad Técnica Federico Santa María, 2026], y es la fuente contra la cual cualquier otro sistema debería contrastar sus datos. Las fechas de evaluación, sin embargo, no forman parte de su modelo, y la razón es estructural más que una omisión: los horarios de clases se fijan antes del semestre y se mantienen estables, mientras que las evaluaciones las decide cada profesor una vez comenzado el semestre, con lo cual quedan fuera del ciclo de planificación que estos sistemas administran.

Los sistemas de timetabling académico de código abierto atacan un problema vecino y bien definido: asignar clases o exámenes a bloques horarios y salas de forma automática, a partir de un modelo completo de restricciones. UniTime es un sistema institucional que cubre horarios de clases, exámenes, eventos e inscripción de estudiantes, y está diseñado para que múltiples encargados departamentales coordinen un horario minimizando los conflictos de los estudiantes [UniTime, LLC, 2026]. FET es una aplicación de escritorio que genera el horario completo de una institución a partir de una paleta amplia de restricciones de tiempo y espacio [Lalescu, 2026]. Ambos presuponen las dos condiciones que el DI precisamente no cumple. Presuponen un modelo completo del problema, es decir, la totalidad de las asignaturas, salas, profesores y restricciones cargadas por adelantado, cuando en el DI las fechas ni siquiera existen registradas en alguna parte. Y presuponen una autoridad capaz de imponer el horario que el solver produce, cuando aquí las fechas las decide cada profesor. Adoptar cualquiera de los dos supondría, además de una adaptación considerable, cambiar el proceso institucional para que se parezca al que la herramienta espera.

La literatura sobre programación de exámenes confirma ese encuadre. La revisión de Qu et al. trata el timetabling de exámenes como una actividad administrativa centralizada y recorre dos décadas de métodos de búsqueda para resolverla [Qu et al., 2009]. Los trabajos aplicados recientes siguen esa línea: el caso de la Universidad de Bucknell aborda la asignación de exámenes finales a bloques, respetando conflictos de estudiantes y capacidad de salas, sobre un período de exámenes centralizado [Chaplin et al., 2025]. El trabajo más cercano al problema de esta memoria es el de Mallari et al., que formula la sincronización de los calen-

darios de evaluaciones de un término académico como un modelo de programación lineal entera mixta, con la carga máxima del estudiante entre sus criterios de calidad, y lo aplica a 236 clases de una universidad [Mallari *et al.*, 2023]. Ese trabajo demuestra que el problema es tratable cuando se dispone de datos completos y de la atribución para fijar el calendario resultante, que son justamente las dos condiciones ausentes en el DI. La distancia entre lo que la investigación produce y lo que las instituciones efectivamente adoptan está a su vez documentada: McCollum señala que las técnicas publicadas rara vez sobreviven a las complejidades de una implantación real [McCollum, 2007], y la revisión sistemática de Oude Vrielink *et al.* se propone medir en qué grado las instituciones de educación superior incorporan los desarrollos de timetabling, motivada por esa misma brecha [Oude Vrielink *et al.*, 2019].

El resultado habitual de esta brecha es que los estudiantes reconstruyen su propio calendario de evaluaciones a mano, a partir de los programas de las asignaturas y de los anuncios en clases, y que ese calendario resulta invisible para los profesores cuando fijan de forma independiente las fechas de sus propias evaluaciones. En conjunto, los enfoques disponibles habilitan solo una respuesta *reactiva*, en la cual los topes se descubren y se intentan resolver con el semestre ya en curso. Lo deseable es una respuesta *proactiva*, en la que los profesores registren sus evaluaciones y comprueben la ausencia de topes al inicio del semestre, antes de que la sobrecarga alcance a los estudiantes.

La tabla 1 resume la comparación. La brecha que queda abierta define el sistema propuesto: hace falta que la carga acumulada sobre una malla curricular sea visible para quien fija la fecha y en el momento en que la fija, sin exigir el modelo completo ni la autoridad central que un solver necesitaría. Esa lectura explica también por qué el sistema informa la sobrecarga en lugar de impedirla, decisión que se retoma en el capítulo 3: donde no hay atribución para imponer una fecha, el aporte posible es dar la información a tiempo a quien sí decide.

Las herramientas descritas, el LMS institucional, los sistemas de información y los calendarios genéricos, comparten una característica: son aplicaciones web accesibles desde el navegador. La solución propuesta se inscribe en esa misma categoría, por lo que las secciones siguientes desarrollan los conceptos de ingeniería de software que sustentan una aplicación web y su arquitectura.

2.2. Aplicaciones web

2.2.1. Aplicación web

Una *aplicación web* es un sistema de software cuya interfaz de usuario es entregada a través de un navegador web estándar, con la lógica de la aplicación y los datos residiendo en un servidor remoto. A diferencia del software de escritorio nativo, una aplicación web no requiere instalación en el dispositivo del usuario y es accesible desde cualquier máquina que cuente con un navegador y conexión a la red.

Enfoque	Qué cubre	Qué le falta
Calendarios genéricos (Google Calendar, Outlook)	Registro individual de eventos y recordatorios.	Vista institucional compartida; cada persona ingresa sus propios datos.
LMS (Moodle, aula.usm.cl)	Calendario por asignatura, agregado para cada estudiante según su inscripción.	Visibilidad para quien fija la fecha; detección de coincidencias; alcance por malla curricular; los eventos derivan de actividades de la plataforma.
Sistemas de información universitarios (SIGA)	Oferta, inscripción y horarios de clases, con datos autoritativos.	Las fechas de evaluación no forman parte del modelo, porque se deciden una vez iniciado el semestre.
Solvers de timetabling (UniTime, FET)	Asignación automática de clases y exámenes a bloques horarios y salas.	Requieren un modelo completo de restricciones y una autoridad que imponga el resultado.
Experiencias publicadas (Bucknell, Mallari et al.)	Optimización de calendarios de exámenes y evaluaciones, con límites de carga del estudiante.	Suponen un período centralizado y datos completos de todas las asignaturas.

Tabla 1: Enfoques existentes frente a las condiciones del problema de coordinación.

Fuente: Elaboración propia.

2.2.2. Arquitectura cliente-servidor

La *arquitectura cliente-servidor* es un modelo en el cual dos roles diferenciados se comunican a través de una red: el *cliente* (típicamente el navegador) inicia las solicitudes, y el *servidor* las procesa y retorna respuestas. En la variante de *tres capas*, el sistema se divide en una capa de presentación (la interfaz en el navegador), una capa de aplicación (la lógica de negocio en el servidor) y una capa de datos (la base de datos), cada una con responsabilidades e interfaces claramente definidas.

2.2.3. Single Page Application (SPA)

Una *Single Page Application* (SPA) es una arquitectura de front-end web en la cual el navegador carga un único documento HTML en la primera visita, y toda la navegación posterior es gestionada por JavaScript en el navegador en lugar de cargar nuevas páginas desde el servidor. A medida que el usuario se desplaza entre vistas, el framework JavaScript actualiza únicamente las partes relevantes de la página sin una recarga completa, resultando en una experiencia más fluida.

2.2.4. API REST

Una *API REST* (Representational State Transfer) es un estilo de diseño para APIs web en el cual el servidor expone un conjunto de endpoints, cada uno representando un recurso del dominio, y los clientes interactúan con estos recursos mediante los verbos HTTP estándar: GET para consultar, POST para crear, PUT y PATCH para actualizar (reemplazando el recurso completo o solo parte de él, respectivamente) y DELETE para eliminar. Este conjunto de operaciones se conoce comúnmente como CRUD (Create, Read, Update, Delete), las cuatro acciones básicas de persistencia de datos. Las respuestas se formatean en JSON (JavaScript Object Notation), un formato de texto ligero e independiente del lenguaje de programación que tanto los navegadores como el código del servidor pueden interpretar directamente.

Dos propiedades del estilo tienen consecuencias de ingeniería que conviene enunciar desde ya. La primera es la *idempotencia*: una operación es idempotente cuando ejecutarla varias veces deja el sistema en el mismo estado que ejecutarla una sola vez. GET, PUT y DELETE lo son por definición, mientras que POST no lo es, y de ahí que reenviar un formulario pueda crear dos registros donde se esperaba uno. La distinción importa porque las redes fallan de maneras ambiguas: un cliente que no recibe respuesta no sabe si la solicitud se procesó, y solo puede reintentar sin riesgo aquellas operaciones que sean idempotentes. La segunda es que la comunicación es *sin estado*, es decir, cada solicitud transporta toda la información necesaria para ser atendida y el servidor no conserva memoria de las anteriores. Esto simplifica el servidor y permite atender solicitudes en cualquier orden, a cambio de que la identidad del usuario deba viajar en cada solicitud, requisito que motiva el esquema de autenticación descrito más adelante.

2.2.5. CORS

CORS (Cross-Origin Resource Sharing) es un mecanismo de seguridad implementado por los navegadores web que controla qué orígenes externos tienen permitido realizar solicitudes HTTP a un servidor determinado. Por defecto, los navegadores bloquean las solicitudes realizadas desde un origen distinto al del servidor (política de mismo origen). CORS permite a los servidores indicar explícitamente, mediante encabezados HTTP, qué orígenes están autorizados a comunicarse con ellos.

2.3. Arquitectura del sistema

Definida la aplicación web y su modelo cliente-servidor, resta decidir cómo se estructura, empaqueta y despliega el lado del servidor. Esas decisiones sobre cómo se organiza el código del back-end, cómo se aísla de su entorno de ejecución y cómo se expone al exterior constituyen la arquitectura del sistema. Los conceptos siguientes fundamentan las elecciones de arquitectura y despliegue que se detallan en el capítulo 3.

2.3.1. Arquitectura monolítica

Una *arquitectura monolítica* es aquella en la cual toda la aplicación back-end es construida, compilada y desplegada como una única unidad. Esto contrasta con la arquitectura de *microservicios*, en la cual el sistema se divide en múltiples servicios desplegables de forma independiente que se comunican a través de una red. Las arquitecturas monolíticas son más simples de desarrollar y operar: existe una única base de código, un único artefacto de despliegue y no hay coordinación de red entre servicios que gestionar.

2.3.2. Contenedores y Docker

La *contenedorización* es la práctica de empaquetar una aplicación y todas sus dependencias en una unidad estandarizada y aislada llamada *contenedor*, que se ejecuta del mismo modo en distintos entornos de host. *Docker* es una plataforma abierta para desarrollar, distribuir y ejecutar aplicaciones, y es la plataforma de contenedorización más ampliamente adoptada [Docker Inc., 2026]. *Docker Compose* la extiende, permitiendo definir y orquestar múltiples contenedores en un único archivo de configuración, gestionando su orden de arranque y las dependencias entre servicios.

2.3.3. Nginx

Nginx es un servidor web de alto rendimiento utilizado frecuentemente como *proxy inverso* [F5, Inc., 2026]. Un proxy inverso es un servidor que se sitúa frente a uno o más servicios back-end, recibiendo todas las solicitudes entrantes y redirigiéndolas al destino apropiado según la ruta solicitada. Además del enrutamiento, *Nginx* puede encargarse de la terminación TLS: el protocolo TLS (Transport Layer Security) cifra las comunicaciones entre el cliente y el servidor, y la terminación TLS consiste en descifrar estas conexiones HTTPS en el punto de entrada antes de reenviar el tráfico de forma interna.

2.4. Bases de datos relacionales

2.4.1. Modelo relacional

Una *base de datos relacional* organiza los datos en tablas de filas y columnas, donde cada tabla representa una entidad específica del dominio y las relaciones entre tablas se expresan mediante *claves foráneas* (FK): referencias desde una columna de una tabla hacia la clave primaria de otra. El estándar para diseñar y visualizar estas relaciones antes de implementarlas es el *diagrama Entidad-Relación* (ER), que representa gráficamente las entidades del domi-

nio, sus atributos y sus relaciones. El modelo relacional garantiza la integridad de los datos mediante restricciones (NOT NULL, UNIQUE, claves foráneas) y soporta las propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad) que garantizan el procesamiento confiable de transacciones incluso ante escrituras concurrentes o fallos del sistema. El lenguaje estándar para consultar y manipular bases de datos relacionales es SQL (Structured Query Language).

2.4.2. PostgreSQL

PostgreSQL es un sistema gestor de bases de datos objeto-relacional de código abierto, reconocido por su cumplimiento de estándares, robustez y riqueza de funcionalidades [PostgreSQL Global Development Group, 2026]. Soporta tipos de datos avanzados, incluyendo timestamps con zona horaria, identificadores únicos universales (UUID) y texto con insensibilidad a mayúsculas y minúsculas, así como extensiones criptográficas para la generación segura de identificadores.

2.4.3. Migraciones de base de datos

Una *migración de base de datos* es un cambio versionado y reproducible al esquema de una base de datos: creación de tablas, adición de columnas, modificación de restricciones, entre otros. Las migraciones se almacenan como archivos ordenados cronológicamente, cada uno con una versión de subida (*up*) que aplica el cambio y una versión de bajada (*down*) que lo revierte. Este mecanismo permite que el esquema de la base de datos evolucione de manera controlada junto al código fuente, garantizando que cualquier entorno pueda alcanzar el estado correcto del esquema de forma reproducible.

2.5. Back-end

El *back-end* es la parte de la aplicación que se ejecuta en el servidor y concentra la lógica de negocio, la autenticación y autorización de los usuarios, la validación de las reglas del dominio y el acceso a la base de datos. Es responsable de garantizar que ninguna operación viole las restricciones del sistema, por ejemplo que no se registren evaluaciones con tope, con independencia de lo que solicite el cliente. Los conceptos y tecnologías que se definen a continuación son los que efectivamente se utilizan en el back-end del sistema, y su elección se justifica en el capítulo 3.

2.5.1. Go

Go (también conocido como Golang) es un lenguaje de programación compilado y de tipado estático desarrollado por Google y publicado como código abierto en 2009 [Google, 2026]. Go compila a un único binario autónomo sin dependencias de runtime externas. Su modelo de concurrencia basado en *goroutines* permite atender múltiples solicitudes simultáneas con bajo consumo de memoria, sin necesidad de gestionar hilos del sistema operativo directamente. El lenguaje exige el manejo explícito de errores en cada sitio de llamada, lo que hace que el flujo de control de la aplicación sea predecible y auditable.

2.5.2. Autenticación con JWT y cookies httpOnly

Un *JSON Web Token* (JWT) es un formato de token compacto y firmado criptográficamente, utilizado para representar afirmaciones de autenticación entre dos partes [Jones *et al.*, 2015]. Un JWT contiene tres partes codificadas: un encabezado (que especifica el algoritmo de firma), un payload (con los datos del usuario, roles y tiempo de expiración) y una firma digital. El servidor puede verificar la validez del token en cualquier solicitud posterior sin necesidad de consultar la base de datos para recuperar el estado de la sesión.

Una *cookie httpOnly* es una cookie de navegador a la que JavaScript no puede acceder mediante `document.cookie`. Esta restricción protege el token de sesión ante ataques de tipo XSS (Cross-Site Scripting), en los cuales un atacante logra inyectar y ejecutar scripts maliciosos en el contexto de la página web de la víctima. Al almacenar el token en una cookie `httpOnly` en lugar de en `localStorage`, incluso si un script malicioso se ejecuta en la página, no puede leer ni exfiltrar el token de autenticación.

2.5.3. HMAC

HMAC (Hash-based Message Authentication Code) es un mecanismo criptográfico que combina una función de hash con una clave secreta para producir un código de autenticación de mensaje [Krawczyk *et al.*, 1997]. A diferencia de un hash simple, HMAC garantiza tanto la *integridad* del mensaje como su *autenticidad*. En el contexto de JWT, la variante HMAC-SHA256 se utiliza para firmar el token, garantizando que su contenido no puede ser modificado sin invalidar la firma.

2.5.4. ORM

Un ORM (Object-Relational Mapper) es una herramienta que mapea las filas de las tablas de una base de datos relacional a objetos o estructuras del lenguaje de programación utilizado,

y viceversa. Permite a la aplicación interactuar con la base de datos mediante llamadas a funciones tipadas en lugar de construir cadenas SQL manualmente. Las consultas generadas por un ORM son parametrizadas de forma automática, lo que cierra la vía de inyección SQL en las consultas que produce.

2.5.5. **bcrypt**

bcrypt es un algoritmo de hash diseñado específicamente para el almacenamiento seguro de contraseñas [Provos y Mazières, 1999]. A diferencia de las funciones de hash de propósito general como SHA-256, *bcrypt* es intencionalmente lento, con un costo computacional configurable que puede incrementarse a medida que el hardware avanza. Esta característica encarece drásticamente los ataques de fuerza bruta o por diccionario incluso si los valores hash almacenados quedan expuestos.

2.6. **Front-end**

El *front-end* es la parte de la aplicación que se ejecuta en el navegador del usuario y es responsable de la interfaz: presentar la información que entrega el back-end, capturar las acciones del usuario y comunicarla al servidor. En este sistema, es la capa que materializa la vista unificada del calendario, la detección visual de conflictos y los formularios de carga y gestión de evaluaciones. Los conceptos y tecnologías que se definen a continuación son los que efectivamente se utilizan en el front-end del sistema, y su elección se justifica en el capítulo 3.

2.6.1. **Vue 3**

Vue 3 es un framework de JavaScript para la construcción de interfaces de usuario, que se construye sobre HTML, CSS y JavaScript estándar y provee un modelo de programación declarativo y basado en componentes [You y Vue.js Contributors, 2026]. *Vue 3* introdujo la *Composition API*, que organiza la lógica de los componentes en funciones composables reutilizables en lugar de distribuirla en un objeto de opciones fijo, facilitando el mantenimiento y la reutilización de lógica compleja entre componentes.

2.6.2. **TypeScript**

TypeScript es un superconjunto de JavaScript desarrollado por Microsoft que agrega anotaciones de tipos estáticos opcionales compiladas antes de la ejecución [Microsoft, 2026]. El

compilador de TypeScript verifica la consistencia de tipos en toda la base de código y reporta errores antes de que el código se ejecute, capturando en tiempo de compilación errores como el acceso a propiedades inexistentes o el paso de tipos incorrectos a funciones.

2.6.3. Vite

Vite es una herramienta de construcción moderna para el front-end [You y Vite Contributors, 2026]. Para producción, genera un bundle optimizado y minificado de los archivos del proyecto, listo para ser servido como archivos estáticos. Durante el desarrollo, sirve los archivos fuente directamente como módulos ES nativos del navegador sin un proceso de empaquetado previo, lo que resulta en un servidor que inicia de forma casi instantánea y aplica cambios en el código sin recargar la página completa.

2.6.4. PrimeVue

Una *librería de componentes* es una colección de componentes de interfaz de usuario prediseñados, estilizados y accesibles que un equipo adopta para evitar implementar elementos comunes de la interfaz desde cero. *PrimeVue 4* es una librería de componentes para Vue 3 con un catálogo amplio de componentes listos para producción y un mecanismo de temas unificado [PrimeTek Informatics, 2026].

2.6.5. vue-router

vue-router 4 es la solución de routing estándar y oficial para Vue 3 [You y Vue Router Contributors, 2026]. Mapea rutas URL a componentes de página y habilita la navegación dentro de la SPA sin recargas completas del navegador. Permite definir *navigation guards*: funciones que se ejecutan antes de entrar a una ruta y pueden redirigir al usuario si una condición no se cumple, como no estar autenticado o no poseer los privilegios necesarios para acceder a esa sección.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

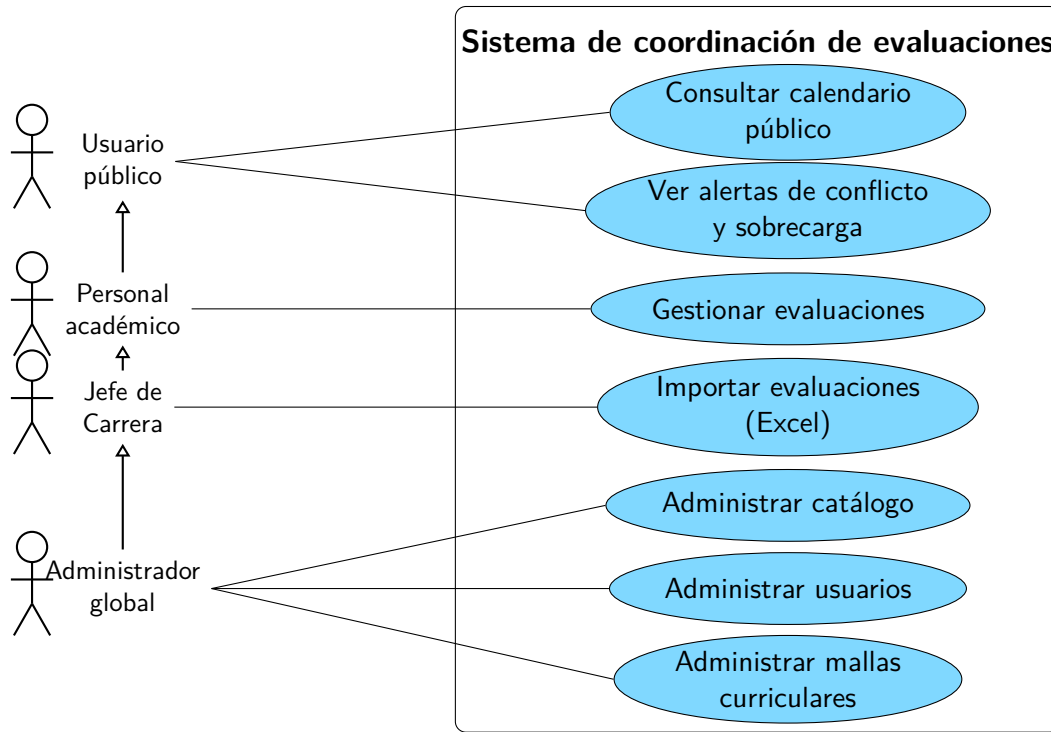
El sistema desarrollado es una aplicación web destinada a centralizar y coordinar la programación de evaluaciones académicas en el Departamento de Informática de la UTFSM, respondiendo directamente a los problemas identificados en el capítulo anterior. La vista global del calendario con detección automática de conflictos aborda la falta de visibilidad entre asignaturas y las semanas de carga crítica. El registro persistente de todas las evaluaciones resuelve además la ausencia de un historial académico. Finalmente, la disponibilidad de información centralizada en tiempo real permite a los jefes de carrera intervenir de manera proactiva ante situaciones de alta carga.

3.1. Actores y casos de uso

El sistema reconoce cuatro actores, organizados de forma jerárquica según su alcance sobre los datos. Los actores forman una relación de generalización: cada actor hereda las capacidades del anterior y añade las propias, de modo que un administrador global puede realizar todo lo que realiza un jefe de carrera, y así sucesivamente hasta el usuario público, que constituye la base.

- *Usuario público (estudiante)* - actor central del sistema. Comprende a cualquier persona que consulta el calendario de evaluaciones sin una cuenta: principalmente los estudiantes, para quienes el sistema se diseña, pero también profesores u otros interesados que no disponen de acceso. Filtra el calendario publicado por malla y semestre curricular y visualiza las alertas de conflicto, sin necesidad de iniciar sesión.
- *Personal académico* (profesores y ayudantes) - además de lo anterior, crea, edita y elimina las evaluaciones de los paralelos que tiene a su cargo.
- *Jefe de Carrera* - además, importa evaluaciones de forma masiva y gestiona las evaluaciones de los departamentos que le han sido asignados. Es el rol administrativo de alcance intermedio, típicamente en manos del jefe de carrera correspondiente.
- *Administrador global* - acceso completo a todos los datos y funciones de configuración: administra el catálogo académico, los usuarios y los programas de estudio.

La figura 2 resume los casos de uso principales del sistema y su asociación con cada actor.



La flecha ▷ indica que el actor hereda los casos de uso del actor superior.

Figura 2: Diagrama de casos de uso del sistema. Los actores se relacionan por generalización: cada uno hereda los casos de uso del actor del que desciende.

Fuente: Elaboración propia.

3.2. Requerimientos del sistema

A partir de los problemas identificados en el capítulo 1 y de las necesidades de los actores, se definieron los requerimientos que el sistema debe satisfacer. Los requerimientos funcionales (RF) describen las capacidades que el sistema debe ofrecer; los requerimientos no funcionales (RNF) describen las cualidades que debe cumplir. Las funcionalidades implementadas, descritas en el resto de este capítulo, se derivan directamente de estos requerimientos.

ID	Requerimiento funcional
RF-1	Autenticar y cerrar la sesión de los usuarios registrados de forma segura.
RF-2	Administrar (CRUD) el catálogo académico: departamentos, códigos, asignaturas, semestres, ofertas de asignatura (una asignatura dictada en un semestre) y paralelos.
RF-3	Administrar usuarios y asignarles la administración de uno o más departamentos.
RF-4	Definir programas de estudio y su malla curricular (asignaturas por semestre curricular).
RF-5	Crear, editar y eliminar (borrado suave) evaluaciones individuales sobre las ofertas y paralelos autorizados.
RF-6	Importar evaluaciones de forma masiva desde un archivo Excel, con revisión asistida y validación previa a la confirmación.
RF-7	Exponer, sin requerir autenticación, un calendario de evaluaciones filtrable por semestre, programa de estudio, semestre curricular y asignatura.
RF-8	Detectar y clasificar por severidad (misma semana, mismo día, tope de horario) las semanas de alta carga sobre el conjunto de asignaturas seleccionadas, y alertar visualmente.
RF-9	Presentar el calendario en vistas mensual y semanal con el detalle de cada evaluación.
RF-10	Persistir las evaluaciones de cada semestre, conformando un registro histórico consultable.

Tabla 2: Requerimientos funcionales del sistema.

Fuente: Elaboración propia.

ID	Requerimiento no funcional
RNF-1	Usabilidad: el sistema debe poder ser utilizado por personal sin formación técnica con una capacitación mínima.
RNF-2	Rendimiento y concurrencia: el sistema debe atender accesos concurrentes al calendario sin degradación significativa del tiempo de respuesta.
RNF-3	Seguridad: el sistema debe proteger las credenciales, las sesiones y los datos frente a los ataques habituales de aplicaciones web.
RNF-4	Reproducibilidad del despliegue: el sistema debe poder desplegarse de forma idéntica y reproducible en distintos entornos.
RNF-5	Mantenibilidad: la base de código debe ser modular, comprensible y extensible.
RNF-6	Integridad de datos: el sistema no debe permitir estados inconsistentes ni la pérdida accidental de datos.
RNF-7	Accesibilidad pública: la vista de calendario orientada a estudiantes debe ser accesible sin autenticación.

Tabla 3: Requerimientos no funcionales del sistema.

Fuente: Elaboración propia.

3.3. Trazabilidad

La tabla 4 vincula cada problema identificado en el capítulo 1 con los requerimientos que lo abordan y la funcionalidad que lo resuelve. Esta trazabilidad garantiza que cada problema

tenga una respuesta concreta y que ninguna funcionalidad exista sin un problema que la justifique. La validación de que cada requerimiento se cumple efectivamente se presenta en el capítulo 4.

Problema (Cap. 1)	Requerimientos	Funcionalidad que lo resuelve
Falta de visibilidad entre asignaturas	RF-7, RF-8, RF-9	Calendario público unificado con filtros, vistas y alertas
Coordinación manual inviable a escala	RF-6, RF-7	Carga masiva e individual centralizada y vista global compartida
Semanas de carga crítica	RF-8	Detección de sobrecarga clasificada por severidad
Falta de registro histórico	RF-5, RF-6, RF-10	Persistencia de todas las evaluaciones por semestre
Impacto en rendimiento y bienestar	RF-8, RNF-1	Alertas tempranas de carga e interfaz usable
Intervención reactiva de jefes de carrera	RF-7, RF-8	Visibilidad compartida para actuar antes de iniciado el semestre

Tabla 4: Matriz de trazabilidad: problema → requerimiento → funcionalidad.

Fuente: Elaboración propia.

3.4. Arquitectura general

El sistema sigue una arquitectura en capas desplegada sobre contenedores Docker. La figura 3 resume esta organización antes de entrar en el detalle de cada capa en las secciones siguientes.

El sistema se organiza en cuatro capas. La capa de *cliente* es una aplicación de página única (SPA) construida con Vue 3 y PrimeVue que se ejecuta en el navegador del usuario. La capa de *borde* es un proxy inverso Nginx que constituye el único punto de entrada al sistema: termina el cifrado TLS de las conexiones HTTPS entrantes, sirve directamente los archivos estáticos de la SPA, enruta las solicitudes `/api/` hacia la capa de aplicación y aplica un límite de tasa por dirección IP. La capa de *aplicación* es el back-end en Go, que concentra la lógica de negocio y expone la API REST. La capa de *datos* es una base de datos PostgreSQL que persiste toda la información del sistema.

La comunicación entre capas emplea protocolos distintos según su grado de exposición. El tráfico entre el navegador y Nginx viaja cifrado sobre HTTPS a través del puerto 443. En cambio, la comunicación entre Nginx y el back-end, y entre el back-end y la base de datos, ocurre en texto plano dentro de la red privada de Docker, a la que ningún cliente externo tiene acceso: el back-end escucha en el puerto interno 8080 y PostgreSQL en el 5432. Esta separación permite concentrar la gestión de certificados y del cifrado TLS en un único punto, sin exponer los servicios internos a la red pública.

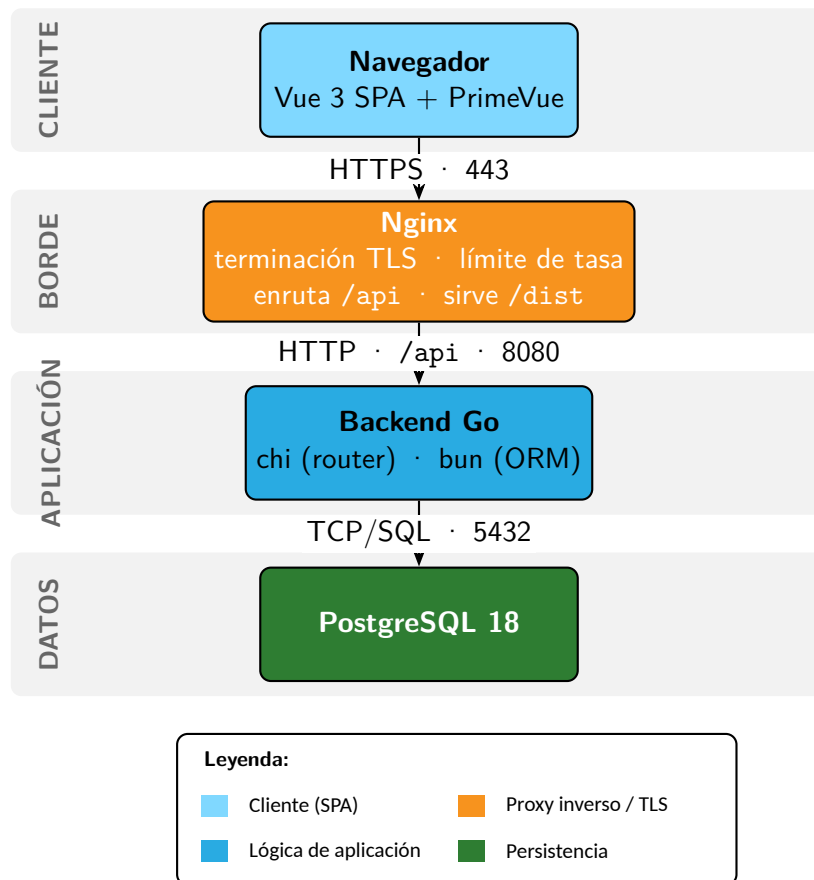


Figura 3: Arquitectura general del sistema en capas, con los protocolos y puertos de comunicación entre ellas.

Fuente: Elaboración propia.

3.5. Diseño de base de datos

El esquema de la base de datos está compuesto por doce tablas que modelan el dominio académico del sistema. Todas las claves primarias son identificadores únicos universales (UUID v4), generados por la función nativa `uuidv4()` de PostgreSQL 18; las únicas extensiones requeridas son `citext`, empleada para el almacenamiento de correos electrónicos con insensibilidad a mayúsculas y minúsculas, y `btree_gist`, que habilita la restricción de exclusión sobre las evaluaciones descrita más adelante. Todos los campos de fecha y hora son de tipo `TIMESTAMPTZ`, garantizando el almacenamiento con zona horaria explícita.

El modelo se organiza en cuatro dominios, que en los diagramas se distinguen por color. El dominio de *usuarios y permisos* agrupa las cuentas del sistema y los privilegios administrativos sobre departamentos. El *catálogo académico* reúne las entidades estables del dominio universitario: departamentos, asignaturas y semestres. El dominio de *oferta académica y evaluaciones* modela la instanciación de ese catálogo en un semestre concreto, junto con las evaluaciones asociadas; su pieza central es la *oferta de asignatura* (`course_offering`), un concepto propio del sistema que representa una asignatura del catálogo dictada en un semestre específico, y sobre la cual se definen los paralelos y las evaluaciones. Por último, el dominio de *programas y malla curricular* describe los programas de estudio y la posición de cada asignatura dentro de su malla. A estos cuatro dominios se suma la tabla de *staging exam_imports*, tratada por separado debido a su naturaleza transitoria.

La figura 4 presenta el modelo entidad-relación completo del esquema, con los atributos de cada tabla. Las líneas representan relaciones por clave foránea anotadas con su cardinalidad (1-N, y 0..1-N cuando la clave foránea es opcional o *nullable*), y el color agrupa las entidades por dominio: verde para usuarios y permisos, azul para el catálogo académico, rosado para la oferta académica y las evaluaciones, y amarillo para los programas y su malla curricular. La clave primaria aparece subrayada, las claves foráneas se marcan con (FK) y los atributos con restricción UNIQUE de una sola columna con (U); por claridad se omiten las columnas de auditoría `created_at` y `updated_at`, presentes en la mayoría de las tablas. La tabla de *staging exam_imports* se describe por separado en la sección de importación masiva, por tratarse de una estructura desnormalizada ajena al núcleo del modelo.

A continuación se describe cada tabla, su propósito y sus atributos principales, agrupadas por dominio.

3.5.1. Usuarios y permisos

users - Almacena todos los usuarios del sistema, independientemente de su rol. Los campos principales son `full_name` e `email`, almacenado como `CITEXT` para garantizar insensibilidad a mayúsculas y minúsculas. La contraseña se guarda únicamente como hash `bcrypt` en el campo `password_hash`, el texto plano nunca se persiste por buenas prácticas de seguridad. Dos banderas booleanas determinan los privilegios especiales del usuario:

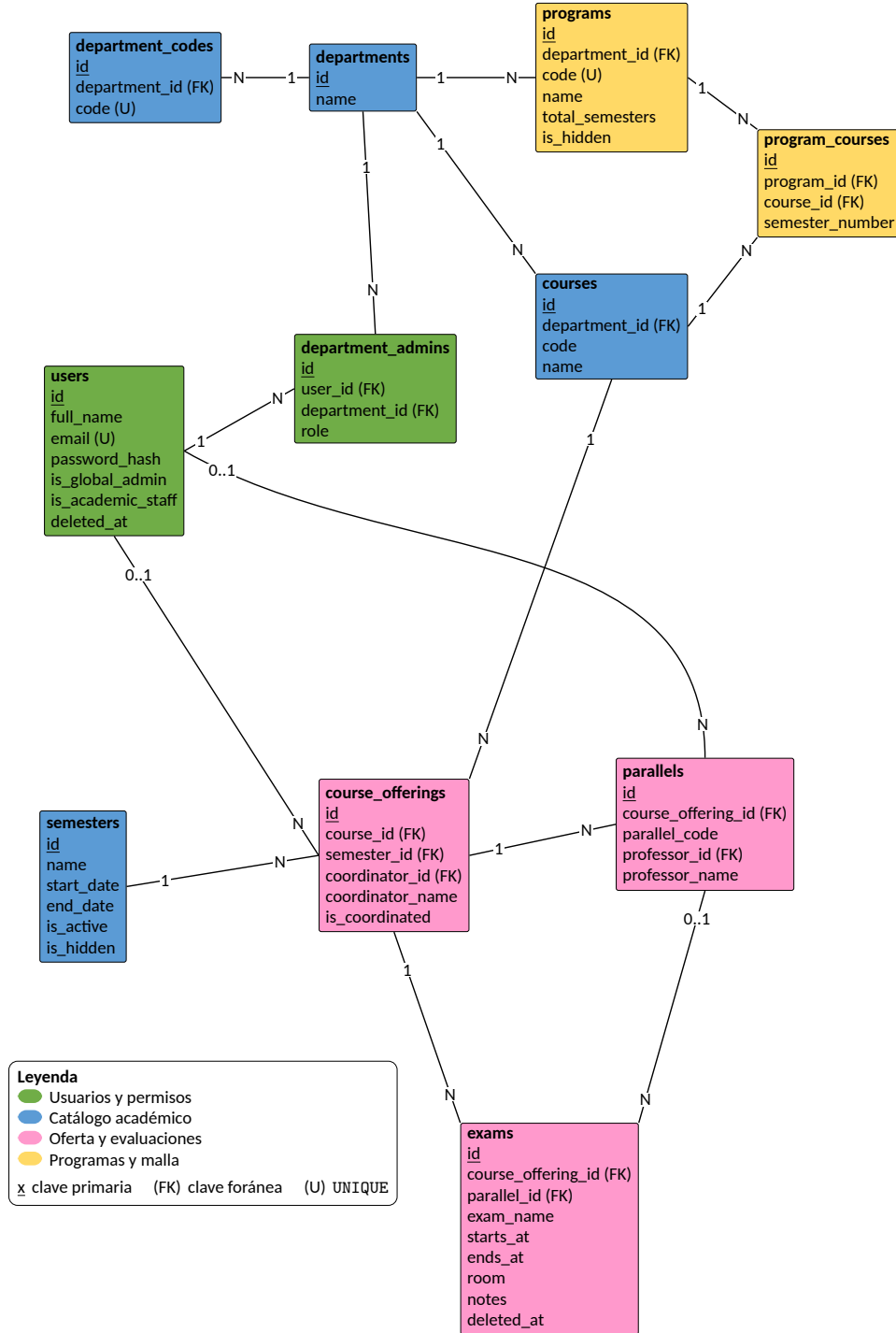


Figura 4: Modelo entidad-relación completo del esquema, con los atributos de cada tabla. Restricciones UNIQUE compuestas (no representables en la notación de columna): courses(department_id, code), course_offerings(course_id, semester_id), parallels(course_offering_id, parallel_code), department_admins(user_id, department_id) y program_courses(program_id, course_id, semester_number).

Fuente: Elaboración propia.

`is_global_admin` e `is_academic_staff` (para profesores y ayudantes). La eliminación de usuarios se implementa mediante borrado suave a través del campo `deleted_at`: un valor nulo indica que el usuario está activo mientras que un timestamp indica que fue eliminado. Esta tabla centraliza la identidad de todos los actores del sistema y los permisos específicos sobre departamentos se derivan de las entradas en `department_admins`.

department_admins - Tabla de unión que otorga a un usuario derechos administrativos sobre un departamento específico. Sus columnas principales son `user_id` (FK a `users`) y `department_id` (FK a `departments`), con una restricción `UNIQUE` sobre su combinación para evitar asignaciones duplicadas. Un mismo usuario puede administrar múltiples departamentos donde cada fila representa uno de esos permisos de forma independiente, lo que permite agregar o revocar accesos por departamento sin modificar el registro principal del usuario.

3.5.2. Catálogo académico

departments - Representa los departamentos académicos de la universidad (por ejemplo, Informática, Matemáticas, Física). Es la entidad raíz del catálogo: contiene únicamente `id` y `name`, y a partir de ella se organizan asignaturas, códigos y usuarios administrativos.

department_codes - Almacena los códigos cortos asociados a cada departamento (por ejemplo, INF, MAT, FIS). Estos códigos se usan como prefijo en los códigos de las asignaturas y permiten identificar el departamento de origen de una asignatura a partir de su código. Un departamento puede tener múltiples códigos para cubrir variaciones históricas o códigos alternativos.

courses - Representa las asignaturas del catálogo académico con independencia del semestre en que se ofrezcan. Sus atributos principales son `code` (por ejemplo, INF67) y `name` (por ejemplo, *Algoritmos y Complejidad*), ambos asociados a un departamento mediante `department_id`. La combinación (`department_id`, `code`) tiene restricción `UNIQUE`, garantizando que no existan dos asignaturas con el mismo código dentro de un mismo departamento.

semesters - Representa los semestres académicos, con su nombre (por ejemplo, 2025–1), fechas de inicio y término (`start_date`, `end_date`), y dos banderas: `is_active`, que marca el semestre mostrado por defecto al cargar el calendario, e `is_hidden`, que oculta semestres en configuración del calendario público. Las fechas de inicio y término se utilizan para restringir que las evaluaciones solo puedan programarse dentro de ese rango temporal.

3.5.3. Oferta académica y evaluaciones

course_offerings - Modela la oferta de una asignatura en un semestre específico. Contiene `course_id` y `semester_id` como claves foráneas, con restricción `UNIQUE` sobre su combinación para evitar duplicados. El campo booleano `is_coordinated` indica si la oferta corresponde a una asignatura coordinada, que puede tener evaluaciones compartidas entre todos sus paralelos. Los campos `coordinator_id` y `coordinator_name` almacenan el responsable de la coordinación. Ambos son opcionales: `coordinator_id` es una FK nullable a `users`, ya que el coordinador puede no tener una cuenta en el sistema. En ese caso, `coordinator_name` permite registrar igualmente su nombre para efectos de visualización.

parallels - Representa cada paralelo de una oferta de asignatura. Se asocia a una `course_offering` mediante `course_offering_id` y se identifica dentro de ella por `parallel_code`. El campo `professor_id` es una FK nullable a `users`: un paralelo puede existir sin que el profesor tenga una cuenta en el sistema. El campo `professor_name` permite registrar el nombre del profesor en ese caso, garantizando que la información de quién dicta la asignatura esté siempre disponible independientemente de si existe un usuario asociado.

exams - Almacena las evaluaciones del sistema y constituye la tabla central del sistema. Cada evaluación se asocia a una oferta de asignatura (`course_offering_id`) y opcionalmente a un paralelo específico mediante `parallel_id`. Cuando `parallel_id` es `NULL`, la evaluación es coordinada, es decir, compartida por todos los paralelos de esa oferta. Cuando tiene un valor, la evaluación pertenece únicamente a ese paralelo. El campo `exam_name` almacena el nombre de la evaluación (por ejemplo, *Certamen 1*). Los campos `starts_at` y `ends_at` almacenan la fecha y hora de inicio y término con zona horaria; además, el sistema impide que un mismo paralelo tenga dos evaluaciones programadas en el mismo horario, para evitar errores de entrada de datos. El campo `room` permite registrar la sala asignada, y `notes` permite agregar observaciones adicionales. El campo `deleted_at` implementa borrado suave: un valor nulo indica que la evaluación está activa, mientras que un *timestamp* indica que fue eliminada. Las evaluaciones eliminadas no desaparecen de la base de datos y pueden ser restauradas.

3.5.4. Importación masiva

exam_imports - Tabla de staging utilizada en el flujo de importación masiva desde archivos Excel. Cada fila representa una evaluación tal como fue leída del archivo, preservando los valores originales en campos `raw_*` (por ejemplo, `raw_course_name`, `raw_exam_start`, `raw_parallel`) para facilitar la depuración y la corrección manual posterior. Junto a estos valores crudos, la tabla almacena las referencias resueltas a las entidades del catálogo (`course_id`, `parallel_id`, `professor_user_id`, etc.) una vez que el algoritmo de matching las identifica. El campo `status` registra el estado de procesamiento de cada fila, con

los valores posibles pending, ready, review, blocked, imported y discarded. Las filas pertenecientes a una misma carga se agrupan mediante upload_id, lo que permite gestionar y descartar lotes completos de forma atómica. Los campos de auditoría (created_by, committed_by, status_updated_by) registran qué usuario realizó cada acción sobre las filas. Esta tabla actúa como buffer de revisión humana antes de escribir datos definitivos en las tablas de producción.

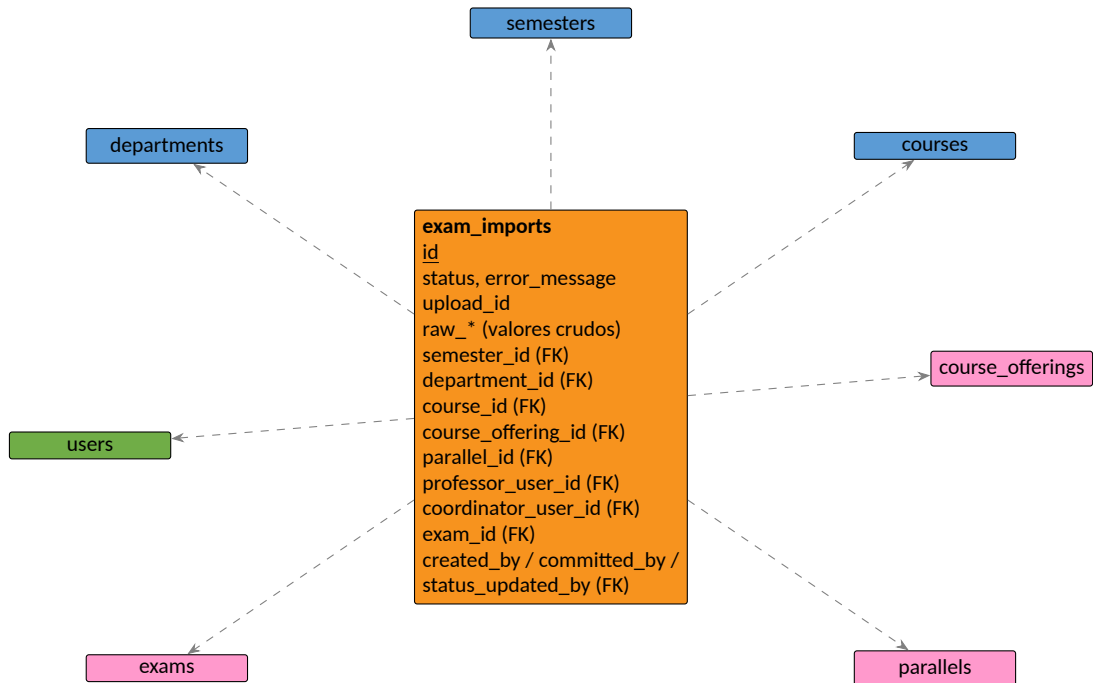


Figura 5: Tabla de *staging* exam_imports y sus referencias resueltas hacia el núcleo del modelo. Las líneas discontinuas son claves foráneas *nullable* que el algoritmo de *matching* completa progresivamente; ninguna es obligatoria, lo que refleja el carácter desnormalizado y transitorio de esta tabla.

Fuente: Elaboración propia.

3.5.5. Programas y malla curricular

programs - Representa los programas de estudio ofrecidos por un departamento (por ejemplo, Ingeniería Civil Informática). Sus atributos incluyen un código único (code), el nombre completo del programa (name), el número total de semestres del plan de estudios (total_semesters) y el departamento al que pertenece. La bandera is_hidden permite ocultar programas del calendario público mientras están en proceso de configuración.

program_courses - Tabla de unión que mapea las asignaturas de un programa a su semestre curricular correspondiente (por ejemplo, INF420 corresponde al semestre 5 del programa ICI). El campo semester_number indica la posición de la asignatura en la malla del pro-

grama. La combinación (`program_id`, `course_id`, `semester_number`) tiene restricción `UNIQUE`. Esta información es utilizada por el calendario público para permitir a los estudiantes filtrar las evaluaciones por malla y semestre curricular, mostrando automáticamente las asignaturas relevantes para su nivel sin necesidad de seleccionarlás manualmente.

3.5.6. Decisiones de diseño

UUID como clave primaria. Todos los identificadores primarios son UUID v4, generados sin necesidad de un contador central. Esto evita exponer el tamaño o la tasa de crecimiento de la base de datos a través de identificadores secuenciales, y facilita la eventual integración de datos provenientes de múltiples fuentes sin conflictos de clave.

Borrado suave en evaluaciones. En lugar de eliminar físicamente las filas de `exams` con una instrucción `DELETE`, se registra la fecha de eliminación en el campo `deleted_at`. Esto permite recuperar evaluaciones eliminadas por error sin pérdida de datos, situación que ocurre con frecuencia en sistemas de coordinación. El front-end filtra las evaluaciones eliminadas por defecto, pero expone un control para visualizarlas y restaurarlas para administradores.

Campos de nombre junto a FKs nullable. Los campos `professor_name` en `parallels` y `coordinator_name` en `course_offerings` existen principalmente porque sus claves foráneas correspondientes (`professor_id`, `coordinator_id`) son nullable: un profesor o coordinador puede no tener una cuenta de usuario en el sistema. En esos casos, el campo de nombre es la única forma de registrar quién cumple ese rol, lo que evita perder información relevante por la ausencia de un usuario asociado.

Tabla de staging para importaciones. El diseño de `exam_imports` como buffer intermedio responde a una necesidad práctica: la carga de datos a través de archivos Excel no garantiza que los datos cargados estén en el formato solicitado por el sistema. Escribir directamente a las tablas de producción puede introducir errores silenciosos difíciles de detectar, y además puede complicar la experiencia del usuario, requiriendo a este que edite constantemente el archivo Excel para que sea aceptado por el sistema en caso de errores. La tabla de staging permite almacenar los datos crudos tal como llegaron, ejecutar un matching automático y exponer al administrador los datos alertando de problemas en las filas que requieren intervención manual para poder finalizar la carga.

3.5.7. Normalización, índices e integridad

Normalización. El núcleo del esquema está normalizado hasta la tercera forma normal (3FN): cada tabla tiene una clave primaria única, todos los atributos dependen íntegramente de esa clave y no existen dependencias transitivas entre atributos no clave. Las relaciones de muchos a muchos se resuelven mediante tablas de unión dedicadas, en lugar de duplicar información: `department_admins` entre usuarios y departamentos, y `program_courses`

entre programas y asignaturas. Del mismo modo, los códigos de departamento se extraen a la tabla `department_codes` para evitar un atributo multivaluado en `departments`. La única excepción deliberada a esta normalización es la tabla de *staging* `exam_imports`, descrita anteriormente: su carácter desnormalizado (columnas `raw_*` junto a referencias resueltas) es intencional, pues su función es almacenar datos crudos en tránsito antes de escribirlos, ya normalizados, en las tablas de producción.

Restricciones de integridad. La integridad de los datos se garantiza a nivel de base de datos mediante varias clases de restricciones. Todas las claves foráneas están declaradas explícitamente, de modo que el motor rechaza referencias a entidades inexistentes. Las restricciones `UNIQUE` evitan duplicados lógicos: sobre una sola columna en `users(email)`, `department_codes(code)` y `programs(code)`; y sobre combinaciones de columnas en `courses(department_id, code)`, `course_offerings(course_id, semester_id)`, `parallels(course_offering_id, parallel_code)`, `department_admins(user_id, department_id)` y `program_courses(program_id, course_id, semester_number)`. Dos restricciones `CHECK` garantizan valores positivos en `programs(total_semesters)` y `program_courses(semester_number)`. Una restricción de exclusión sobre `exams` impide además que dos evaluaciones vigentes de un mismo paralelo se crucen en el horario. La validación de que las evaluaciones caigan dentro del rango de fechas de su semestre, en cambio, se resuelve en la capa de aplicación, ya que depende de las fechas del semestre asociado y no de una condición local a la fila.

Índices. Además de los índices creados automáticamente para las claves primarias y las restricciones `UNIQUE`, el esquema define índices secundarios orientados a las consultas más frecuentes del sistema. Las consultas del calendario, que filtran evaluaciones por oferta y las ordenan cronológicamente, se apoyan en los índices `idx_exams_course_offering` (sobre `course_offering_id`) e `idx_exams_starts_at` (sobre `starts_at`). La resolución de ofertas y paralelos por semestre y oferta se apoya en `idx_course_offerings_semester` e `idx_parallels_course_offering`; el filtro por malla curricular, en `idx_program_courses_program_semester`; y la agrupación de lotes de importación, en `idx_exam_imports_upload_id`. Un caso particular es `idx_semesters_visible`, un *índice parcial* definido sobre `start_date` únicamente para las filas con `is_hidden = false`: al indexar solo los semestres visibles, se reduce su tamaño y se acelera la consulta pública del calendario, que nunca considera semestres ocultos.

3.6. Diseño back-end

3.6.1. Decisiones tecnológicas

Cada decisión de esta sección se presenta por el requerimiento que la exige, la propiedad que ese requerimiento vuelve necesaria y el costo que se aceptó al tomarla. Los criterios provienen de los requerimientos no funcionales de la tabla 3 y de una restricción del contex-

to: el sistema debe poder operar en un único servidor modesto, administrado por la misma persona que lo desarrolla y sin un equipo de operaciones detrás.

A ellos se suman tres criterios que atraviesan todas las decisiones. El primero es la proporcionalidad, que mide cada incorporación por la relación entre lo que ahorra y lo que impone: una pieza que resuelve un problema acotado y deja intacto el resto del código se acepta, mientras que la que obliga a organizar el sistema a su alrededor se evita aunque resuelva más, porque su costo no se paga una vez al adoptarla sino cada vez que el sistema cambia. El segundo es la experiencia previa con la tecnología, que en un sistema desarrollado, desplegado y mantenido por una sola persona es un recurso del proyecto antes que una inclinación personal: reduce el riesgo de las decisiones que se toman con la herramienta y vuelve previsible su comportamiento, y por eso se invoca donde efectivamente pesó. El tercero es la adopción entendida como criterio de herencia, que atiende menos a las cualidades de una pieza que a la probabilidad de que quien reciba el sistema sea capaz de hacerse cargo de él, y que se apoya en lo que la adopción arrastra consigo: documentación, un ecosistema alrededor y una comunidad que la mantiene.

Lenguaje: Go. La reproducibilidad del despliegue (RNF-4) exige que el artefacto verificado durante el desarrollo sea idéntico al que se ejecuta en producción, sin pasos de instalación intermedios que introduzcan variación entre ambos entornos. Go compila a un binario estático que, con el enlazado con C deshabilitado (CGO_ENABLED=0), no depende de bibliotecas dinámicas del sistema anfitrión, lo que permite empaquetarlo mediante una construcción en dos etapas en una imagen de ejecución *distroless*, sin sistema operativo, sin intérprete y sin gestor de paquetes, que contiene el binario y nada más y se ejecuta con un usuario sin privilegios. La misma propiedad que asegura la reproducibilidad mantiene así acotado el consumo de recursos del servidor y reduce la superficie de ataque, al no quedar en el entorno de ejecución ningún componente ajeno al programa (RNF-3). Sobre esa base, el tipado estático y la verificación en tiempo de compilación sostienen la mantenibilidad exigida por el RNF-5, y Go era además una tecnología conocida, por haber trabajado antes el autor en el back-end de una aplicación web cuyos servicios estaban implementados en ese lenguaje. El modelo de concurrencia del lenguaje aporta una ventaja adicional, ya que las goroutines permiten expresar los handlers en estilo secuencial y atender solicitudes simultáneas sin propagar construcciones asíncronas por la base de código (RNF-2), con un costo por solicitud que deja margen para absorber crecimientos de carga sobre el mismo servidor.

El costo aceptado es la menor disponibilidad de componentes de propósito general, que obliga a implementar de forma explícita la validación de las entradas, la traducción de los errores del dominio a respuestas HTTP y las funciones de administración; se asume por el alcance acotado del dominio y porque deja el flujo de control explícito en el código.

Router: chi. El enrutamiento es el punto por el que entra toda solicitud y, por lo mismo, la parte del back-end que más condiciona cuánto cuesta agregarle funcionalidades más adelante, que es donde se decide la extensibilidad exigida por el RNF-5. Construirlo sobre la biblioteca estándar únicamente habría dejado fuera tres piezas que el sistema necesita, los parámetros de ruta, la agrupación jerárquica de rutas y el middleware asociado a esos grupos, y

que al reimplementarse pasan a ser código propio que quien herede el proyecto debe aprender antes de poder extenderlo. chi las aporta operando sobre las interfaces de `net/http`, de modo que los handlers siguen siendo `http.Handler` corrientes, y su amplia adopción en la industria opera aquí como criterio de herencia. De las tres, la que más ordena el diseño es el middleware por grupo: la verificación de la sesión se monta sobre el grupo de rutas en lugar de repetirse dentro de cada handler, con lo que la frontera entre lo público y lo protegido queda fijada por la estructura del enrutador y no por que cada función recuerde comprobarla, que es la forma en que esa clase de comprobación termina omitiéndose (RNF-3).

El costo aceptado es introducir una dependencia externa justo en el punto de entrada del sistema; queda acotado porque los handlers y el middleware son los de `net/http`, de manera que lo que depende de chi es el registro de las rutas y no la lógica del sistema.

Base de datos: PostgreSQL. De todas las decisiones tecnológicas del sistema, la del motor de base de datos es la más costosa de revertir: los datos sobreviven a las versiones del código, y cambiar de motor obliga a migrar el esquema, las consultas y las garantías sobre las que se apoya la aplicación, de modo que lo que en un sistema pequeño parece una elección de poca consecuencia se convierte en deuda técnica cuando ese sistema crece. El criterio no fue entonces lo que el sistema necesita hoy, sino qué motor sigue respondiendo cuando el modelo de datos se vuelve más exigente, lo que se decide por la profundidad del sistema de tipos, por que las restricciones las aplique el motor en lugar de la aplicación y por la posibilidad de incorporar extensiones.

PostgreSQL ya cobró esas propiedades en el sistema: las fechas y horas de las evaluaciones se almacenan con zona horaria explícita (`TIMESTAMPTZ`), de modo que el instante queda definido sin depender de la configuración del servidor, y las direcciones de correo se comparan sin distinguir mayúsculas dentro del motor mediante el tipo `CITEXT`, provisto por una extensión. La más relevante es la restricción de exclusión que impide registrar dos evaluaciones con tope de horarios (RNF-6), donde ambas capacidades se encuentran: comprobar un tope exige el operador de solapamiento de rangos temporales, que al carecer de un orden total requiere un índice GiST, mientras que acotar esa comprobación a una misma asignatura y paralelo son igualdades sobre identificadores, y como toda restricción de exclusión reside en un único índice, es la extensión `btree_gist` la que permite que ambas condiciones convivan en él. Al residir en el motor, la restricción rige sobre todas las escrituras sin depender de que la aplicación consulte antes de insertar, comprobación que dos solicitudes simultáneas pueden superar a la vez (RNF-2). A ello se suma la familiaridad previa con el motor, adquirida en el trabajo con sistemas de datos, que reduce el riesgo de las decisiones de modelado y de operación.

El costo aceptado es un motor más exigente de administrar y de mayor consumo de recursos que las opciones concebidas para proyectos pequeños, gasto razonable porque se ejecuta como contenedor con configuración por defecto.

Acceso a datos: bun. Se optó por un ORM antes que por escribir cada consulta directamente en SQL: la consulta se compone desde el código, de modo que un cambio en su forma es un

cambio en el programa, y las filas se corresponden con estructuras de Go, lo que evita repetir el escaneo columna por columna en cada una. Eso importa porque al momento de decidirlo aún no se conoce la forma que tendrán las consultas, y un sistema como este no se resuelve con las simples. Componer desde el código hace además que los valores viajen siempre como parámetros, lo que evita los errores de programación que abren la puerta a la inyección de SQL (RNF-3). Se usó `bun` por familiaridad previa, lo que hizo previsible su comportamiento, y donde componer no aporta las consultas se escriben como SQL literal.

El costo aceptado recae sobre quien herede el sistema, que conocerá SQL antes que la interfaz de una biblioteca en particular; queda acotado porque las operaciones del ORM se corresponden de forma directa con las cláusulas del lenguaje que ya conoce.

Migraciones: `golang-migrate`. El esquema evoluciona con el sistema, y la reproducibilidad (RNF-4) exige que ese historial sea explícito: aplicar los cambios a mano no deja registro de qué se ejecutó en cada entorno, de modo que ninguno queda reconstruible con certeza. Para las migraciones el SQL plano es preferible a expresarlas mediante la biblioteca de acceso a datos, porque son el punto donde el esquema se define con precisión y en el vocabulario del motor. Este esquema lo confirma: instalar una extensión, declarar una restricción de exclusión sobre una expresión y acotarla con un índice parcial son instrucciones propias de PostgreSQL, y escribirlas directamente evita traducirlas a través de una capa intermedia justo donde el resultado debe ser exacto. Como efecto adicional, el historial queda legible para quien no conozca la biblioteca y sobrevive a un eventual reemplazo de esta. `golang-migrate` ejecuta precisamente eso, archivos SQL numerados en pares `up` y `down` versionados junto al código, y se ejecuta como contenedor de un solo uso durante el arranque, lo que incorpora la migración al despliegue en lugar de dejarla como un paso manual que alguien deba recordar.

El costo aceptado es que el esquema queda declarado en dos lugares, los archivos de migración y las estructuras que mapean las tablas, sin que nada obligue a que coincidan: mantenerlos alineados queda a cargo de quien introduce el cambio.

Hash de contraseñas: `bcrypt`. Las contraseñas se almacenan mediante una función de hash de costo deliberado, de manera que a quien llegue a obtener la base de datos le resulte caro probar candidatos a gran escala (RNF-3). Se usó `bcrypt` por su amplia adopción, que tratándose de criptografía mide además el escrutinio acumulado, y por provenir de golang.org/x/crypto, mantenida por el propio equipo del lenguaje. Su factor de costo queda almacenado dentro del hash, así que puede elevarse más adelante sin cambiar de algoritmo ni invalidar las credenciales ya registradas.

El costo aceptado es que `bcrypt` no exige memoria apreciable para calcularse, propiedad que los diseños más recientes incorporan para encarecer los ataques con hardware especializado; migrar a una función que la ofrezca solo requiere recalcular el hash en el siguiente inicio de sesión de cada usuario, por lo que queda registrado como trabajo futuro.

3.6.2. Estructura de paquetes

El back-end está organizado en paquetes internos con responsabilidades bien delimitadas:

- **auth** - Manejo de login, logout, cambio de contraseña, firma y verificación de JWT, gestión de la cookie `httpOnly`, middlewares `RequireAuth` y `RequireAdmin`, y limitador de intentos de login por IP.
- **catalog** - Servicio de gestión (escritura) del catálogo académico: creación, edición y eliminación de departamentos, asignaturas, semestres, ofertas de asignatura y paralelos, como único punto de escritura. Traduce las violaciones de las restricciones de integridad de la base de datos en errores tipados (`ErrNotFound`, `ErrConflict`, `ErrBadInput`) para que los handlers puedan mapearlos a códigos HTTP apropiados sin filtrar errores internos de la base de datos al cliente.
- **gestion** - CRUD de evaluaciones para usuarios autenticados. Verifica permisos por paralelo y oferta, y detecta topes temporales entre evaluaciones del mismo paralelo, rechazándolos con un error HTTP 409.
- **imports** - Carga de archivos Excel, almacenamiento en `exam_imports`, algoritmo de matching difuso contra el catálogo, validación por parte del usuario, que incluye verificar que las fechas queden dentro del rango del semestre, y commit final a la tabla `exams`.
- **admin** - Handlers exclusivos para el administrador global: gestión del catálogo, administración de usuarios y asignación de departamentos, y gestión de programas curriculares.
- **schedule** - Endpoints públicos de solo lectura para el calendario de evaluaciones y los programas de estudio. No requiere autenticación.
- **platform** - Paquete transversal: conexión a la base de datos, configuración del router (CORS, logging, recuperación ante panics, timeouts), helper de respuestas JSON y logger estructurado.
- **config** - Carga de variables de entorno con tipos y valores por defecto (`PORT`, `POSTGRES_*`, `AUTH_SECRET`, `ACCESS_TOKEN_TTL`, `COOKIE_SECURE`, `ALLOW_ORIGIN`, `LOG_LEVEL` y los plazos máximos por operación, como `HANDLER_TIMEOUT`).

La figura 6 muestra las dependencias entre estos paquetes, organizadas en tres niveles. En el nivel superior están los paquetes que exponen los *handlers* HTTP. Estos se apoyan en un nivel de servicios compartidos, formado por `auth`, que provee el middleware de autenticación, y `catalog`, que centraliza la gestión de escritura del catálogo académico. Ambos niveles

descansan sobre la infraestructura transversal, compuesta por `platform` y `config`. La estructura refleja las responsabilidades descritas: `schedule` no depende de `auth`, por tratarse de endpoints públicos, mientras que `gestion`, `imports` y `admin` sí requieren autenticación.

El paquete `catalog` concentra las operaciones de creación y edición del catálogo por un único punto y traduce las violaciones de las restricciones de integridad de la base de datos en errores tipados (`ErrConflict`, `ErrNotFound`, `ErrBadInput`), de modo que los handlers los mapean a códigos HTTP sin filtrar errores internos; lo consume `admin`, que es el rol encargado de administrar el catálogo. Las lecturas del catálogo, en cambio, se resuelven en cada funcionalidad según su patrón de acceso: el calendario (`schedule`) y la gestión de evaluaciones (`gestion`) realizan consultas directas ajustadas a sus vistas, mientras que la importación (`imports`) implementa su propio algoritmo de matching contra el catálogo. Esta separación entre una escritura centralizada, que canaliza todas las modificaciones y su manejo de errores por un solo lugar, y lecturas especializadas por funcionalidad evita acoplar todas las funcionalidades a una única capa de acceso a datos y mantiene cada consulta cercana a la vista que la necesita.

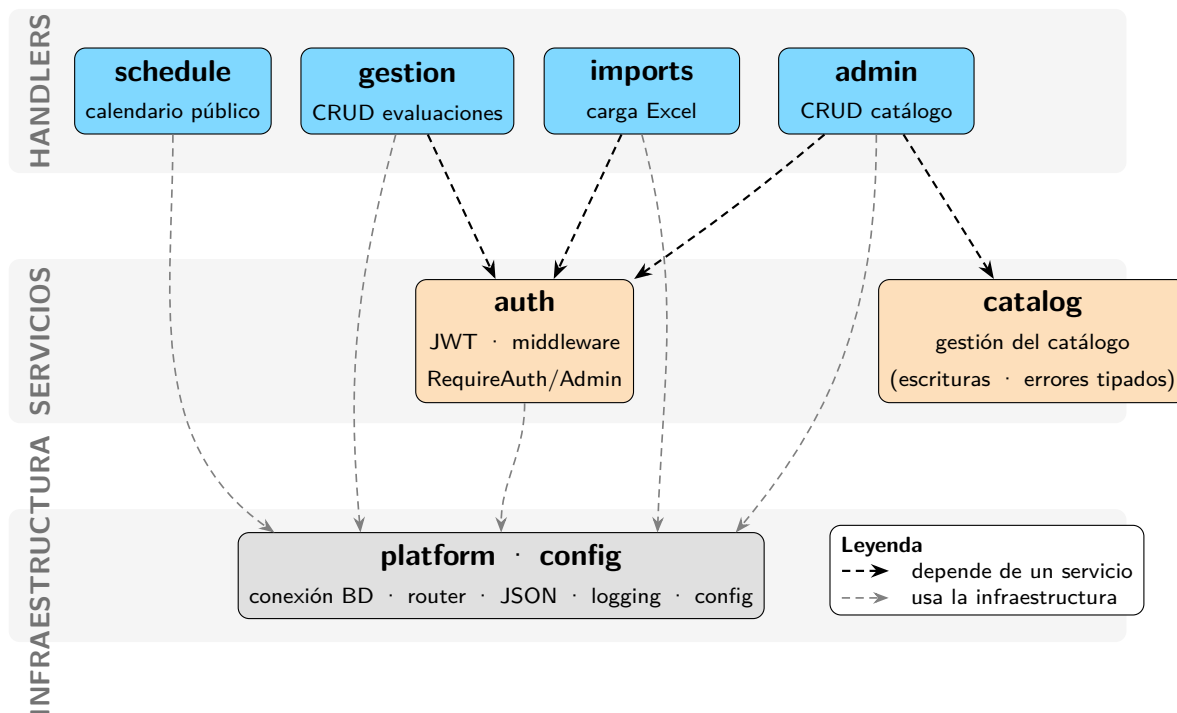


Figura 6: Diagrama de componentes: paquetes del back-end y sus dependencias, agrupados en niveles de *handlers*, servicios e infraestructura.

Fuente: Elaboración propia.

3.6.3. Autenticación y autorización

La autenticación se implementa mediante JSON Web Tokens (JWT) firmados con HMAC-SHA256. Al autenticarse correctamente, el servidor genera un token que contiene el identificador del usuario, una bandera de administrador y un tiempo de expiración configurable. Este token se entrega simultáneamente de dos formas: como cookie `httpOnly` con atributo `SameSite=Lax`, y en el cuerpo de la respuesta JSON.

La cookie `httpOnly` es el mecanismo principal para el cliente web: el navegador la envía automáticamente en cada solicitud y JavaScript no puede leerla, lo que impide que scripts maliciosos inyectados en la página roben el token de sesión. La entrega simultánea en el cuerpo permite que clientes programáticos externos autenticuen sus solicitudes mediante el encabezado `Authorization: Bearer <token>` sin necesidad de acceso a cookies.

La autorización opera en tres niveles:

- **Administrador global** - Acceso completo a todos los datos y funciones. Identificado por la bandera `is_global_admin` en el token JWT. Protegido por el middleware `RequireAdmin`.
- **Jefe de Carrera** - Acceso restringido a los departamentos asignados, verificado mediante consulta a `department_admins` en cada solicitud relevante. Protegido por `RequireAuth` más verificación por handler.
- **Personal académico** - Acceso a los paralelos asignados como profesor. El permiso se verifica a nivel de cada evaluación individual en tiempo de solicitud.

La figura 7 ilustra esta relación: cada nivel está contenido dentro del alcance de datos del nivel anterior, de modo que un administrador global puede ver todo lo que ve un jefe de carrera, y este a su vez todo lo que ve personal académico sobre sus propios paralelos.

3.7. Diseño de la API

La API sigue los principios REST: los endpoints representan recursos del dominio (evaluaciones, asignaturas, semestres, usuarios), y las acciones sobre ellos se expresan mediante los verbos HTTP estándar: GET para consultar, POST para crear, PUT para reemplazar, PATCH para actualizar parcialmente y DELETE para eliminar. Todas las respuestas utilizan un formato JSON uniforme: `{"ok": true, "data": ...}` para respuestas exitosas y `{"ok": false, "error": "...}"` para errores. Los errores internos de la base de datos nunca se exponen al cliente por seguridad, el servidor los registra internamente y retorna mensajes genéricos para no filtrar información sensible del sistema. Todos los endpoints se encuentran bajo el prefijo `/api`.

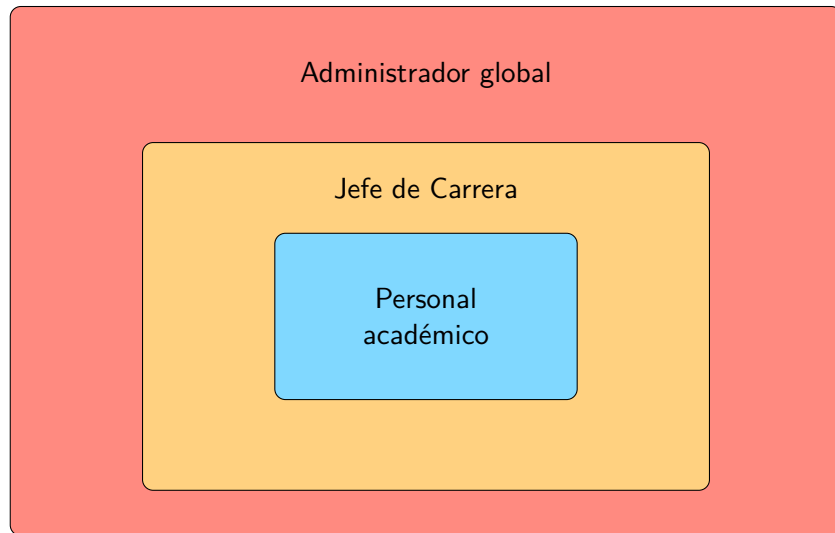


Figura 7: Niveles de autorización, anidados según el alcance de datos visible.

Fuente: Elaboración propia.

Los endpoints se organizan en tres grupos según su nivel de autenticación requerido.

Endpoints públicos - No requieren autenticación.

- [GET](#) /api/schedule/exams - Retorna el calendario de evaluaciones con filtros por semestre, programa y asignatura. Es el endpoint principal consumido por el calendario público.
- [GET](#) /api/schedule/programs - Retorna los programas de estudio disponibles para poblar el filtro de programa en el calendario.
- [POST](#) /api/auth/login - Autentica al usuario y establece la cookie de sesión.
- [POST](#) /api/auth/logout - Elimina la cookie de sesión.

Endpoints autenticados - Requieren sesión activa verificada por el middleware RequireAuth. Dentro de este grupo no todos los usuarios tienen acceso a todos los endpoints: el nivel de acceso sobre los datos depende del rol del usuario y, en el caso de los jefes de carrera, de los departamentos que tienen asignados.

- [GET](#) /api/auth/me - Informa la identidad asociada a la sesión activa. La SPA lo invoca en cada carga de página para restablecer su estado de autenticación; cuando no hay una sesión válida el endpoint responde con 401, respuesta que la SPA interpreta como usuario no autenticado.
- [POST](#) /api/auth/password - Permite al usuario autenticado cambiar su propia contraseña.

- **GET** /api/gestion/exams - Lista evaluaciones con filtros por semestre, departamento y asignatura.
- **POST** /api/gestion/exams - Crea una nueva evaluación.
- **PATCH** /api/gestion/exams/:id - Edita o elimina suavemente una evaluación existente.
- **POST** /api/imports - Sube un archivo Excel e inicia el proceso de importación.
- **GET** /api/imports - Lista las filas importadas agrupadas por lote de carga.
- **PATCH** /api/imports/:id/status - Actualiza el estado de una fila individual.
- **POST** /api/imports/discard / **POST** /api/imports/discard-upload - Descarta filas individuales o un lote completo.
- **POST** /api/imports/validate - Ejecuta una validación y retorna un resumen de filas válidas e inválidas.
- **POST** /api/imports/commit - Confirma la importación de las filas seleccionadas.

Endpoints de administrador global - Requieren `is_global_admin = true`, verificado por el middleware `RequireAdmin`.

- /api/admin/catalog - Operaciones CRUD completas sobre departamentos, códigos de departamento, asignaturas, semestres, ofertas de asignatura y paralelos.
- /api/admin/users - Búsqueda, creación, edición y eliminación (borrado suave) de usuarios, y gestión de sus asignaciones de departamentos administrativos.
- /api/admin/programs - Gestión de programas de estudio y sus mapeos de asignaturas por semestre curricular.

3.7.1. Flujos de solicitud representativos

Para ilustrar cómo interactúan el cliente, el back-end y la base de datos, esta sección presenta tres flujos mediante diagramas de secuencia: la autenticación, la creación de una evaluación con verificación de tope y la importación masiva. En todos ellos el tráfico atraviesa el proxy Nginx descrito en la sección de infraestructura, que se omite en los diagramas por claridad.

Autenticación. El flujo de autenticación (figura 8) comienza cuando la SPA envía las credenciales al back-end. Antes de consultar la base de datos, el back-end valida el formato del payload y aplica un límite de intentos por dirección IP para mitigar ataques de fuerza bruta.

Grupo	Ruta base	Autenticación re-querida	Operaciones principales
Público	/api/schedule, /api/auth (login, logout)	Ninguna	Consultar calendario público, iniciar y cerrar sesión
Autenticado	/api/gestion, /api/imports, /api/auth (me, password)	Sesión activa (cookie o Bearer)	CRUD de evaluaciones, importaciones, sesión y contraseña
Administrador global	/api/admin	is_global_admin = true	CRUD de catálogo, usuarios y programas

Tabla 5: Resumen de grupos de endpoints de la API REST.

Fuente: Elaboración propia.

Si las credenciales son válidas, lo que se comprueba contrastando la contraseña contra su hash bcrypt almacenado, el back-end firma un JWT y lo entrega como cookie `httpOnly` y en el cuerpo de la respuesta. Las credenciales inválidas se responden con un 401 genérico y el exceso de intentos con un 429.

Creación de una evaluación. Este flujo (figura 9) muestra cómo el sistema impide los toques de horario. Tras autenticar la solicitud y validar los campos y el rango de fechas del semestre, el back-end inserta la evaluación, y es el motor el que resuelve: la restricción de exclusión declarada sobre `exams` rechaza toda escritura que deje dos evaluaciones vigentes de un mismo paralelo cruzadas en el horario. Cuando eso ocurre, el back-end traduce la violación a un código 409 y no queda nada escrito; en caso contrario, la evaluación se registra y se responde con un 201. Declarar la condición en el esquema es lo que la sostiene incluso cuando dos solicitudes concurrentes llegan a la vez, situación que la sección 4.8 documenta.

Importación masiva. La importación (figura 10) se desarrolla en cuatro fases sobre la tabla de *staging*. En la *carga*, el back-end procesa el archivo Excel y guarda cada fila con estado *pending*. En el *matching*, al listar las filas el back-end las contrasta contra el catálogo y les asigna candidatos y un estado (*ready*, *review* o *blocked*), y el usuario resuelve las ambigüedades. En la *validación*, un *dry-run* anticipa cuántas filas se importarán correctamente sin escribir nada. Finalmente, en el *commit*, el back-end abre una transacción y, por cada fila lista, resuelve las entidades, inserta la evaluación en la tabla definitiva y marca la fila como *imported*.

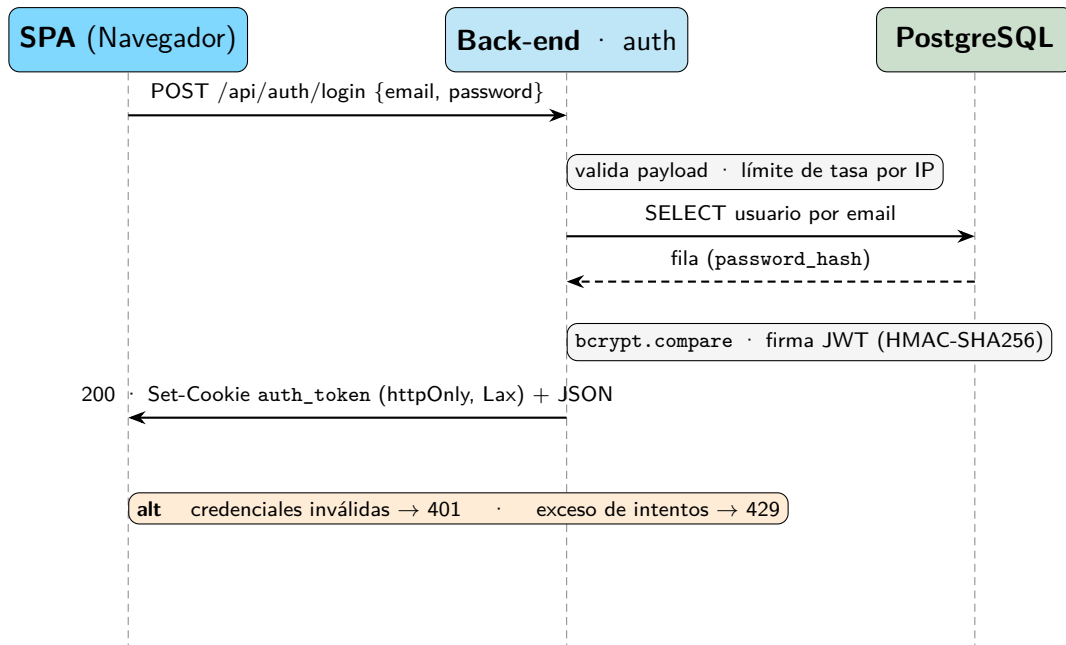


Figura 8: Diagrama de secuencia del flujo de autenticación (login).

Fuente: Elaboración propia.

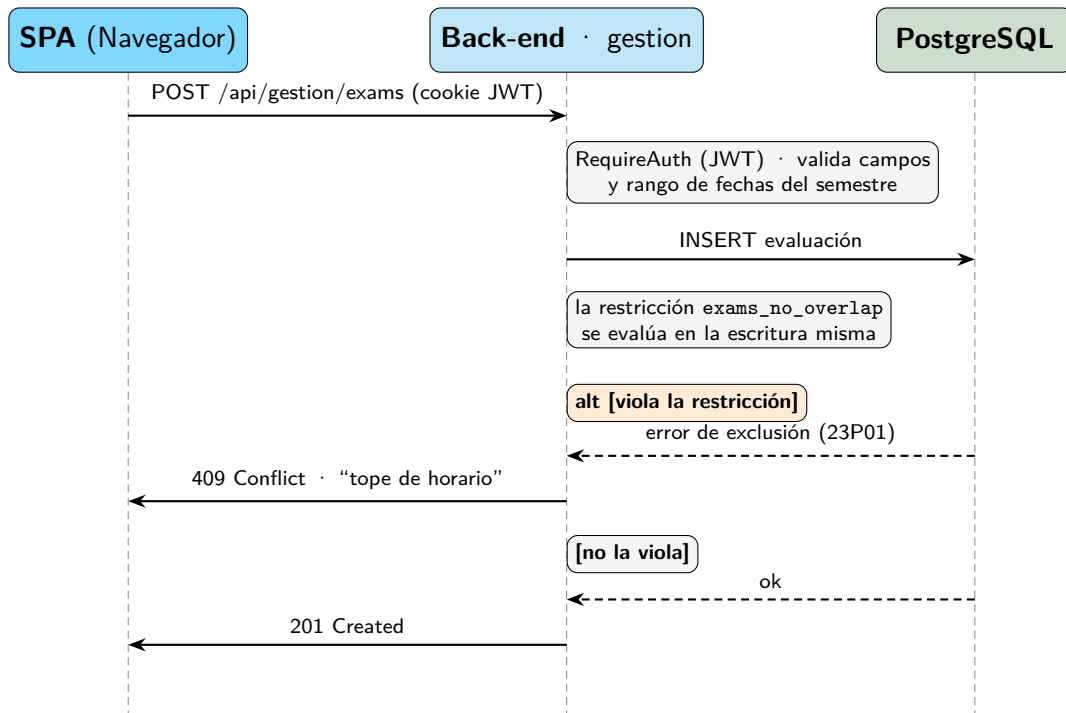


Figura 9: Diagrama de secuencia de la creación de una evaluación con la restricción de exclusión.

Fuente: Elaboración propia.

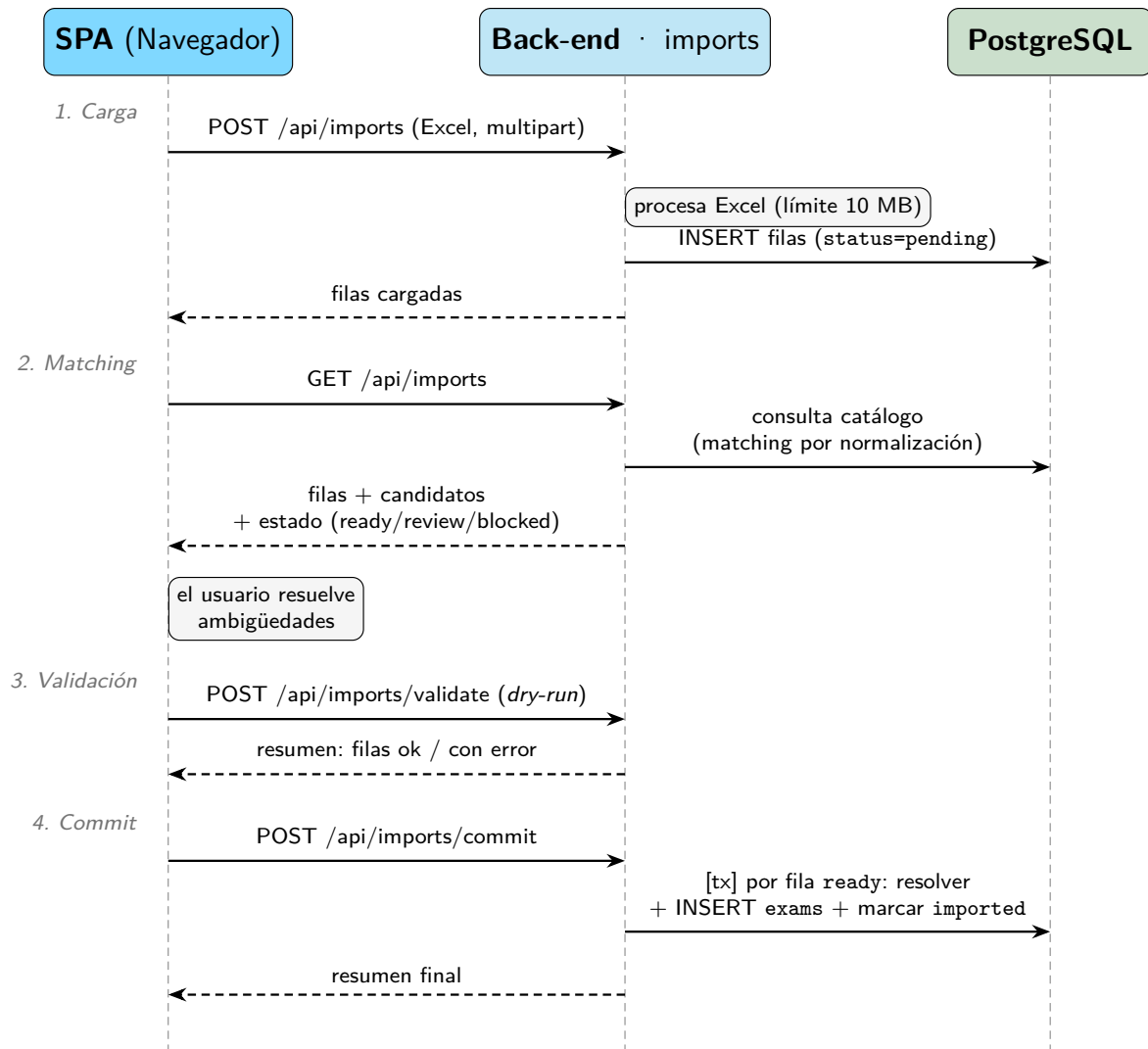


Figura 10: Diagrama de secuencia de la importación masiva desde Excel, en sus cuatro fases.
Fuente: Elaboración propia.

3.8. Infraestructura y despliegue

3.8.1. Docker

Docker es una plataforma de contenedorización que empaqueta una aplicación y todas sus dependencias en una unidad aislada y reproducible llamada contenedor. A diferencia de instalar el software directamente en el sistema operativo del servidor, un contenedor incluye todo lo necesario para ejecutar la aplicación: lenguaje de runtime, librerías del sistema y configuración de entorno. Esto elimina la clase de problemas en la que diferencias sutiles entre el entorno de desarrollo y el de producción causan comportamientos inesperados.

Docker se eligió como plataforma de despliegue porque garantiza que el sistema se comporta de manera idéntica en cualquier máquina con Docker instalado, sin importar el sistema operativo subyacente.

3.8.2. Docker Compose

Docker Compose extiende Docker permitiendo definir y orquestar múltiples contenedores en un único archivo de configuración (`docker-compose.yml`). Cada componente del sistema se ejecuta en su propio contenedor aislado, y Compose gestiona el orden de arranque y las dependencias entre servicios.

El orden de inicialización del sistema es el siguiente: la base de datos (db) arranca primero y expone un healthcheck que indica cuándo está lista. El contenedor de migraciones (`migrate`) espera a que la base de datos esté disponible y aplica todas las migraciones pendientes; luego se inicializa el back-end, y finalmente Nginx queda disponible para terminar el TLS, servir los archivos estáticos de la SPA y enrutar el tráfico hacia el back-end. Este orden garantiza que el esquema de la base de datos siempre esté actualizado antes de que el back-end comience a atender solicitudes, sin necesidad de intervención manual en cada despliegue.

3.8.3. Nginx

Nginx es un servidor web de alto rendimiento utilizado como proxy inverso. Recibe todas las solicitudes entrantes y las redirige al destino correcto: las solicitudes a la ruta `/api/*` son enviadas al back-end Go (puerto 8080), mientras que el resto son respondidas sirviendo los archivos estáticos del bundle de la SPA. En producción, Nginx también se encarga de la terminación TLS, descifrando las conexiones HTTPS antes de reenviarlas a los servicios internos mediante HTTP plano dentro de la red privada de Docker. Adicionalmente, situar a Nginx al frente del back-end permite absorber cargas de tráfico elevadas sin saturar al servicio Go. Nginx sirve directamente los archivos estáticos de la SPA, lo que descarga al back-end de ese

tráfico para que reserve sus recursos a las solicitudes de la API, aplica *buffering* de las conexiones lentas de los clientes y puede limitar la tasa de solicitudes, de modo que los *peaks* de concurrencia se gestionan en el proxy antes de alcanzar la lógica de negocio. Esta arquitectura centraliza en un único punto la gestión de certificados, caché de archivos estáticos y enrutamiento de solicitudes.

3.8.4. Topología de despliegue

La figura 11 integra los elementos anteriores en la topología de despliegue completa: los contenedores que componen el sistema, la red privada que los comunica, los puertos expuestos y el orden de arranque.

Cada componente del sistema corre en su propio contenedor y todos comparten una red privada de Docker (app), de modo que solo Nginx publica puertos hacia el exterior (443 y 80); el back-end (8080) y PostgreSQL (5432) permanecen inaccesibles desde fuera del servidor. El contenedor *migrate* es de un solo uso: aplica las migraciones pendientes y termina, por lo que no forma parte del sistema en régimen. El volumen *pgdb-data* desacopla los datos de PostgreSQL del ciclo de vida del contenedor, permitiendo recrear o actualizar la imagen de la base de datos sin perder la información persistida.

3.9. Diseño front-end

3.9.1. Decisiones tecnológicas

Las decisiones del front-end se rigen por los mismos criterios enunciados para el back-end, con una diferencia de contexto que conviene explicitar: se tomaron desde fuera de la especialidad, de modo que aquí el criterio dominante es la convención del ecosistema. Ante una tecnología que no se domina, seguir lo que la comunidad recomienda y adopta de forma mayoritaria protege mejor al sistema que ejercer un criterio propio que no se tiene, porque una pieza convencional está documentada, sostenida y probada por otros antes que por este proyecto, y porque es la que quien herede el sistema tiene más probabilidad de conocer. A ello se suman la facilidad de aprendizaje y la suficiencia para el problema, entendida como que la pieza resuelva lo que el sistema necesita sin exigir a cambio que se organice a su alrededor. Ese criterio explica la mayoría de las elecciones que siguen y por eso no se repite en cada una; se detallan, en cambio, las decisiones donde el sistema se apartó de la opción convencional, que es donde hubo algo que resolver.

Aplicación de página única. El front-end se construyó como una aplicación de página única, que es la forma convencional de una aplicación web actual y la que sus usuarios esperan encontrar: una interfaz que responde a cada interacción sin recargar la página completa. El

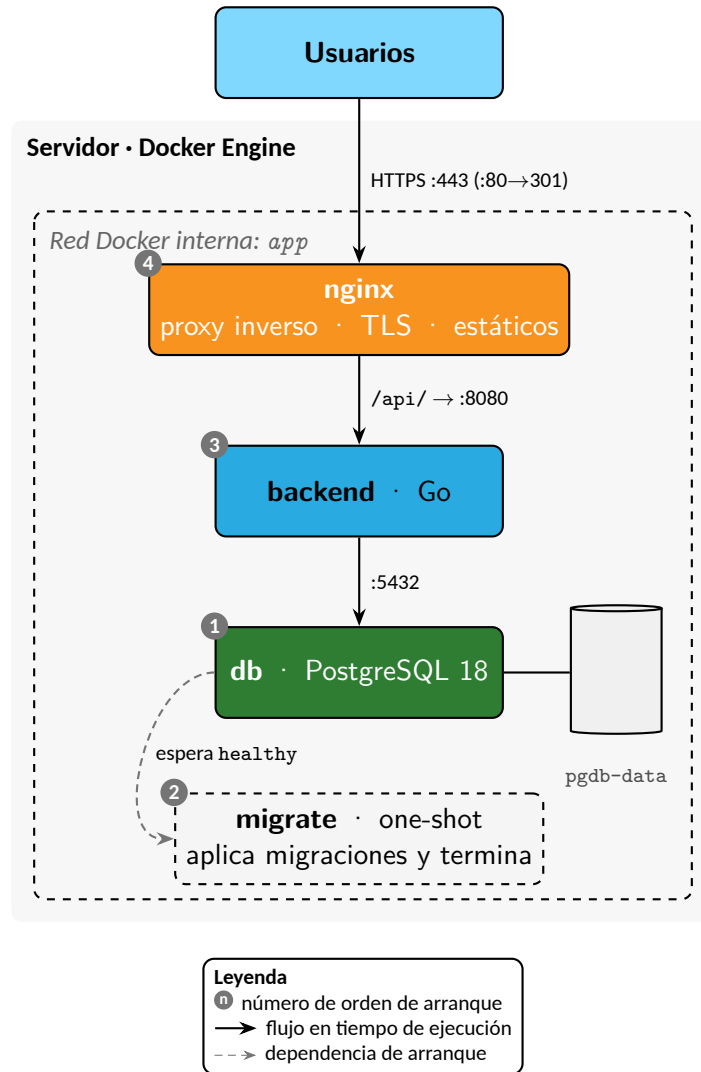


Figura 11: Topología de despliegue en contenedores Docker: servicios, red privada y puertos. Los números indican el orden de arranque (db → migrate → backend → nginx): db expone un *healthcheck* (pg_isready) y los demás servicios esperan a que esté disponible. Nginx publica los puertos 443 y 80 hacia el host, mientras que 8080 (backend) y 5432 (PostgreSQL) permanecen internos a la red app.

Fuente: Elaboración propia.

calendario es donde esa expectativa pesa más, porque el usuario cambia de semestre, filtra por malla y semestre curricular, busca asignaturas y alterna entre la vista mensual y la semanal, mientras la detección de sobrecarga se recalcula sobre la selección vigente (RF-7, RF-8 y RF-9). Todas esas operaciones transforman datos que el navegador ya tiene, de modo que resolver cada una con una recarga completa habría entregado una herramienta perceptiblemente más lenta. El costo aceptado es el proceso de construcción, un segundo lenguaje en el sistema, el volcado inicial de la aplicación al navegador y un fallback de rutas en Nginx, todo lo cual habría podido evitarse resolviendo las páginas en el servidor. El posicionamiento en buscadores, objeción habitual a este enfoque, no aplica en este caso por tratarse de una herramienta interna a la que se accede por su dirección conocida.

Framework: Vue 3. Vue es uno de los frameworks de mayor adopción del ecosistema y el de aprendizaje más accesible entre ellos, lo que lo sitúa de lleno en el criterio con que se tomaron estas decisiones. La propiedad que el sistema aprovecha es su reactividad: los valores derivados rastrean sus dependencias de forma automática, algo pertinente en una pantalla como el calendario, donde las alertas de sobrecarga dependen a la vez de la selección, de los filtros y del semestre. El costo aceptado es un ecosistema de bibliotecas de terceros más acotado que el de la opción más difundida del mercado.

TypeScript. Como los componentes de archivo único de Vue ya exigen un paso de compilación, incorporar tipos no agrega prácticamente costo y sí un beneficio concreto: las respuestas de la API se consumen como tipos explícitos en la capa `api/`, de modo que un cambio en un campo del back-end se manifiesta como un error de compilación antes que como un valor indefinido en pantalla frente a un usuario. El costo aceptado es que esos tipos se mantienen a mano y duplican las estructuras del back-end, sin que nada garantice que no se desincronicen; generarlos a partir de una especificación de la API queda como trabajo futuro.

Herramienta de construcción: Vite. Es la herramienta que el propio ecosistema de Vue recomienda y la que su documentación asume, de manera que la decisión venía tomada de antemano y se consigna por completitud. Como beneficio adicional, acelera el ciclo de desarrollo al servir los módulos nativos del navegador sin empaquetarlos previamente.

Componentes de interfaz: PrimeVue 4. El sistema necesita tablas con ordenamiento y paginación, un selector de fechas restringido al rango del semestre activo, autocompletado para la búsqueda de usuarios, selección en cascada y diálogos de confirmación. Son piezas resueltas hace tiempo y de dificultad conocida, en las que un desarrollo propio habría consumido un esfuerzo considerable sin aportar nada al problema de coordinación, que es donde este trabajo tiene algo que decir; por proporcionalidad, entonces, se adoptó una biblioteca de componentes. PrimeVue cubre ese catálogo, resuelve la consistencia visual mediante un mecanismo único de temas en lugar de hojas de estilo dispersas y trae soporte de accesibilidad, pertinente para la usabilidad exigida por el RNF-1. El costo aceptado es el peso que la biblioteca agrega al paquete que se descarga y la fricción para personalizar un componente más allá de lo que el tema permite.

Vistas de calendario propias. Aquí el sistema se aparta de la opción convencional, que habría

sido incorporar una biblioteca de calendario. El calendario de este sistema no muestra eventos, sino carga académica: agrupa las evaluaciones por semana para clasificar la severidad de esa carga, colorea por asignatura y reacciona a los filtros de malla y semestre curricular. Una biblioteca genérica resuelve la grilla, que es la parte fácil, y no la semántica, que es la que da valor a la vista, de modo que adaptarla habría exigido más trabajo de integración, y más dependencia de sus puntos de extensión, que escribir las dos vistas que el sistema necesita. El costo aceptado es que cualquier vista adicional deberá implementarse a mano.

Manejo de estado: composable propio. El sistema también se aparta aquí de la opción convencional, que habría sido adoptar la biblioteca de manejo de estado que el ecosistema ofrece para este fin. El único estado verdaderamente global de la aplicación es la sesión del usuario, mientras que el resto vive dentro de cada página, y un composable de pocas líneas (`stores/auth.ts`) cubre ese caso sin sumar una dependencia ni una capa de abstracción para un único consumidor. El costo aceptado es que, si más adelante aparece estado compartido entre páginas, habrá que sustituirlo por una solución dedicada.

Ruteo: vue-router. Es la solución que el ecosistema de Vue provee para este fin, de modo que lo relevante no fue adoptarla sino delimitar el alcance de sus *navigation guards*. En el sistema distinguen tres niveles de acceso, ya que las rutas de gestión e importación exigen sesión activa y la de administración exige además el privilegio de administrador global, pero esa distinción es una comodidad de la interfaz antes que un control de seguridad: los guards se ejecutan en el navegador, bajo el control del usuario, y por eso la autorización se verifica siempre en el servidor, como se detalla en la sección 3.11.

3.9.2. Estructura del SPA

El front-end se organiza en cuatro páginas principales, cada una correspondiente a una funcionalidad central del sistema:

- **Home** - Calendario público de evaluaciones, accesible sin autenticación. Incluye selector de semestre, panel de búsqueda de asignaturas por nombre o código, filtro por malla y semestre curricular, vistas de calendario mensual y semanal, y alertas automáticas de conflicto cuando dos o más evaluaciones seleccionadas coinciden en el mismo día o semana.
- **Gestión** - Administración de evaluaciones para usuarios autenticados. Muestra una tabla filtrable de evaluaciones con opciones de creación, edición y eliminación suave, con retroalimentación visual cuando se detecta tope de fechas.
- **Cargas** - Flujo de importación masiva desde archivos Excel, disponible para jefes de carrera y administrador global. Implementa las tres etapas del proceso: carga del archivo, revisión de filas agrupadas por asignatura, y confirmación con validación previa.

- **Administración** - Panel de administración global con tres pestañas: Catálogo (departamentos, asignaturas, semestres, ofertas y paralelos), Usuarios (búsqueda, creación, edición y asignación de departamentos), y Programas (gestión de programas de estudio y su malla curricular).

La capa de API (`api/`) contiene funciones tipadas que encapsulan las llamadas a cada grupo de endpoints, todas configuradas con `credentials`: `'include'` para enviar automáticamente la cookie de sesión. El estado de autenticación se gestiona mediante un composable (`stores/auth.ts`) que hidrata los datos del usuario al cargar la página a través de `GET /api/auth/me`, permitiendo que el estado de sesión persista entre navegaciones sin requerir un nuevo login.

3.10. Diseño interfaz de usuario

Todas las páginas comparten una barra de navegación persistente en la parte superior. Las páginas de *Gestión* y *Administración* utilizan un diseño de dos columnas: una barra lateral de navegación y el área de contenido principal. La página de inicio emplea un diseño de ancho completo con paneles flotantes que se deslizan desde el lado derecho. La página de importaciones usa un diseño de columna única centrada.

3.10.1. Calendario de evaluaciones

La página de inicio es la vista principal orientada al estudiante y no requiere autenticación. Presenta un selector de semestre, un panel de búsqueda de asignaturas por nombre o código, y un panel de filtro por malla y semestre curricular. El calendario ocupa el ancho completo y puede alternarse entre una vista mensual y una semanal.

A medida que el usuario agrega asignaturas, el sistema ejecuta la detección de conflictos sobre el conjunto de evaluaciones seleccionadas y clasifica cada semana según la severidad: tope de horario, mismo día o misma semana. Esta clasificación determina el color de una barra en el margen izquierdo de cada fila del calendario, que va de amarillo a rojo según la gravedad. Al hacer clic en esa barra se abre un diálogo con el detalle de los conflictos detectados en esa semana.

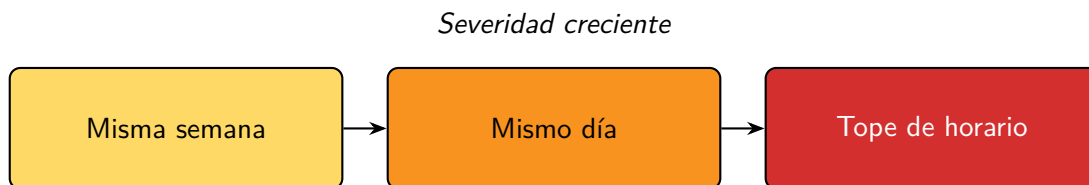


Figura 12: Escala de severidad usada para clasificar las semanas con conflictos.

Fuente: Elaboración propia.

Detalle de la detección de conflictos. La detección trabaja sobre el conjunto de evaluaciones de las asignaturas que el usuario ha seleccionado. De cada evaluación interesan tres datos: su fecha y hora de inicio, su fecha y hora de término, y la oferta de asignatura a la que pertenece, es decir, la asignatura dictada en un semestre determinado. Se dice que dos evaluaciones se cruzan en el horario cuando sus franjas horarias comparten algún tramo de tiempo, es decir, cuando una comienza antes de que la otra termine. Si dos evaluaciones apenas se tocan en un único instante, por ejemplo cuando una termina justo cuando la otra empieza, no se consideran en conflicto.

El sistema solo compara evaluaciones que pertenecen a ofertas de asignatura distintas, porque el objetivo es visibilizar la carga que enfrenta un estudiante que cursa varias asignaturas a la vez, y no las evaluaciones internas de una misma asignatura. A cada par de evaluaciones de asignaturas distintas se le asigna una de tres severidades, excluyentes entre sí y de gravedad creciente (figura 12):

- **Tope de horario:** las dos evaluaciones se cruzan en el horario. Es el caso más grave, porque resulta materialmente imposible rendir ambas.
- **Mismo día:** las evaluaciones no se cruzan en el horario, pero caen en la misma fecha.
- **Misma semana:** las evaluaciones caen en días distintos de una misma semana calendario, de lunes a domingo.

Las evaluaciones se agrupan por semana calendario y, dentro de cada semana, el sistema revisa todos los pares posibles de evaluaciones de asignaturas distintas. La severidad que se atribuye a la semana es la más grave encontrada entre esos pares. Para orientar la atención hacia las semanas más críticas, cada semana recibe además un puntaje que suma sus conflictos dando más peso a los topes de horario que a las coincidencias de día o de semana. Ese puntaje determina el color de la barra lateral del calendario, que va del amarillo al rojo a medida que la semana concentra conflictos más numerosos o más graves.

Esta detección tiene un carácter informativo: resalta los conflictos para que el actor correspondiente decida cómo actuar, pero no bloquea ninguna acción. Ese carácter es deliberado y define el alcance del sistema: la sobrecarga entre asignaturas distintas (RF-8) es precisamente lo que el sistema no puede impedir, porque resolverla exige acuerdos entre profesores que ninguna herramienta puede imponer, pero sí puede anticiparla para que se resuelva a tiempo. La única situación que el sistema descarta de plano es distinta y mucho más acotada: dos evaluaciones vigentes de un mismo paralelo cruzadas en el mismo horario, algo que describe a los mismos estudiantes en dos lugares a la vez y que, más que una decisión de planificación, corresponde a un error de registro. Se trata como una condición de integridad de los datos (RNF-6) y no como una regla de coordinación.

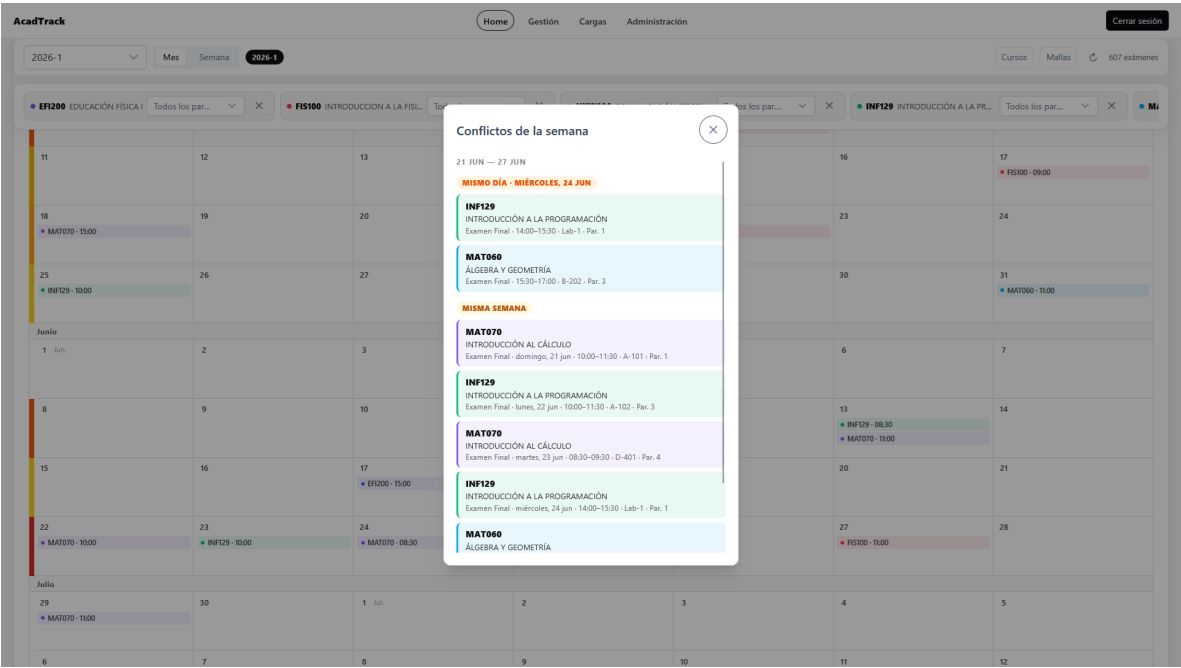


Figura 13: Calendario público de evaluaciones con detección automática de conflictos.
Fuente: Elaboración propia.

3.10.2. Gestión de evaluaciones

La página de gestión presenta una tabla filtrable de evaluaciones del semestre activo. Los usuarios autenticados pueden crear evaluaciones mediante un formulario modal, editar evaluaciones existentes y aplicar borrado suave desde la misma interfaz.

Tipo T1	Asignatura T1	Fecha T1	Horario	Paralelo / Sala	Acciones
Certamen 1	INF246 SISTEMAS OPERATIVOS 2026-1	30-03-2026	09:00 — 11:00	Par. 1 - BONNAIRE FAVRE XAVIER EMMANUEL OLIVIER D-401	Editar Eliminar
Certamen 2	INF246 SISTEMAS OPERATIVOS 2026-1	13-05-2026	15:00 — 17:00	Par. 1 - BONNAIRE FAVRE XAVIER EMMANUEL OLIVIER B-201	Editar Eliminar
Examen Final	INF246 SISTEMAS OPERATIVOS 2026-1	13-06-2026	12:00 — 13:30	Par. 1 - BONNAIRE FAVRE XAVIER EMMANUEL OLIVIER A-102	Editar Eliminar

Figura 14: Vista de gestión de evaluaciones para usuarios autenticados.
Fuente: Elaboración propia.

3.10.3. Flujo de importación masiva

La página de importaciones implementa el flujo por etapas: carga del archivo Excel, almacenamiento en staging, revisión de filas, validación previa y confirmación. Las filas se agrupan por asignatura, de modo que los campos a nivel de oferta (departamento, semestre, identidad de la asignatura) se resuelven una vez por grupo y se propagan a todas sus filas, reduciendo el trabajo de resolución manual.

Resolución de referencias mediante matching. El desafío central de la importación es convertir los valores de texto libre que trae la planilla en referencias a las entidades del catálogo del sistema. Para lograrlo, el sistema procesa cada fila con un algoritmo de matching que resuelve las entidades en un orden que respeta sus dependencias. Primero identifica el semestre y el departamento. Con el departamento ya conocido, busca la asignatura dentro de ese departamento. Luego, a partir de la asignatura y el semestre, ubica la oferta de asignatura correspondiente, es decir, la asignatura dictada en ese semestre. Desde la oferta identifica el paralelo, y por último resuelve al profesor y al coordinador. Resolver en este orden acota la búsqueda de cada entidad al ámbito de las que ya fueron identificadas, por ejemplo buscando una asignatura solo dentro de su departamento, lo que reduce las coincidencias incorrectas.

Antes de comparar, el sistema normaliza cada valor de la planilla para absorber diferencias superficiales de escritura. Convierte el texto a minúsculas y le quita los acentos, y en el caso

de los códigos de asignatura descarta además los caracteres que no son letras ni números, de modo que INF-123, INF 123 e inf123 se tratan como el mismo código. Una vez normalizados los valores, el sistema busca en el catálogo las entidades que les corresponden.

Un aspecto importante del diseño es que todo lo que ocurre después depende solo de cuántos candidatos se encontraron para cada campo, y no del criterio con que se compararon los textos. Esa separación deja el criterio de comparación como un punto de extensión aislado: una versión posterior podría reemplazar la coincidencia exacta sobre el texto normalizado por una comparación aproximada por similitud, más tolerante con erratas de escritura, sin modificar el resto del flujo, ya que este reacciona al número de candidatos y no al método por el que se obtuvieron. En concreto, el resultado de cada campo se reduce a tres casos. Si se encuentra exactamente un candidato, el campo se resuelve de forma automática. Si se encuentran varios, el campo queda marcado como ambiguo y su resolución se delega al usuario, a quien se le ofrece la lista de candidatos para que elija. Si no se encuentra ninguna, el campo queda sin resolver.

Estado de una fila. El estado de cada fila (figura 15) se deduce del resultado de sus campos. Algunos campos son obligatorios para poder crear la evaluación: el departamento, el semestre, la asignatura y las fechas de inicio y término. Si alguno de estos campos obligatorios queda sin resolver, la fila pasa al estado `blocked` y no puede importarse hasta que el usuario complete la información que falta. Si todos los campos obligatorios están resueltos pero alguno de ellos tiene varios candidatos posibles, la fila pasa al estado `review`, lo que significa que puede importarse, pero solo después de que el usuario confirme cuál candidato es el correcto. Si todos los campos obligatorios están resueltos y ninguno presenta ambigüedad, la fila queda en estado `ready`. El usuario puede promover manualmente una fila desde `review`, completar los campos de una fila `blocked` para llevarla a `ready`, o descartar cualquier fila que todavía no haya sido importada. Solo las filas en estado `ready` se escriben en la tabla de producción cuando se confirma la carga. Este esquema refleja un patrón de trabajo con intervención humana: el sistema resuelve por su cuenta los casos en que la evidencia es clara y le presenta al usuario únicamente aquellos que no puede decidir con seguridad, evitando así tanto la escritura de datos incorrectos como el trabajo manual sobre las filas que sí puede resolver solo.

Antes de confirmar, el sistema ejecuta una validación y presenta un resumen con cuántas filas se importarán exitosamente y cuáles fallarán, permitiendo al usuario cancelar si el resultado no es el esperado.

En la figura 15, las flechas discontinuas hacia `discarded` representan que el usuario puede descartar una fila desde cualquiera de los estados de trabajo, es decir desde `review`, `ready` o `blocked`, mientras la fila no haya sido importada.

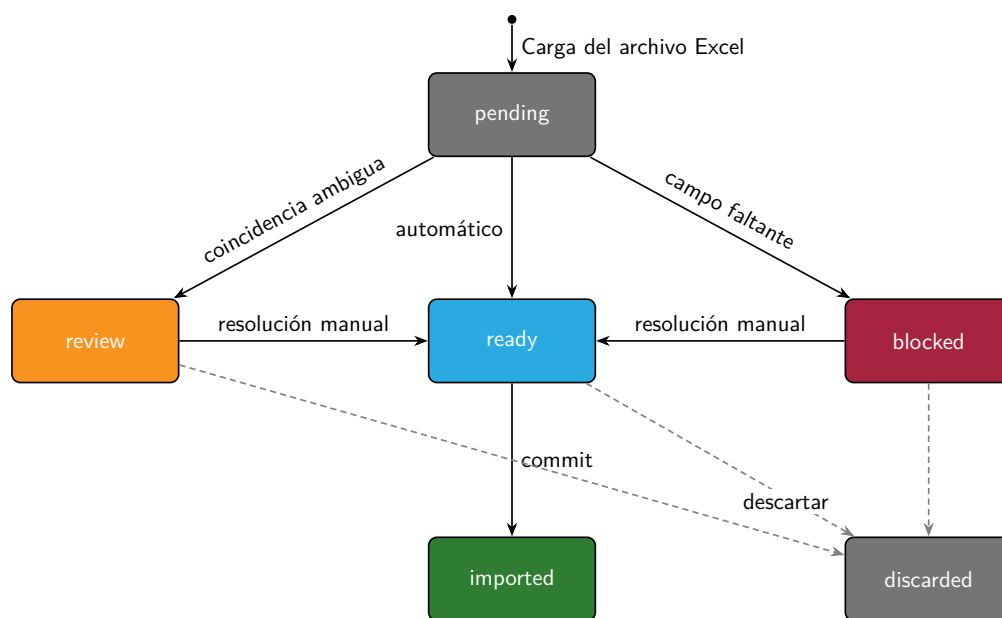


Figura 15: Ciclo de vida de una fila de importación.
Fuente: Elaboración propia.

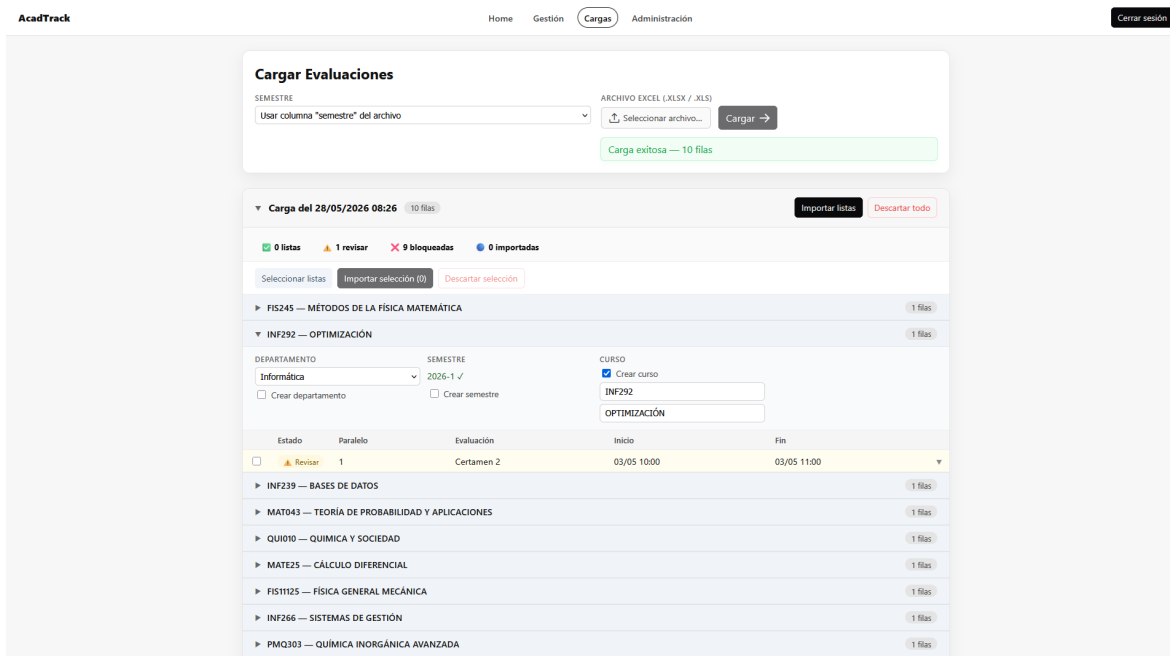


Figura 16: Revisión de filas en el flujo de importación masiva desde Excel.
Fuente: Elaboración propia.

3.10.4. Panel de administración

El panel de administración organiza sus funciones en tres pestañas: *Catálogo* (departamentos, asignaturas, semestres, ofertas de asignatura y paralelos), *Usuarios* (búsqueda, creación, edición y asignación de departamentos administrativos) y *Programas* (gestión de programas de estudio y su malla curricular). Cada pestaña carga su propio componente de gestión de forma independiente.

Los jefes de carrera ven únicamente los departamentos que tienen asignados; los administradores globales ven todos. Este filtrado se aplica en el componente raíz del panel antes de pasar los datos a cada subcomponente, de modo que los componentes internos no necesitan conocer el rol del usuario activo.

The screenshot displays the 'AcadTrack' system interface. At the top, there's a navigation bar with 'Home', 'Gestión', 'Cargas', and 'Administración' (the active tab). A 'Cerrar sesión' button is on the right. The left sidebar shows a menu for 'Administración' with sub-items: 'Admin global', 'Asignaturas', 'Ofertas y paralelos' (highlighted), 'Mallas', 'Semestres', 'Departamentos', and 'Usuarios'. The main content area is titled 'OFERTAS Y PARALELOS' and 'Ofertas de asignaturas'. It includes a description: 'Crea asignaturas por semestre y gestiona sus paralelos.' The form contains several fields: 'Departamento' (dropdown with 'Informática' selected), 'Asignatura' (dropdown with 'INF20325 — ESTRUCTURA DE DATOS' selected), 'Semestre' (dropdown with '2026-1' selected), 'Selección oferta existente' (dropdown with 'Nueva oferta' selected), and a checkbox for 'Asignatura coordinada'. There are also input fields for 'Código' (with '1' entered), 'Profesor (opcional)' (with a search prompt 'Buscar o texto libre...'), and 'Coordinador (opcional)' (with a search prompt 'Buscar por nombre...'). Action buttons include 'Nueva oferta', 'Agregar paralelo', and a large 'Crear oferta' button at the bottom. At the bottom left of the page, there are 'Actualizar datos' and 'Cerrar sesión' buttons.

Figura 17: Panel de administración del sistema.
Fuente: Elaboración propia.

3.11. Consideraciones de seguridad

La seguridad del sistema (RNF-3) no reside en un componente concreto, sino en decisiones repartidas entre el navegador, el proxy inverso, el back-end y la base de datos. Esta sección las consolida como un modelo de amenazas: primero delimita qué se protege y de quién, luego enumera las amenazas consideradas y su mitigación, y finalmente expone los trade-offs asumidos en el diseño.

Activos y superficie de ataque. Los activos a proteger son cuatro: las credenciales de los

usuarios, las sesiones activas, la integridad de las fechas de evaluación y la disponibilidad del servicio. La integridad merece atención particular: el daño de una escritura no autorizada no es la exposición de un dato sensible, sino una fecha de evaluación falsa o alterada que una parte del estudiantado tomaría por oficial, con consecuencias académicas concretas. La superficie expuesta es deliberadamente estrecha. El sistema se despliega en un único servidor, donde el proxy inverso es el único componente con puertos publicados a Internet: atiende en el puerto 443 y redirige el 80 hacia él. El back-end y la base de datos son alcanzables solo desde la red interna de contenedores, de modo que ninguna solicitud llega al back-end sin atravesar antes el proxy, que aplica la terminación TLS y el límite de tasa por dirección IP. Sobre esa superficie se consideran tres actores de amenaza: un atacante anónimo desde Internet, que tiene acceso legítimo al calendario público (RNF-7) y por lo tanto a parte de la API; un usuario autenticado que intenta operar fuera del alcance que le corresponde; y un atacante situado en la red entre el navegador y el servidor. Quedan fuera del alcance la seguridad física del servidor, el compromiso del proveedor de infraestructura y los ataques de denegación de servicio distribuidos a gran escala, que exceden lo que una aplicación puede mitigar por sí sola.

La tabla 6 resume las amenazas consideradas, el vector por el que se materializan y el control que las mitiga.

Autorización en el servidor. La verificación de permisos merece subrayarse porque es donde una aplicación web suele fallar. El sistema trata la interfaz como una comodidad y no como una barrera: los *navigation guards* de vue-router y los botones que se ocultan según el rol solo evitan que el usuario navegue hacia donde no le sirve. Toda decisión de autorización se toma nuevamente en el servidor, en cada solicitud, a partir de los *claims* del token y de una consulta de alcance, porque el cliente está bajo el control de quien lo usa y la API es alcanzable sin pasar por él.

Trade-off: sesiones sin estado y revocación. Firmar la sesión en un token en lugar de guardarla en el servidor evita mantener estado de sesión, pero tiene una consecuencia directa: el servidor no puede invalidar un token ya emitido. El cierre de sesión borra la cookie del navegador, aunque el token siga siendo válido hasta que expire, de modo que una copia obtenida antes seguiría sirviendo durante ese lapso. El tiempo de expiración configurable acota esa ventana. Las alternativas evaluadas fueron mantener una lista de revocación, que reintroduce el estado que la decisión buscaba evitar, y emitir tokens de vida muy corta con un mecanismo de renovación, que agrega complejidad al cliente. Se aceptó la limitación porque los usuarios son internos y conocidos, y el activo en juego es la integridad de un calendario académico y no datos financieros. Si el sistema creciera en alcance, la renovación con tokens de vida corta sería el siguiente paso.

Trade-off: SameSite en lugar de un token anti-falsificación. La protección contra la falsificación de solicitudes entre sitios descansa en el atributo `SameSite=Lax` de la cookie y no en un token dedicado por formulario. El atributo impide que un sitio de terceros envíe la cookie en solicitudes que no sean una navegación de primer nivel, y como todas las escrituras del sistema usan POST, PUT, PATCH o DELETE, quedan cubiertas. El límite conocido es que Lax sí

Amenaza	Vector	Mitigación
Fuerza bruta de credenciales	Intentos de inicio de sesión repetidos	Límite de cinco intentos por minuto y por dirección IP en el back-end, sobre el límite general de tasa del proxy; respuesta genérica que no revela si el correo existe
Filtración de contraseñas	Acceso a la base de datos o a un respaldo	Almacenamiento como hash bcrypt con factor de costo, nunca en texto plano ni de forma reversible
Robo de sesión desde el navegador	Script inyectado que intenta leer el token	Cookie httpOnly, inaccesible desde JavaScript; Vue escapa por defecto el contenido interpolado
Robo de sesión en tránsito	Escucha de la red	TLS terminado en el proxy, redirección de HTTP a HTTPS y cookie con atributo Secure
Falsificación de solicitudes entre sitios	Sitio de terceros que dispara una escritura con la sesión de la víctima	Cookie con SameSite=Lax y escrituras restringidas a métodos distintos de GET
Inyección SQL	Texto del usuario en filtros, búsquedas o importaciones	Consultas parametrizadas generadas por bun, sin concatenación de entrada en el SQL
Falsificación del token	Alterar los <i>claims</i> para declararse administrador	Firma HMAC-SHA256 verificada con comparación en tiempo constante; secreto inyectado por variable de entorno
Escalada de privilegios	Invocar la API directamente, omitiendo la interfaz	Middlewares RequireAuth y RequireAdmin más verificación del alcance (departamento o paralelo) en cada solicitud
Fuga de información técnica	Provocar errores para inferir el esquema	Errores tipados traducidos a mensajes genéricos; el detalle va solo al log del servidor
Agotamiento de recursos	Cargas voluminosas o solicitudes lentas	Límite de tamaño en la carga de archivos, <i>timeouts</i> por operación, recuperación ante <i>panics</i> y límite de tasa en el proxy

Tabla 6: Modelo de amenazas: amenazas consideradas y controles que las mitigan.

Fuente: Elaboración propia.

permite navegaciones GET, lo que resulta inocuo aquí porque ningún GET modifica datos y la lectura del calendario es pública por diseño. Un token anti-falsificación habría agregado una capa independiente del comportamiento del navegador, al costo de sincronizarlo en cada formulario.

Gestión del secreto de firma. El secreto que firma los tokens se inyecta como variable de entorno y no reside en el repositorio, ya que quien lo conozca puede emitir tokens válidos con privilegios arbitrarios y ningún otro control lo detendría. Por la misma razón, el despliegue debe fallar de forma ruidosa si el secreto no está definido, en vez de continuar con un valor por defecto: un valor por defecto que funciona es un modo de falla silencioso, porque el sistema opera con normalidad mientras su autenticación es trivialmente falsificable. La misma consideración aplica a las credenciales de la base de datos.

3.12. Nuevo proceso de planificación de evaluaciones

Con el sistema en operación, el proceso de programación de evaluaciones cambia de manera significativa para cada actor involucrado. Además de centralizar la carga de evaluaciones, el sistema detecta automáticamente conflictos de horario y semanas con sobrecarga de evaluaciones, y los resalta visualmente en el calendario para que cualquiera de los tres actores pueda identificarlos e intervenir de forma preventiva antes de que el semestre esté en curso.

Los jefes de carrera cargan las evaluaciones de las asignaturas coordinadas y las asignaturas externas al inicio del semestre, utilizando el flujo de importación masiva desde Excel o mediante la creación individual en la interfaz de gestión. Estas fechas quedan inmediatamente disponibles en el calendario centralizado para todos los usuarios. Cuando el jefe de carrera consulta el calendario y selecciona las asignaturas a su cargo, el sistema resalta los conflictos de horario y las semanas con sobrecarga de evaluaciones detectados, permitiéndole reorganizar dichas evaluaciones antes de que el semestre esté en curso, de forma proactiva en lugar de reactiva.

Los profesores consultan el calendario centralizado antes de definir las fechas de sus propias evaluaciones, visualizando la carga existente para cada semana del semestre. Las fechas confirmadas quedan registradas y son visibles de inmediato para todos los usuarios del sistema. Si al consultar el calendario el profesor observa que una de sus evaluaciones queda marcada con un conflicto de horario, puede reprogramarla si aún es posible, en lugar de enterarse del problema solo cuando un estudiante se acerca a negociar un cambio.

Los estudiantes pueden consultar su calendario personalizado en cualquier momento del semestre, filtrando por su malla y semestre curricular para ver únicamente las asignaturas relevantes a su situación, o también seleccionando asignaturas específicas que está tomando. El sistema resalta visualmente las semanas en que dos o más evaluaciones de las asignaturas seleccionadas coinciden en el mismo día o en la misma semana, permitiendo a los estudiantes prepararse con anticipación o solicitar un ajuste antes de que la sobrecarga sea inevitable.

Este nuevo proceso transforma la coordinación de evaluaciones de una actividad descentralizada, dependiente de la memoria individual y sin visibilidad compartida, a un proceso centralizado donde jefes de carrera, profesores y estudiantes consultan la misma información en tiempo real y pueden identificar visualmente un conflicto y actuar de forma preventiva, en lugar de reaccionar una vez que el problema ya afecta a los estudiantes.

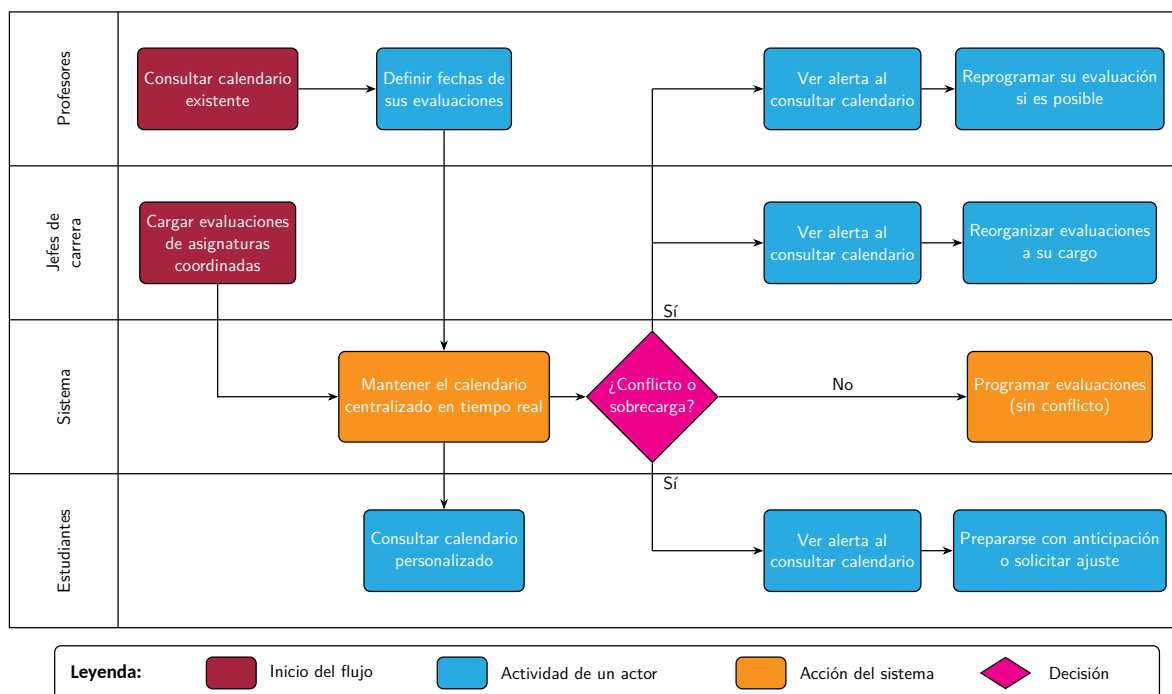


Figura 18: Nuevo proceso de planificación de evaluaciones con el sistema de coordinación. Cada carril representa un actor; las cajas son actividades y los rombos, decisiones.

Fuente: Elaboración propia.

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

4.1. Metodología de validación

La validación del sistema se organiza en niveles complementarios, ordenados según su cercanía al código. En la base están las pruebas automatizadas, que verifican el comportamiento de la API y de las reglas de autorización e integridad contra una base de datos real, y que el desarrollador ejecuta durante el desarrollo cada vez que introduce un cambio de fondo. Sobre ellas, los escenarios funcionales ejecutados por el desarrollador verifican los flujos completos de cada funcionalidad, de la interfaz a la base de datos, sobre un conjunto de datos representativo. En la cúspide, una evaluación de aceptación con usuarios mide lo que ningún nivel anterior puede observar: la usabilidad percibida por quienes ejercen los roles a los que el sistema se dirige. Cada nivel cubre así una clase de defecto distinta, y los tres en conjunto permiten trazar cada requerimiento del capítulo 3 a la evidencia de su cumplimiento.

La distribución de responsabilidades entre niveles es una decisión metodológica. El trabajo administrativo, como crear usuarios, definir mallas curriculares o mantener el catálogo, no forma parte de la interacción cotidiana con el sistema: se realiza pocas veces por semestre y por un número reducido de personas. Ese trabajo se valida en el nivel automatizado, que puede ejercitar los caminos administrativos de forma repetible e incluir los casos límite que una sesión con usuarios difícilmente produce. La evaluación con usuarios se reserva para los flujos de uso frecuente, que son el calendario público, la gestión de evaluaciones y la importación masiva.

Los escenarios funcionales se ejecutan sobre un conjunto de datos construido a partir de información real del Departamento de Informática: las asignaturas, los paralelos, los profesores y las mallas curriculares son reales, y las evaluaciones fueron generadas para poblar los semestres de prueba con la estructura habitual de certámenes y exámenes. Esto permite ejercitar el sistema con el volumen y la forma de los datos que enfrentaría en producción, sin depender de las planillas históricas de los departamentos.

La tabla 7 indica el nivel en que se valida cada requerimiento y la sección de este capítulo que documenta la evidencia correspondiente. La reproducibilidad del despliegue (RNF-4) no aparece en la tabla porque su evidencia es la puesta en producción misma: el sistema desplegado se construye y ejecuta con las mismas definiciones de contenedores utilizadas durante el desarrollo. La mantenibilidad (RNF-5) tampoco se valida empíricamente en este trabajo; descansa en las decisiones de diseño y organización del código documentadas en el capítulo 3.

Requerimiento	Nivel de validación	Evidencia
RF-1 (autenticación)	Automatizada	Sección 4.2
RF-2, RF-3 (catálogo y usuarios)	Automatizada	Sección 4.2
RF-4 (mallas curriculares)	Automatizada	Sección 4.2
RF-5 (gestión individual)	Funcional y aceptación	Secciones 4.4 y 4.7
RF-6 (importación masiva)	Funcional y aceptación	Secciones 4.3 y 4.7
RF-7, RF-9 (calendario y vistas)	Funcional y aceptación	Secciones 4.5 y 4.7
RF-8 (detección de sobrecarga)	Funcional y aceptación	Secciones 4.5 y 4.7
RF-10 (registro histórico)	Funcional	Sección 4.5
RNF-1 (usabilidad)	Aceptación	Sección 4.7
RNF-2 (rendimiento)	Carga y concurrencia	Sección 4.6
RNF-3 (seguridad)	Automatizada y diseño	Secciones 4.2 y 3.11
RNF-6 (integridad)	Automatizada y esquema	Secciones 4.2 y 4.8
RNF-7 (acceso público)	Funcional	Sección 4.5

Tabla 7: Requerimientos y nivel de validación que aporta su evidencia.

Fuente: Elaboración propia.

Las secciones siguientes documentan cada nivel: las pruebas automatizadas, los escenarios funcionales con la entrada utilizada y el comportamiento observado, la evaluación de usabilidad con usuarios y los hallazgos que la validación produjo sobre el propio sistema. El manual de usuario del sistema, organizado por rol, se incluye en el anexo A.

4.2. Pruebas automatizadas

El back-end cuenta con dos capas de pruebas automatizadas. Las pruebas unitarias verifican la lógica que no depende de la base de datos, concentrada en la normalización y el matching del flujo de importación y en la carga de configuración del servicio. Las pruebas de integración levantan el enrutador completo de la API contra una instancia real de PostgreSQL y ejercitan los endpoints como lo haría un cliente HTTP, incluidas la autenticación por cookie, la autorización por rol y las restricciones declaradas en el esquema. Probar contra el motor real, en lugar de sustituirlo por una imitación en memoria, es deliberado: buena parte de las garantías del sistema reside en el esquema, y una prueba que no pasa por el motor no las observa.

La suite de integración cubre la autenticación y el ciclo de sesión (RF-1); el mantenimiento del catálogo académico (RF-2) y la administración de usuarios (RF-3); la definición de mallas curriculares (RF-4); la creación de evaluaciones, incluido el rechazo de dos evaluaciones de un mismo paralelo que se cruzan en el horario (RNF-6); y el flujo de importación completo. Una parte de estos casos ejercita el sistema de forma adversa, en línea con el modelo de amenazas de la sección 3.11: se verifica que un Jefe de Carrera no pueda leer ni modificar recursos de un departamento que no administra, aunque construya las solicitudes a mano,

y que la importación no pueda escribir en un departamento distinto del autorizado aunque el cuerpo de la solicitud lo nombre (RNF-3).

En conjunto, la suite automatizada supera los cien casos de prueba y cubre cerca de un tercio de las sentencias del código del back-end. La cobertura se concentró deliberadamente en los caminos de escritura y en las reglas de autorización e integridad, donde un defecto es silencioso y su costo se descubre tarde, antes que en los caminos de lectura, cuya falla es visible de inmediato en la interfaz.

4.3. Importación masiva de evaluaciones

Este escenario valida que el sistema permita cargar las evaluaciones de un semestre completo a partir de un archivo Excel entregado por un departamento, con revisión asistida y validación previa a la confirmación (RF-6).

4.3.1. Carga del archivo

Se sube un archivo `.xlsx` con las evaluaciones del Departamento de Informática para el semestre 2026-2 (figura 19). El sistema procesa el archivo, normaliza las celdas de fecha y hora, y almacena cada fila en la tabla de importaciones con estado `pending`, agrupadas bajo un mismo identificador de carga. El resultado se confirma de inmediato al usuario con un resumen de cuántas filas fueron reconocidas (figura 20).

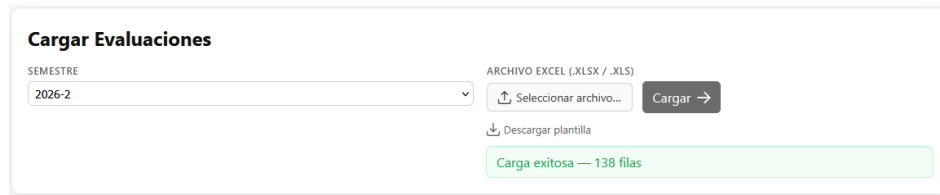


Figura 20: Confirmación de carga exitosa con el número de filas reconocidas.
Fuente: Elaboración propia.

4.3.2. Revisión de filas

Las filas almacenadas se agrupan por asignatura, y para cada una el sistema ejecuta el algoritmo de matching automático sobre departamento, semestre, asignatura, paralelo y profesor. Cada fila queda con un estado visible (*Lista*, *Revisar* o *Bloqueada*) según el resultado de esa coincidencia (figura 21).

The screenshot shows the 'Cargar Evaluaciones' (Load Evaluations) interface. At the top, there is a navigation bar with 'Home', 'Gestión', 'Cargas' (highlighted), and 'Administración'. A 'Cerrar sesión' (Log out) button is in the top right. The main form has a title 'Cargar Evaluaciones'. Below it, there is a 'SEMESTRE' dropdown menu with '2026-2' selected. To the right, there is a section for 'ARCHIVO EXCEL (.XLSX / .XLS)' with a file upload icon, the filename 'Informatica.xlsx', and a 'Cargar →' button. Below the upload section is a link to 'Descargar plantilla' (Download template). At the bottom of the form, a message states: 'Aún no hay cargas. Sube un archivo para comenzar.' (There are no loads yet. Upload a file to get started.)

Figura 19: Selección del archivo Excel y semestre de destino para la carga.
Fuente: Elaboración propia.

Para una fila bloqueada por falta de la asignatura, se observa que el sistema indica explícitamente el campo faltante (Falta: asignatura) y expone un formulario para resolverlo manualmente en este caso, marcando la opción de crear la asignatura y el paralelo junto con los datos ya reconocidos del profesor y la sala (figura 22). Esto demuestra que ninguna fila ambigua se importa sin intervención humana, y que la intervención requerida es mínima cuando los datos crudos son parcialmente reconocibles.

AcadTrack Home Gestión Cargas Administración Cerrar sesión

Seleccionar listas Importar selección (0) Descartar selección

▼ INF446 — ALGORITMOS MULTIOBJETIVO 3 filas

DEPARTAMENTO: Informática SEMESTRE: — seleccionar — CURSO: INF446 — ALGORITMOS ...

☐ Crear curso

Estado	Paralelo	Evaluación	Inicio	Fin
☐ Lista	1	Certamen 1	02/04 16:00	02/04 17:30
☐ Lista	1	Certamen 2	08/05 15:00	08/05 16:59
☐ Lista	1	Examen Final	26/06 16:59	26/06 18:29

▼ INF221 — ALGORITMOS Y COMPLEJIDAD 3 filas

DEPARTAMENTO: Informática SEMESTRE: — seleccionar — CURSO: INF221 — ALGORITMOS Y ...

☐ Crear curso

Estado	Paralelo	Evaluación	Inicio	Fin
☐ Revisar	1	Certamen 1	25/03 10:59	25/03 12:29
☐ Revisar	1	Certamen 2	11/05 16:00	11/05 18:00
☐ Revisar	1	Examen Final	19/06 15:00	19/06 16:30

▼ INF236 — ANÁLISIS Y DISEÑO DE SOFTWARE 3 filas

DEPARTAMENTO: Informática SEMESTRE: — seleccionar — CURSO: — seleccionar —

☐ Crear curso

Estado	Paralelo	Evaluación	Inicio	Fin
☐ Bloqueada	1	Certamen 1	21/03 16:59	21/03 18:00
☐ Bloqueada	1	Certamen 2	18/05 10:59	18/05 12:29
☐ Bloqueada	1	Examen Final	11/06 10:59	11/06 12:00

▼ INF245 — ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORES 4 filas

DEPARTAMENTO: Informática SEMESTRE: — seleccionar — CURSO: — seleccionar —

☐ Crear curso

Figura 21: Filas de importación agrupadas por asignatura, con su estado de resolución.
Fuente: Elaboración propia.

▼ INF245 — ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORES 4 filas

DEPARTAMENTO: Informática SEMESTRE: 2026-1 (activo) CURSO: — seleccionar —

☐ Crear curso

Estado	Paralelo	Evaluación	Inicio	Fin
☐ Bloqueada	1	Certamen 1	25/04 10:59	25/04 12:29

Falta: curso

PARALELO ☒ Crear paralelo

Raw: 1

EVALUACIÓN Raw: Certamen 1

INICIO Raw: 25/04 10:59

FIN Raw: 25/04 12:29

SALA Raw: Lab-2

PROFESOR Cuenta del sistema (opcional) Raw: CAÑAS ROBLES JAVIER HERNAN

COORDINADOR ☒ Evaluación coordinada Cuenta del sistema (opcional) Raw: CAÑAS ROBLES JAVIER HERNAN

Figura 22: Resolución manual de una fila bloqueada por falta de coincidencia de asignatura.
Fuente: Elaboración propia.

4.3.3. Commit y validación previa

Antes de escribir cualquier dato en las tablas de producción, el usuario selecciona las filas a importar y el sistema ejecuta una validación. El resultado muestra cuántas filas se importarán correctamente y cuáles fallarán, junto con la razón de cada falla, incluyendo, en este caso, evaluaciones cuyas fechas quedan fuera del rango del semestre seleccionado (figura 23). Esto permite al usuario corregir o descartar filas problemáticas antes de confirmar, evitando que errores de origen en el archivo lleguen al calendario público.

Al confirmar, el commit puede terminar en tres resultados observados: éxito total, cuando todas las filas seleccionadas se procesan sin error (figura 24); éxito parcial, cuando algunas filas fallan pero el resto se importa igualmente (figura 25); o falla, cuando ninguna fila pudo procesarse (figura 26). En todos los casos el sistema reporta el detalle por fila y solo las filas exitosas pasan a estado `imported`, lo que confirma que el flujo de importación es seguro frente a datos parcialmente inválidos.

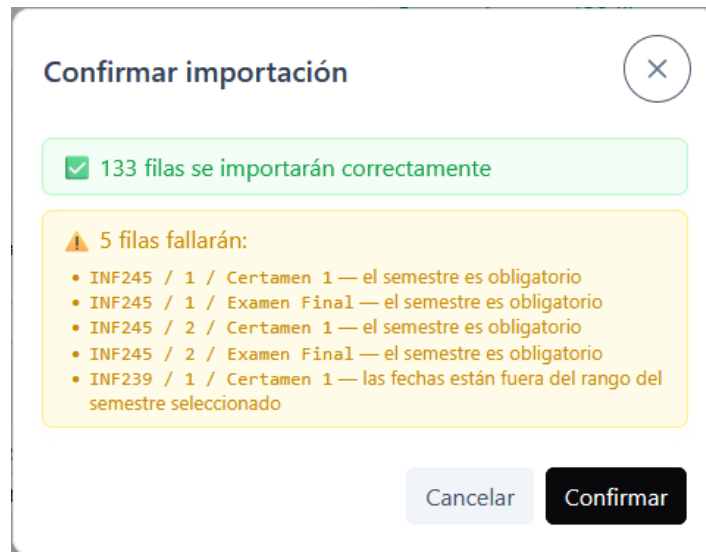


Figura 23: Resumen de validación previa al commit, con filas que se importarán y filas que fallarán.

Fuente: Elaboración propia.



Figura 24: Resultado de un commit exitoso.

Fuente: Elaboración propia.

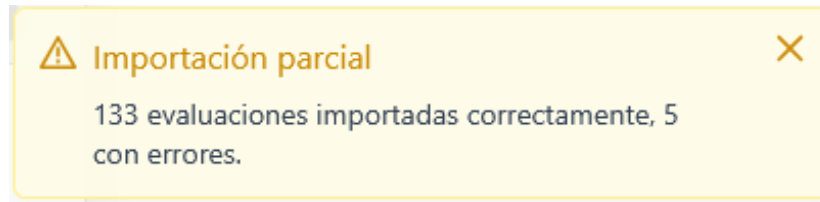


Figura 25: Resultado de un commit parcialmente exitoso.
Fuente: Elaboración propia.

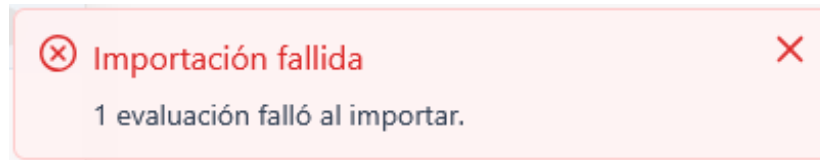


Figura 26: Resultado de un commit fallido.
Fuente: Elaboración propia.

4.4. Gestión individual de evaluaciones

Este escenario valida la capacidad del sistema para crear, editar y eliminar evaluaciones de forma individual (RF-5), el mecanismo que complementa la carga masiva durante el semestre en curso.

La página de gestión lista las evaluaciones del semestre activo con filtros por departamento, asignatura y estado de eliminación (figura 27). Al crear una nueva evaluación mediante el formulario modal, se ingresan el nombre, las fechas de inicio y fin, la sala y notas opcionales para la oferta de asignatura seleccionada, entendida como una asignatura dictada en un semestre determinado (figura 28); el mismo formulario se reutiliza para editar una evaluación existente.

Al intentar crear una evaluación que se cruza en el horario con otra ya registrada para la misma oferta y paralelo, el sistema rechaza la operación y muestra el mensaje correspondiente sin cerrar el formulario, permitiendo corregir el horario antes de reintentar (figura 29). Este rechazo proviene de la restricción de integridad declarada en el esquema (RNF-6), por lo que rige por igual en la edición individual y en la importación masiva; la sección 4.8 documenta cómo la validación con usuarios motivó ese diseño.

La eliminación de una evaluación pide confirmación explícita antes de aplicar el borrado suave (figura 30). Activando el filtro *Mostrar eliminados*, la evaluación eliminada permanece visible en la tabla marcada como *Eliminado*, junto con la opción de restaurarla (figura 31), lo que evita la pérdida irreversible de datos ante un borrado accidental.

2026-1

Informática

INF134 — ESTRUCT...

☐
Mostrar eliminados

+ Nuevo examen

Tipo	Asignatura	Fecha	Horario	Paralelo / Sala	Acciones
Prueba 1 Compartido	INF134 ESTRUCTURAS DE DATOS 2026-1	23-06-2026	14:30 — 15:00	Todos F-123	Editar Eliminar
Examen Final	INF134 ESTRUCTURAS DE DATOS 2026-1	15-06-2026	10:00 — 11:30	Par. 1 · CAZORLA SANTANA JOSE MIGUEL Lab-2	Editar Eliminar
Examen Final	INF134 ESTRUCTURAS DE DATOS 2026-1	15-06-2026	06:00 — 07:30	Par. 1 · CAZORLA SANTANA JOSE MIGUEL Lab-2	Editar Eliminar
Certamen 2	INF134 ESTRUCTURAS DE DATOS 2026-1	11-05-2026	17:00 — 19:00	Par. 1 · CAZORLA SANTANA JOSE MIGUEL A-102	Editar Eliminar
Certamen 2	INF134 ESTRUCTURAS DE DATOS 2026-1	11-05-2026	12:59 — 14:59	Par. 1 · CAZORLA SANTANA JOSE MIGUEL A-102	Editar Eliminar
Certamen 1	INF134 ESTRUCTURAS DE DATOS 2026-1	01-04-2026	10:00 — 11:30	Par. 1 · CAZORLA SANTANA JOSE MIGUEL C-301	Editar Eliminar
Certamen 1	INF134 ESTRUCTURAS DE DATOS 2026-1	01-04-2026	07:00 — 08:30	Par. 1 · CAZORLA SANTANA JOSE MIGUEL C-301	Editar Eliminar

<<

<

1

>

>>

20

>

Figura 27: Listado filtrable de evaluaciones en la página de gestión.
Fuente: Elaboración propia.

Nuevo examen

Oferta de curso *

INF134 — ESTRUCTURAS DE DATOS · 2026-1

Paralelo

Examen compartido (todos los paralelos)

Nombre del examen *

Prueba 1

Fecha inicio *

23/06/2026

Hora inicio *

02:30 PM

Fecha fin *

23/06/2026

Hora fin *

03:00 PM

Sala

F-123

Notas

Prueba Corta 1

Cancelar

Crear examen

Figura 28: Formulario de creación de una nueva evaluación.
Fuente: Elaboración propia.

Nuevo examen ✕

Oferta de curso *
INF134 — ESTRUCTURAS DE DATOS · 2026-1 ▾

Paralelo
Examen compartido (todos los paralelos) ▾

Nombre del examen *
Prueba 3

Fecha inicio *
23/06/2026 📅

Hora inicio *
02:30 PM

Fecha fin *
23/06/2026 📅

Hora fin *
03:00 PM

Sala
F-123

Notas
Indicaciones adicionales...

Ya existe un examen que se superpone en ese horario para esta oferta.

Cancelar Crear examen

Figura 29: Rechazo de una evaluación que se cruza en el horario con otra existente.
Fuente: Elaboración propia.

Confirmar eliminación ✕

⚠️ ¿Eliminar "Prueba 1" de INF134?

Cancelar Eliminar

Figura 30: Confirmación previa al borrado suave de una evaluación.
Fuente: Elaboración propia.

2026-1	Informática	INF134 — ESTRUCT...	☒ Mostrar eliminados	+ Nuevo examen		
Tipo [1]	Asignatura [1]	Fecha [1]	Horario	Paralelo / Sala	Acciones	
Prueba 1 Compartido	INF134 ESTRUCTURAS DE DATOS 2026-1	23-06-2026	14:30 — 15:00	Todos F-321	Editar	Eliminar
Prueba 2 Eliminado Compartido	INF134 ESTRUCTURAS DE DATOS 2026-1	23-06-2026	02:30 — 02:45	Todos F-123	Editar	Restaurar
Examen Final	INF134 ESTRUCTURAS DE DATOS 2026-1	15-06-2026	10:00 — 11:30	Par. 1 · CAZORLA SANTANA JOSE MIGUEL Lab-2	Editar	Eliminar
Examen Final	INF134 ESTRUCTURAS DE DATOS 2026-1	15-06-2026	06:00 — 07:30	Par. 1 · CAZORLA SANTANA JOSE MIGUEL Lab-2	Editar	Eliminar
Certamen 2	INF134 ESTRUCTURAS DE DATOS 2026-1	11-05-2026	17:00 — 19:00	Par. 1 · CAZORLA SANTANA JOSE MIGUEL A-102	Editar	Eliminar
Certamen 2	INF134 ESTRUCTURAS DE DATOS 2026-1	11-05-2026	12:59 — 14:59	Par. 1 · CAZORLA SANTANA JOSE MIGUEL A-102	Editar	Eliminar
Certamen 1	INF134 ESTRUCTURAS DE DATOS 2026-1	01-04-2026	10:00 — 11:30	Par. 1 · CAZORLA SANTANA JOSE MIGUEL C-301	Editar	Eliminar
Certamen 1	INF134 ESTRUCTURAS DE DATOS 2026-1	01-04-2026	07:00 — 08:30	Par. 1 · CAZORLA SANTANA JOSE MIGUEL C-301	Editar	Eliminar

Figura 31: Listado con evaluaciones eliminadas visibles y opción de restauración.
Fuente: Elaboración propia.

4.5. Calendario público y detección de conflictos

Este escenario valida la vista global semestral de las evaluaciones, accesible sin autenticación (RF-7, RNF-7), sus vistas mensual y semanal (RF-9), la detección de semanas de alta carga (RF-8) y la consulta de semestres anteriores como registro histórico (RF-10).

Un visitante no autenticado puede explorar el calendario público agregando asignaturas de interés, sin necesidad de iniciar sesión (figura 32). El selector de semestre permite alternar entre los semestres disponibles sin perder las asignaturas ya seleccionadas (figura 33), lo que además da acceso a los semestres pasados como registro histórico consultable, y la vista semanal muestra el detalle de horario y sala de cada evaluación del conjunto seleccionado (figura 34). Adicionalmente, el panel de mallas permite seleccionar todas las asignaturas de un semestre curricular de una malla específica con un solo paso, en vez de buscarlas una por una (figura 35).

Sobre el conjunto de asignaturas seleccionadas, el sistema ejecuta la detección de conflictos y agrupa los resultados por semana, distinguiendo entre evaluaciones que caen el mismo día y evaluaciones que caen en la misma semana (figura 36). Esta alerta es la materialización directa del objetivo de detectar de manera proactiva la alta carga académica de los estudiantes, permitiendo identificar semanas críticas antes de que el semestre esté en curso.

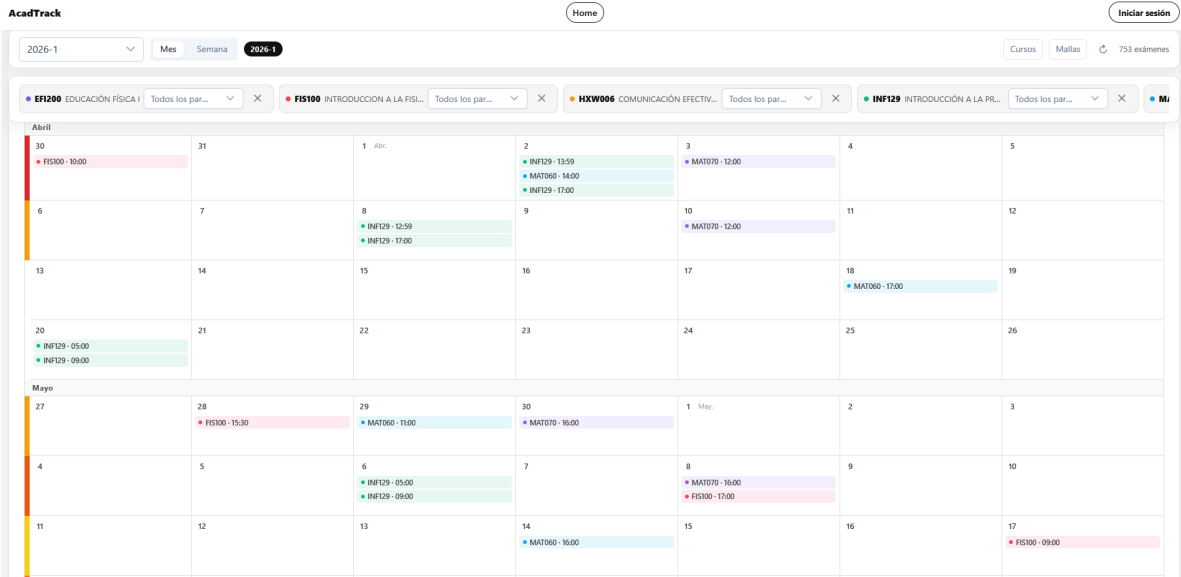


Figura 32: Calendario público de evaluaciones, accesible sin autenticación.
Fuente: Elaboración propia.

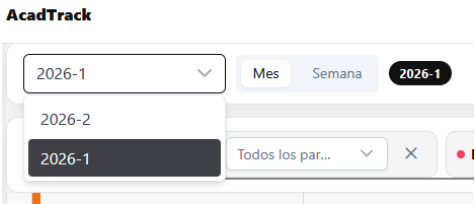


Figura 33: Selector de semestre del calendario público.
Fuente: Elaboración propia.

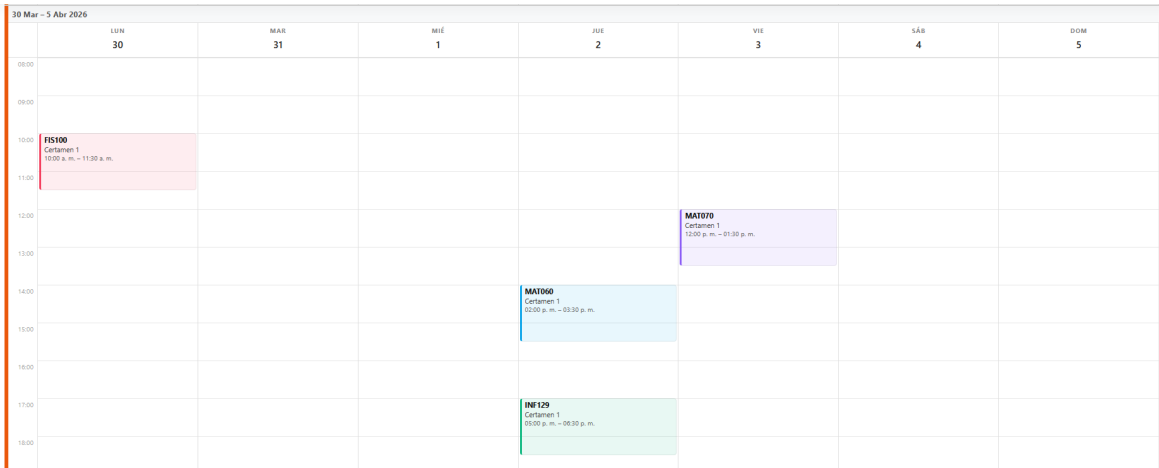


Figura 34: Vista semanal del calendario con el detalle de horario y sala de cada evaluación.
Fuente: Elaboración propia.

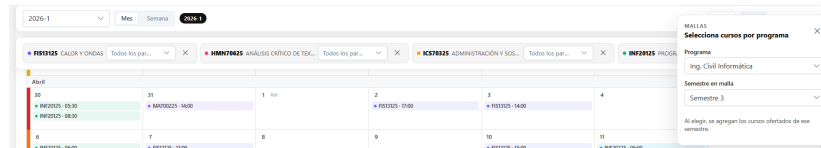


Figura 35: Selección de asignaturas por malla y semestre curricular.
Fuente: Elaboración propia.



Figura 36: Detalle de conflictos de la semana, agrupados por severidad.
Fuente: Elaboración propia.

4.6. Prueba de carga y concurrencia

El requerimiento RNF-2 exige que el sistema atienda accesos concurrentes al calendario sin una degradación significativa del tiempo de respuesta. El escenario de mayor carga se supone conocido de antemano: el calendario público es el único punto del sistema expuesto

a un número de clientes que la institución no controla, pues no requiere autenticación, y cabe esperar que las consultas se concentren al inicio del semestre y en los días previos a las semanas de mayor concentración de evaluaciones, cuando el interés por consultarlo es mayor. El supuesto sigue el patrón habitual de este tipo de servicios y es el que define el peor caso al que conviene dimensionar; la prueba se diseña, por lo tanto, sobre los endpoints de consulta del calendario.

El escenario somete la consulta de evaluaciones de un semestre, con el conjunto de datos descrito en la metodología cargado, a niveles crecientes de solicitudes concurrentes, y registra para cada nivel el rendimiento en solicitudes por segundo, la latencia en su mediana y sus percentiles 95 y 99, y la tasa de error, hasta identificar el punto en que la latencia se degrada. De forma complementaria, una prueba de concurrencia sobre la creación de evaluaciones verifica que la restricción de integridad del esquema mantiene la invariante bajo solicitudes simultáneas, que es justamente el caso que una comprobación realizada por la aplicación antes de escribir no puede garantizar.

La prueba se ejecutó con la herramienta de medición ApacheBench, versión 2.3, sobre dos entornos distintos, porque la capacidad de la aplicación y la capacidad del servidor contratado para alojarla son dos preguntas separadas y conviene no confundirlas. El primer entorno es un equipo de desarrollo con procesador Intel Core i5-10300H de cuatro núcleos y 16 GB de memoria, con la base de datos PostgreSQL ejecutándose en un contenedor de la misma máquina y las solicitudes cursadas por la interfaz de red local; mide lo que la aplicación entrega cuando el hardware no la limita. El segundo repite el escenario íntegro sobre un servidor pequeño, simulado con las restricciones de recursos de Docker: el contenedor del back-end se limitó a un núcleo y 1 GB de memoria, que es lo que ofrece el plan del servidor donde el sistema está efectivamente desplegado. En ambos casos cada solicitud recorre el back-end y la base de datos completos y recibe el conjunto de datos íntegro del semestre de prueba, cerca de 0,55 MB de JSON con sus más de 600 evaluaciones, sin caché ni compresión de por medio. Las cifras no incluyen la latencia de red ni la terminación TLS del despliegue real, que agregan un costo por solicitud pero no alteran la capacidad de servicio medida aquí. Las tablas 8 y 9 resumen ambas corridas.

Clientes concurrentes	Solicitudes/s	p50 (ms)	p95 (ms)	p99 (ms)	Errores
1	91,6	11	12	13	0
10	294,8	29	48	229	0
25	317,2	72	114	269	0
50	327,6	146	220	313	0
100	330,8	286	473	579	0
200	334,0	563	984	1213	0

Tabla 8: Resultados de la prueba de carga sobre la consulta del calendario público, en el equipo de desarrollo.

Fuente: Elaboración propia.

Clientes concurrentes	Solicitudes/s	p50 (ms)	p95 (ms)	p99 (ms)	Errores
1	20,4	47	57	245	0
10	25,8	383	592	697	0
25	27,0	899	1298	1502	0
50	28,1	1710	2469	2798	0
100	28,9	3218	5100	5264	68
200	41,4	4998	5221	5327	449

Tabla 9: Resultados de la misma prueba con el back-end limitado a un núcleo y 1 GB de memoria, los recursos del servidor donde el sistema está desplegado. La columna de errores cuenta las solicitudes de cada nivel, sobre mil, que no obtuvieron una respuesta correcta.

Fuente: Elaboración propia.

En el equipo de desarrollo los resultados muestran la curva de saturación esperable en un servicio limitado por procesamiento. El rendimiento crece hasta estabilizarse en torno a las 330 solicitudes por segundo a partir de los 25 clientes concurrentes, y desde ese punto la concurrencia adicional se traduce en tiempo de espera en cola: la latencia crece de forma aproximadamente proporcional al número de clientes, mientras la tasa de error se mantiene en cero en todos los niveles, incluido el de 200 clientes simultáneos, donde el percentil 99 alcanza 1,2 segundos. La aplicación, por lo tanto, no impone por sí misma un techo cercano a la demanda del problema.

Sobre el servidor pequeño el cuadro cambia de forma sustantiva, y esa diferencia es el resultado más informativo de la sección. El rendimiento se estabiliza en torno a las 28 solicitudes por segundo, un orden de magnitud por debajo, y a partir de los 100 clientes concurrentes aparecen errores: un 7 % de las solicitudes a ese nivel y un 45 % a los 200. Todos responden a un mismo mecanismo, que el registro del servicio identifica sin ambigüedad: el plazo máximo por operación, fijado en cinco segundos, vence mientras la consulta sigue en curso, de modo que la solicitud se cancela y el servicio responde con un error antes que esperar indefinidamente. Los errores no señalan entonces un agotamiento del servidor, sino la decisión deliberada de descartar el trabajo que excede ese plazo.

El alza aparente de rendimiento a los 200 clientes conviene leerla con cuidado, porque no indica una mejora. Al vencer el plazo, las solicitudes canceladas liberan el procesador antes de completarse, de manera que el sistema descarta trabajo y con ello atiende antes lo que queda. Descontando las respuestas fallidas, el rendimiento útil desciende de unas 27 solicitudes por segundo a los 100 clientes a unas 23 a los 200: pasado el punto de saturación, el sistema deja de degradarse de forma benigna y empieza a perder solicitudes.

Puestas en la escala del problema, las 28 solicitudes por segundo del servidor de despliegue equivalen a unas 1 700 cargas completas del calendario por minuto, cifra que deja un margen amplio sobre la demanda que puede producir un departamento, aunque no el margen holgado que sugiere la primera corrida. Dos precisiones acotan el alcance de la segunda. La restricción se aplicó al contenedor del back-end, mientras la base de datos siguió disponien-

do de los recursos del equipo de desarrollo; en el servidor real ambos comparten el mismo núcleo, junto con el proxy inverso, de modo que las cifras medidas son una cota superior de lo que ese servidor entrega. Y el escenario se ejecuta sin compresión, que el proxy podría aplicar sobre una respuesta de este tamaño con un efecto que aquí no se midió.

La prueba de concurrencia sobre la escritura reprodujo el caso que la sección 4.8 identifica como imposible de garantizar mediante una comprobación en la aplicación: veinte solicitudes idénticas y simultáneas de creación de una evaluación para el mismo paralelo y horario. El resultado fue exactamente el esperado: una solicitud creó la evaluación, las diecinueve restantes recibieron la respuesta HTTP 409 y en la base de datos quedó registrada una única fila. La invariante se sostiene bajo concurrencia porque la verifica el motor en la escritura misma, que es lo que la prueba se propuso demostrar (RNF-6).

La lectura conjunta de ambas corridas delimita el cumplimiento de RNF-2 mejor que cualquiera por separado. El requerimiento se cumple para la demanda que el escenario supone: el servidor de despliegue atiende sin errores hasta los cincuenta clientes concurrentes, con una degradación progresiva de la latencia, y su techo cubre con margen las consultas que un departamento puede generar. Lo que la segunda corrida precisa es dónde está el límite y de qué depende. No lo fija el diseño de la aplicación, que en hardware holgado atiende un orden de magnitud más, sino el plan del servidor contratado, y el punto en que la degradación deja de ser benigna queda determinado por el plazo de cinco segundos por operación. Ese plazo, la compresión de la respuesta en el proxy y el tamaño del conjunto de datos que el calendario devuelve son los tres parámetros sobre los que actuar si la demanda llegara a acercarse a ese punto, y ninguno exige rediseñar el sistema.

4.7. Evaluación de usabilidad (SUS)

Esta sección corresponde al nivel de aceptación de la metodología: mide con usuarios el requerimiento no funcional de usabilidad (RNF-1), que exige que el sistema pueda ser utilizado por personal sin formación técnica con una capacitación mínima.

La *System Usability Scale* (SUS) es un cuestionario estandarizado de diez afirmaciones, cada una evaluada en una escala Likert de 1 a 5, cuyas respuestas se combinan en un puntaje único de 0 a 100 que mide la usabilidad percibida de un sistema [Brooke, 1996]. Las afirmaciones alternan entre formulaciones positivas y negativas, lo que obliga a leer cada una con atención antes de responder y reduce el riesgo de respuestas mecánicas. Se eligió este instrumento por ser el estándar de facto en evaluación de usabilidad: es breve, independiente de la tecnología o dominio del sistema evaluado, y produce un puntaje numérico que permite comparar la usabilidad percibida entre distintos grupos de usuarios del sistema. Como criterio de aceptación se fijó de antemano el punto de referencia de 68 puntos, que corresponde al promedio agregado entre los estudios que utilizan el instrumento: un puntaje sobre ese umbral indica una usabilidad percibida superior a la mitad de los sistemas evaluados con SUS en la literatura [Sauro, 2011].

4.7.1. Protocolo de prueba

El cuestionario se administró a tres grupos de participantes: uno con el rol de Jefe de Carrera y dos con el rol de personal académico. El primero lo componen docentes del propio Departamento de Informática, que son quienes hoy realizan la coordinación de evaluaciones; los otros dos reúnen a personas que ejercen los roles de ayudante y de profesor fuera del departamento, incorporadas como representantes del perfil de uso antes que como usuarios actuales. Los dos grupos de personal académico se evaluaron en sesiones independientes y se distinguen por el alcance de la tarea que realizaron, según se detalla más adelante.

Un primer grupo, de tres participantes, realizó la prueba utilizando cuentas de prueba con el rol de *Jefe de Carrera*. El grupo estuvo compuesto por tres docentes del Departamento de Informática, todos mayores de 45 años: dos profesoras que ejercen como coordinadoras de asignaturas y un profesor con dedicación parcial. El perfil es representativo del rol: son quienes hoy realizan la coordinación de evaluaciones por los canales descritos en el capítulo 1, y ninguno participó en el desarrollo del sistema. Se les entregó una guía escrita con tres tareas secuenciales: crear una oferta de asignatura para el semestre de prueba, crear un examen individual para dicha oferta, y realizar una importación masiva de evaluaciones mediante un archivo Excel, resolviendo los conflictos que el sistema señalara durante la revisión. Al finalizar las tareas, cada participante completó el cuestionario SUS.

4.7.2. Resultados: Jefes de Carrera

Los puntajes SUS individuales, calculados según el método estándar, fueron 85,0, 67,5 y 67,5, lo que da un promedio de 73,3 sobre 100 con una desviación estándar de 10,1 puntos. El promedio supera el criterio de aceptación de 68 fijado a priori y se ubica en la banda “Bueno” de la escala de adjetivos correspondiente [Bangor *et al.*, 2009], aunque la dispersión merece nota: el promedio se sostiene en un puntaje individual alto acompañado de dos puntajes que quedan justo bajo el umbral, por lo que el resultado se lee como una aceptación positiva pero no unánime.

Al observar las respuestas agrupadas por categoría temática del cuestionario se aprecian patrones consistentes entre los tres participantes.

- **Aprendizaje:** los tres coincidieron unánimemente, con el puntaje máximo, en que la mayoría de las personas aprendería a usar el sistema rápidamente, y las respuestas sobre la necesidad de apoyo técnico y de aprender muchas cosas antes de poder usarlo fueron uniformemente bajas, lo que sugiere una curva de aprendizaje baja.
- **Facilidad de uso y complejidad:** las respuestas fueron mayoritariamente positivas aunque menos unánimes; los tres coincidieron en que el sistema era fácil de usar, pero uno de los participantes lo calificó como comparativamente más complicado que los otros

dos, lo cual es consistente con el puntaje SUS más bajo de ese mismo participante y con la retroalimentación cualitativa descrita a continuación.

- **Integración y consistencia:** se observaron las respuestas comparativamente más moderadas; si bien ningún participante reportó inconsistencia severa, el grado de acuerdo con que las distintas funcionalidades estaban bien integradas fue menor que en el resto de las categorías, posicionándola como la dimensión con mayor margen de mejora relativo dentro de un resultado globalmente positivo.
- **Frecuencia de uso y confianza:** los participantes reportaron sentirse seguros usando la plataforma y expresaron disposición a usarla con frecuencia, lo que indica una buena aceptación general del sistema por parte de este rol.

4.7.3. Resultados: Personal Académico (ayudantes)

Se aplicó el mismo instrumento a un segundo grupo, de cuatro participantes, con el rol de personal académico. A diferencia del anterior, su perfil de uso se concentra en una sola tarea: crear evaluaciones individuales para una asignatura y paralelo previamente configurados por un administrador, sin gestionar ofertas de asignatura ni importaciones masivas. El grupo estuvo compuesto por cuatro ayudantes de asignatura, todos mayores de 25 años: dos mujeres, una estudiante de pregrado y una ingeniera, y dos hombres, ambos estudiantes de pregrado. Ninguno ejerce en el Departamento de Informática ni participó en el desarrollo del sistema. Se les entregó una guía acotada a la tarea de crear una o más evaluaciones para el paralelo asignado, seguida del mismo cuestionario SUS.

Los puntajes SUS individuales de los cuatro participantes fueron 87,5, 90,0, 92,5 y 95,0, lo que da un promedio de 91,25 sobre 100 con una desviación estándar de 3,2 puntos. Los cuatro superan el criterio de aceptación de 68 con amplitud y el promedio se ubica en la banda “Excelente” de la escala de adjetivos correspondiente [Bangor *et al.*, 2009]. La dispersión es baja, lo que indica una aceptación homogénea dentro del grupo.

Al observar las respuestas agrupadas por categoría temática, el patrón es uniformemente positivo, consistente con lo acotado de la tarea.

- **Aprendizaje:** los cuatro coincidieron en que la mayoría de las personas aprendería a usar el sistema rápidamente, y las respuestas sobre la necesidad de apoyo técnico y sobre tener que aprender muchas cosas antes de usarlo fueron uniformemente bajas, lo que sugiere una curva de aprendizaje muy baja para esta tarea.
- **Facilidad de uso y complejidad:** el acuerdo con que el sistema es fácil de usar fue unánime y máximo, y ninguno lo consideró innecesariamente complejo ni complicado de usar.

- **Integración y consistencia:** el grado de acuerdo con que las funcionalidades estaban bien integradas fue alto y la percepción de inconsistencia, baja.
- **Frecuencia de uso y confianza:** los participantes se sintieron seguros usando la plataforma y expresaron disposición a usarla con frecuencia.

4.7.4. Resultados: Personal Académico (profesores)

Un tercer grupo, también con el rol de personal académico, repitió la evaluación en una sesión posterior con una variante en la tarea. Además de crear evaluaciones para su paralelo asignado, se les pidió consultar el calendario para localizar las evaluaciones recién creadas, de modo que la sesión ejercita en el nivel de aceptación tanto la creación de evaluaciones (RF-5) como la consulta del calendario y sus vistas (RF-7, RF-9) y las alertas de sobrecarga (RF-8), que hasta aquí contaban solo con validación funcional. El grupo estuvo compuesto por cuatro participantes: dos mujeres mayores de 40 años, ambas con dedicación docente, y dos hombres mayores de 30 años, un ingeniero con experiencia previa como ayudante y un ingeniero con dedicación docente parcial. Ninguno ejerce en el Departamento de Informática ni participó en el desarrollo del sistema.

Los puntajes SUS individuales fueron 87,5, 92,5, 97,5 y 100,0, con un promedio de 94,4 sobre 100 y una desviación estándar de 5,5 puntos. Los cuatro superan el umbral con amplitud y el promedio se ubica también en la banda “Excelente”. El patrón por categoría reprodujo el del grupo de ayudantes: acuerdo unánime y máximo en que la mayoría aprendería a usar el sistema con rapidez, alta facilidad de uso sin percepción de complejidad, buena integración con inconsistencia percibida casi nula, y disposición a usarlo con frecuencia.

La parte de la tarea dedicada al calendario dejó una observación cualitativa relevante. Una vez ubicado el punto de acceso, los participantes localizaron sin dificultad las evaluaciones en la vista, pero varios no dieron de inmediato con el selector de asignatura ubicado a un costado de la pantalla, de modo que al principio no encontraban las evaluaciones que acababan de crear. Los participantes describieron el obstáculo sin proponer una solución; incorporar una guía en pantalla, análoga a la que acompaña la importación, es la respuesta que de él se desprende. El puntaje alto convive así con un obstáculo concreto de navegación, que se recoge entre las mejoras de la sección 4.8.

4.7.5. Lectura conjunta de los resultados

Los tres grupos superan el criterio de aceptación fijado a priori, con promedios de 73,3 para los Jefes de Carrera, 91,25 para el grupo de ayudantes y 94,4 para el de profesores. Los dos grupos de personal académico, cuya prueba se concentró en la creación de evaluaciones, calificaron el sistema notablemente más alto que los Jefes de Carrera. Dos lecturas admite esa diferencia y los datos no permiten separarlas. La primera atiende al alcance de la tarea:

el protocolo de los Jefes de Carrera recorrió la mayor superficie de la aplicación, incluidas la creación de ofertas y la importación masiva con resolución de conflictos, mientras que el personal académico ejerció una tarea acotada, de modo que a mayor porción del sistema ejercida, más pesó su complejidad sobre la usabilidad percibida. La segunda es que las funciones que solo el Jefe de Carrera utiliza sean efectivamente más difíciles de usar, en cuyo caso el desfase señala dónde el sistema debe mejorar. Ambas apuntan al mismo lugar, y las mejoras que la sección 4.8 recoge se concentran justamente ahí. El alcance de la evidencia conviene precisarlo. Con tres, cuatro y cuatro participantes por grupo, el instrumento entrega un indicador de aceptación, no una estimación del puntaje poblacional. Además, solo el grupo de Jefes de Carrera está compuesto por integrantes del departamento donde el sistema se implantaría; los otros dos ejercen sus roles fuera de él y participaron como representantes del perfil, de manera que su puntaje habla de la usabilidad de la tarea y no de la adopción por parte del departamento. Dentro de ese alcance, los tres grupos superan el umbral en su promedio y nueve de los once participantes lo superan individualmente. Los dos que no lo alcanzan pertenecen al grupo de Jefes de Carrera, que es a la vez el más pequeño, el de tarea más amplia y el único formado por usuarios del departamento, por lo que su resultado, positivo pero ajustado, es el que conviene leer con más cuidado: el requerimiento de usabilidad (RNF-1) se cumple, y el margen con que lo hace es estrecho precisamente en el rol que más superficie del sistema utiliza.

4.8. Hallazgos de la validación

4.8.1. Un defecto expuesto por el uso real

El aporte más valioso de la validación con usuarios no salió del cuestionario. Al revisar después de las sesiones los datos que los participantes habían creado, se encontraron evaluaciones de un mismo paralelo cruzadas en el horario, un estado que el sistema debía rechazar y que nadie advirtió mientras lo producía. La condición se comprobaba entonces desde la aplicación antes de escribir, de manera que quedaba sin efecto por dos vías: un camino de escritura que no la invocara, como ocurría en la importación masiva, y dos solicitudes simultáneas que la superaran ambas antes de que cualquiera insertara. La corrección declaró la invariante como una restricción de exclusión de PostgreSQL, que el motor verifica en toda escritura y que ningún camino de código puede omitir; la aplicación traduce su violación a una respuesta HTTP 409, y a un error legible por fila en la importación. Después del cambio, el estado observado en las sesiones dejó de ser registrable por ninguno de los caminos. El episodio precisa además el aporte de cada nivel de la metodología: las pruebas automatizadas y los escenarios funcionales verificaban el rechazo allí donde la comprobación se ejecutaba, y fue el uso real, con datos creados fuera del guion de una prueba, el que expuso los caminos donde no ocurría.

4.8.2. Mejoras de usabilidad detectadas

Uno de los participantes del grupo de Jefes de Carrera complementó el cuestionario con retroalimentación cualitativa, y la observación de las sesiones, junto con los comentarios del personal académico, aportó puntos adicionales. En conjunto, se identificaron las siguientes mejoras:

- Consultó si el sistema podría integrarse con SIGA, dado que gran parte de la información gestionada (asignaturas, paralelos, profesores) también existe en dicho sistema institucional.
- En la pantalla de inicio, al seleccionar muchas asignaturas estas se agrupan en la parte superior ocupando espacio considerable; sugirió reemplazar esta agrupación por una lista desplegable.
- En la misma pantalla, sugirió destacar con un color distintivo los eventos del calendario correspondientes a la asignatura seleccionada, para facilitar su ubicación visual.
- En la vista de calendario, varios participantes del personal académico no ubicaron de inmediato el selector de asignatura que permite filtrar y localizar las evaluaciones propias, por lo que al principio no encontraban las que acababan de crear. De esa dificultad se desprende la conveniencia de incorporar una guía en pantalla, análoga a la que acompaña la importación.
- En la gestión de exámenes coordinados, no resultó evidente cómo registrar las salas de todos los paralelos. La observación de las sesiones precisó la causa: un examen coordinado admite una única sala, cuando en la práctica se rinde en varias a la vez. Se propuso además evaluar una lista de aulas administrable desde el sistema.
- Las evaluaciones periódicas, como los controles quincenales, deben crearse una a una; se planteó incorporar un mecanismo de recurrencia o de duplicación.
- Un examen coordinado, por aplicar a todos los paralelos de la asignatura, debería señalarse en cruce con una evaluación registrada para un paralelo específico de la misma asignatura, cosa que el calendario no hacía. La advertencia se incorporó después de las sesiones.
- En el flujo de importación, solicitó una explicación más detallada de la plantilla Excel, campo por campo.
- En administración, sugirió controlar el largo de los nombres y siglas de las asignaturas.
- No encontró dónde visualizar las mallas curriculares de los programas, aun cuando el sistema permite configurarlas como parámetro.

Estos puntos no invalidan los resultados cuantitativos del cuestionario: delimitan el trabajo futuro con la precisión que solo el uso real produce. Varios de ellos, en particular los relativos a los exámenes coordinados, tienen su raíz en un mismo límite del modelo de datos, el registro del examen coordinado como una única evaluación, que el capítulo 5 recoge entre los límites del sistema.

CAPÍTULO 5

CONCLUSIONES

5.1. Resumen del trabajo realizado

Esta memoria abordó el problema de la planificación de evaluaciones académicas en el Departamento de Informática de la UTFSM, caracterizado por su naturaleza descentralizada y la ausencia de visibilidad compartida entre los actores involucrados. La falta de un registro unificado derivaba en semanas con alta concentración de evaluaciones, dificultad de intervención por parte de los jefes de carrera e imposibilidad de coordinación a la escala del departamento.

Para resolver este problema se diseñó, desarrolló e implementó un sistema de coordinación de evaluaciones accesible como aplicación web. El sistema centraliza la información de evaluaciones de todas las asignaturas activas en un semestre y la expone a través de una vista de calendario unificada, accesible sin autenticación para los estudiantes. Se definieron cuatro roles de usuario con distintos niveles de acceso: administrador global, jefe de carrera, personal académico y usuario público no autenticado.

El desarrollo abarcó el diseño de una base de datos relacional con doce tablas que modelan el dominio académico, un back-end desarrollado en Go con autenticación basada en JWT y cookies `httpOnly`, una API REST organizada en grupos de rutas según el nivel de autenticación que requieren, una infraestructura de despliegue reproducible basada en Docker Compose, y una interfaz web desarrollada con Vue 3 y TypeScript. Se incluyó además un flujo de importación masiva de evaluaciones desde archivos Excel, con una etapa de revisión y resolución asistida por matching automático, orientado a reducir el trabajo manual de carga al inicio de cada semestre.

5.2. Objetivos realizados

Objetivo general. Se diseñó, desarrolló e implementó un sistema de coordinación que proporciona una vista global semestral unificada de la carga académica de los estudiantes del Departamento de Informática de la UTFSM. El sistema opera sobre el catálogo académico real del departamento; las evaluaciones con las que se ejercitó fueron generadas para poblar los semestres de prueba, según se detalla en el capítulo 4.

Objetivo específico 1. El proceso actual de planificación de evaluaciones fue investigado y documentado, identificando los actores involucrados, las dos modalidades de coordinación existentes y los problemas recurrentes que el sistema debe resolver.

Objetivo específico 2. Se implementó la carga de las evaluaciones de las asignaturas impartidas cada semestre mediante tres mecanismos complementarios: creación individual mediante formulario, importación masiva desde archivos Excel con revisión asistida, y administración del catálogo académico desde el panel de administración. Sobre las evaluaciones cargadas, el sistema detecta y señala automáticamente los periodos de alta carga académica, clasificando cada coincidencia según su severidad (tope de horario, mismo día o misma semana) y presentando el detalle al usuario al interactuar con la alerta.

Objetivo específico 3. Se implementó una vista global de evaluaciones accesible sin autenticación, con vistas mensual y semanal y filtros por semestre, malla y semestre curricular, que permite a estudiantes y profesores consultar en un solo lugar las evaluaciones del conjunto de asignaturas que seleccionen, provengan o no de un mismo plan de estudio.

Más allá de los objetivos, el capítulo 4 traza cada requerimiento del sistema al nivel de validación que aporta su evidencia. La usabilidad exigida por el requerimiento RNF-1 se evaluó con usuarios mediante el cuestionario System Usability Scale, en tres grupos que promediaron 73,3 puntos sobre 100 los Jefes de Carrera, y 91,25 y 94,4 los dos grupos de personal académico, los tres por sobre el criterio de aceptación de 68 fijado de antemano. Solo el primer grupo lo integran docentes del propio departamento; los otros dos ejercen esos roles fuera de él y participaron como representantes del perfil de uso, según se detalla en el capítulo 4. La misma validación identificó mejoras concretas de usabilidad y expuso un defecto de integridad que las pruebas del desarrollador no habían encontrado, cuya corrección, trasladar la invariante al esquema de la base de datos, quedó documentada en ese capítulo.

5.3. Límites

Ausencia de integración con el sistema universitario. El sistema no tiene conexión con el Sistema de Información y Gestión Académica (SIGA) de la universidad. Esto significa que la asignación de asignaturas a estudiantes no se obtiene automáticamente: los calendarios personalizados dependen de la selección manual del propio usuario, y la carga de asignaturas al sistema requiere importación desde archivos externos o ingreso manual al inicio de cada semestre.

Alcance departamental. El sistema fue diseñado y validado en el contexto del Departamento de Informática. Su extensión a otros departamentos requeriría la adopción administrativa correspondiente y la configuración del catálogo para cada unidad, pero la arquitectura del sistema no impone restricciones técnicas para ello.

Detección de conflictos basada en selección manual. La detección de conflictos opera sobre las asignaturas que el estudiante selecciona explícitamente en el calendario, no sobre su matrícula real. Un estudiante que no conozca todas sus asignaturas activas o no las haya seleccionado no recibirá alertas completas.

Registro de cruces entre un examen coordinado y evaluaciones de paralelo. Un examen coordinado se registra como una única evaluación sin paralelo asociado, y la restricción del esquema compara los paralelos por igualdad, de modo que no impide registrar a la misma hora un examen coordinado y una evaluación de un paralelo de la misma asignatura. El calendario sí advierte del cruce, señalando que la validación motivó y que se incorporó después de las sesiones, por lo que el caso queda a la vista de quien consulta aunque el registro lo admita.

Resolución manual en importaciones ambiguas. El algoritmo de matching automático en el flujo de importación reduce significativamente el trabajo manual, pero no elimina la necesidad de intervención humana cuando los datos del archivo fuente son ambiguos o inconsistentes con el catálogo del sistema.

5.4. Trabajo futuro

Integración con SIGA. La mejora de mayor impacto sería integrar el sistema con SIGA para obtener automáticamente los datos de matrícula de los estudiantes. Esto permitiría generar calendarios personalizados basados en la inscripción real de asignaturas, eliminar la necesidad de selección manual y ofrecer alertas precisas sin intervención del usuario.

Notificaciones de cambios. Implementar un sistema de notificaciones que alerte a los usuarios cuando se agreguen, modifiquen o eliminen evaluaciones en las asignaturas de su interés. Esto reduciría la necesidad de revisar activamente el calendario para mantenerse informado de cambios de última hora.

Notificaciones push de alertas de conflicto. Actualmente las alertas de conflicto y sobrecarga son visuales: el sistema las resalta en el calendario, pero solo se hacen visibles cuando el usuario lo consulta activamente. Una extensión natural sería incorporar notificaciones push, por correo electrónico o al teléfono, que avisen proactivamente a jefes de carrera, profesores y estudiantes apenas el sistema detecte un conflicto o una semana de sobrecarga, sin que estos deban revisar el calendario manualmente para enterarse.

Ponderación de evaluaciones según su peso académico real. El sistema actual trata todas las evaluaciones como equivalentes al momento de determinar la severidad de una semana de alta carga, sin considerar que no todas representan el mismo esfuerzo o impacto para el estudiante. Una extensión futura podría calcular un peso para cada evaluación a partir de los créditos SCT (Sistema de Créditos Transferibles) de la asignatura y el porcentaje que esa evaluación representa de la nota final de la asignatura, permitiendo distinguir entre semanas con varias evaluaciones de bajo impacto y semanas con pocas evaluaciones de alto impacto académico, y priorizar las alertas en consecuencia.

Análisis histórico de carga académica. Con el registro acumulado de evaluaciones de múltiples semestres, el sistema podría ofrecer visualizaciones históricas que permitan a los jefes

de carrera identificar semanas de alta carga recurrentes, fundamentando decisiones de política académica con datos concretos en lugar de percepciones.

Extensión a otros departamentos. Generalizar la adopción del sistema a otros departamentos de la universidad que enfrentan el mismo problema de coordinación, en particular aquellos cuyas asignaturas son cursadas simultáneamente por estudiantes de múltiples programas.

5.5. Reflexión final

Este capítulo cierra con una lista de trabajo futuro, y conviene precisar qué significa que la tenga. El software rara vez alcanza un estado terminal en el que no quede nada por mejorar: mientras un sistema se usa, el uso sigue mostrando aristas que ningún diseño anticipa por completo, y cada mejora que se incorpora abre a su vez la posibilidad de la siguiente. Las mejoras de usabilidad y las funcionalidades que la validación dejó a la vista, recogidas en la sección 4.8, no son por lo tanto defectos que la construcción pasara por alto, sino el resultado esperable de poner el sistema frente a quienes habrían de usarlo. Un proyecto que no las produjera diría menos sobre la calidad de lo construido que sobre lo poco que se lo expuso al uso real.

Ese es, en rigor, el ciclo que esta memoria documenta. Se caracterizó un problema a partir de cómo se planifican hoy las evaluaciones, se derivaron requerimientos de ese diagnóstico, se construyó un sistema sobre ellos, se lo puso en manos de usuarios y se corrigió aquello que el contacto con el uso reveló, incluido un defecto de integridad que ninguna prueba escrita por el desarrollador había encontrado. Lo que queda al final no es un producto cerrado, sino uno en condiciones de seguir evolucionando: con su problema entendido, sus decisiones justificadas y sus límites conocidos y escritos, que es la forma en que el software sostiene su utilidad más allá del momento en que se entrega.

Para quien lo desarrolló, el trabajo fue además un ejercicio de ingeniería antes que uno de programación. Obligó a decidir con información incompleta, a justificar cada elección técnica frente a alternativas razonables y a hacerse cargo de sus costos, a distinguir lo que el sistema garantiza de lo que simplemente no ha fallado todavía, y sobre todo a constatar que la distancia entre un sistema que funciona en el equipo de quien lo escribe y uno que funciona en manos de otras personas solo se recorre poniéndolo en esas manos.

REFERENCIAS BIBLIOGRÁFICAS

- [Bangor *et al.*, 2009] Bangor, A., Kortum, P. T., y Miller, J. T. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of Usability Studies*, 4(3):114–123.
- [Brooke, 1996] Brooke, J. (1996). SUS: A quick and dirty usability scale. En Jordan, P. W., Thomas, B., Weerdmeester, B. A., y McClelland, I. L., editores, *Usability Evaluation in Industry*, pp. 189–194. Taylor & Francis, London.
- [Cepeda *et al.*, 2006] Cepeda, N. J., Pashler, H., Vul, E., Wixted, J. T., y Rohrer, D. (2006). Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychological Bulletin*, 132(3):354–380.
- [Chaplin *et al.*, 2025] Chaplin, C., Gai, S., Gutekunst, S. C., Miyake, T., Serra, T., Snyder, L., Tran, V., y Waddell, L. (2025). Final exam scheduling at Bucknell university: A case study and open-source tool. arXiv:2509.11031. <https://arxiv.org/abs/2509.11031>.
- [Credé y Kuncel, 2008] Credé, M. y Kuncel, N. R. (2008). Study habits, skills, and attitudes: The third pillar supporting collegiate academic performance. *Perspectives on Psychological Science*, 3(6):425–453.
- [Docker Inc., 2026] Docker Inc. (2026). Docker overview. Documentación oficial. Consultado en 2026. <https://docs.docker.com/get-started/docker-overview/>.
- [F5, Inc., 2026] F5, Inc. (2026). nginx. Sitio oficial. Consultado en 2026. <https://nginx.org/>.
- [Google, 2026] Google (2026). The go programming language. Documentación oficial. Consultado en 2026. <https://go.dev/>.
- [Jones *et al.*, 2015] Jones, M. B., Bradley, J., y Sakimura, N. (2015). JSON Web Token (JWT). RFC 7519, Internet Engineering Task Force. <https://datatracker.ietf.org/doc/html/rfc7519>.
- [Krawczyk *et al.*, 1997] Krawczyk, H., Bellare, M., y Canetti, R. (1997). HMAC: Keyed-Hashing for Message Authentication. RFC 2104, Internet Engineering Task Force. <https://datatracker.ietf.org/doc/html/rfc2104>.
- [Lalescu, 2026] Lalescu, L. (2026). FET – free timetabling software. Sitio oficial. Consultado en 2026. <https://lalescu.ro/liviu/fet/>.
- [Mallari *et al.*, 2023] Mallari, C. B., San Juan, J. L., y Li, R. (2023). The university coursework timetabling problem: An optimization approach to synchronizing course calendars. *Computers & Industrial Engineering*, 184:109561.

- [McCollum, 2007] McCollum, B. (2007). A perspective on bridging the gap between theory and practice in university timetabling. En *Practice and Theory of Automated Timetabling VI*, volumen 3867 de *Lecture Notes in Computer Science*, pp. 3–23. Springer, Berlin, Heidelberg.
- [Microsoft, 2026] Microsoft (2026). Typescript. Sitio oficial. Consultado en 2026. <https://www.typescriptlang.org/>.
- [Moodle Pty Ltd, 2026] Moodle Pty Ltd (2026). Calendar — MoodleDocs. Documentación oficial. Consultado en 2026. <https://docs.moodle.org/en/Calendar>.
- [Oude Vrielink et al., 2019] Oude Vrielink, R. A., Jansen, E. A., Hans, E. W., y van Hillegersberg, J. (2019). Practices in timetabling in higher education institutions: a systematic review. *Annals of Operations Research*, 275(1):145–160.
- [PostgreSQL Global Development Group, 2026] PostgreSQL Global Development Group (2026). PostgreSQL documentation – what is PostgreSQL? Documentación oficial. Consultado en 2026. <https://www.postgresql.org/docs/current/intro-what-is.html>.
- [PrimeTek Informatics, 2026] PrimeTek Informatics (2026). Introduction — primevue. Documentación oficial. Consultado en 2026. <https://primevue.org/introduction>.
- [Provos y Mazières, 1999] Provos, N. y Mazières, D. (1999). A future-adaptable password scheme. En *Proceedings of the 1999 USENIX Annual Technical Conference*, Monterey, CA, USA.
- [Qu et al., 2009] Qu, R., Burke, E. K., McCollum, B., Merlot, L. T. G., y Lee, S. Y. (2009). A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1):55–89.
- [Richardson et al., 2012] Richardson, M., Abraham, C., y Bond, R. (2012). Psychological correlates of university students' academic performance: A systematic review and meta-analysis. *Psychological Bulletin*, 138(2):353–387.
- [Sauro, 2011] Sauro, J. (2011). *A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices*. Measuring Usability LLC, Denver, CO.
- [UniTime, LLC, 2026] UniTime, LLC (2026). UniTime – comprehensive university timetabling system. Documentación oficial. Consultado en 2026. <https://www.unitime.org/>.
- [Universidad Técnica Federico Santa María, 2026] Universidad Técnica Federico Santa María (2026). SIGA – sistema de información para la gestión académica, oferta de asignaturas. Plataforma de uso interno. Requiere autenticación institucional. Consultado en 2026. <https://siga.usm.cl>.
- [Vázquez et al., 2012] Vázquez, C. M., Cavallo, M. A., Aparicio, S. N., Muñoz, B. L., Robson, C. M., Ruíz, L. I., Secreto, M. F., Sepiarsky, P. A., y Escobar, M. E. (2012). Factores de impacto en el rendimiento académico universitario. un estudio a partir de las percepciones de los estudiantes.

[You y Vite Contributors, 2026] You, E. y Vite Contributors (2026). Getting started — vite. Documentación oficial. Consultado en 2026. <https://vite.dev/guide/>.

[You y Vue Router Contributors, 2026] You, E. y Vue Router Contributors (2026). Guide — vue router. Documentación oficial. Consultado en 2026. <https://router.vuejs.org/guide/>.

[You y Vue.js Contributors, 2026] You, E. y Vue.js Contributors (2026). Introduction — vue.js. Documentación oficial. Consultado en 2026. <https://vuejs.org/guide/introduction.html>.

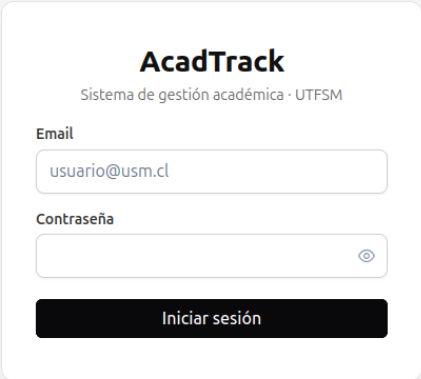
ANEXO A: MANUAL DE USUARIO

Este anexo describe el uso del sistema organizado por rol, desde el perfil de menor al de mayor alcance. Las capacidades son acumulativas: el personal académico puede hacer todo lo que hace el usuario público, y el Jefe de Carrera todo lo que hace el personal académico, restringido a los departamentos que administra.

A.1 Acceso al sistema

El calendario de evaluaciones es público: consultarlo no requiere cuenta ni inicio de sesión. Las cuentas existen solo para quienes registran información en el sistema (personal académico, jefes de carrera y administradores) y las crea un administrador; no hay registro autónomo de usuarios.

El sistema está desplegado como una instalación de prueba, a la que se accede por la dirección IP del servidor en el navegador: no se le asignó un nombre de dominio, de modo que la dirección la entrega el administrador junto con las credenciales. Al abrirla sin una sesión iniciada se llega al calendario público; el enlace de inicio de sesión conduce al formulario de la figura 37, donde se ingresan el correo y la contraseña asignados. La sesión queda activa en el navegador hasta cerrarla desde el menú de usuario o hasta su expiración automática, tras lo cual el sistema vuelve a pedir las credenciales.



AcadTrack
Sistema de gestión académica · UTFSM

Email

Contraseña

Figura 37: Formulario de inicio de sesión.
Fuente: Elaboración propia.

A.2 Consulta del calendario (usuario público)

Al abrir el sistema se muestra el calendario del semestre activo (figura 32). La consulta habitual sigue estos pasos:

1. **Elegir el semestre.** El selector de semestre (figura 33) permite cambiar entre los semestres visibles; las asignaturas ya seleccionadas se conservan al cambiar.
2. **Seleccionar asignaturas.** El buscador permite agregar asignaturas por nombre o sigla. Cada asignatura agregada incorpora sus evaluaciones al calendario.
3. **Seleccionar por malla.** El panel de mallas (figura 35) agrega de una vez todas las asignaturas de un semestre curricular de una malla, que es la forma recomendada para un estudiante de obtener su semestre completo en un paso.
4. **Leer el calendario.** La vista mensual muestra las evaluaciones del conjunto seleccionado; la vista semanal (figura 34) agrega el detalle de horario y sala de cada una.
5. **Interpretar las alertas.** Las semanas con acumulación de evaluaciones quedan señaladas en el propio calendario. Al abrir el detalle de una semana (figura 36), las coincidencias aparecen agrupadas por severidad: evaluaciones que se cruzan en el horario, evaluaciones el mismo día y evaluaciones en la misma semana.

A.3 Registro de evaluaciones (personal académico)

El personal académico crea y mantiene las evaluaciones de los paralelos que tiene asignados, desde la página *Gestión*.

1. **Ubicar el paralelo.** La página lista las evaluaciones del semestre activo y ofrece filtros por departamento, asignatura y estado de eliminación (figura 27).
2. **Crear una evaluación.** El botón de creación abre un formulario (figura 28) que pide el nombre de la evaluación, la fecha y hora de inicio y de término, y opcionalmente la sala y notas. Las fechas deben caer dentro del semestre correspondiente; el selector de fechas ya acota el rango.
3. **Si el sistema rechaza el horario.** Cuando la evaluación se cruza en el horario con otra ya registrada para el mismo paralelo, el sistema no la guarda y lo indica sin cerrar el formulario (figura 29); basta corregir el horario y reintentar.
4. **Editar o eliminar.** Cada evaluación del listado puede editarse con el mismo formulario o eliminarse previa confirmación (figura 30). La eliminación es reversible: activando el filtro *Mostrar eliminados*, la evaluación aparece marcada y puede restaurarse (figura 31).

A.4 Gestión de un departamento (Jefe de Carrera)

El Jefe de Carrera administra el catálogo de su departamento y dispone de la carga masiva de evaluaciones.

A.4.1 Ofertas de asignatura y paralelos

Antes de registrar evaluaciones, la asignatura debe estar ofertada en el semestre: la pestaña *Ofertas* del panel permite crear la oferta de una asignatura para un semestre, agregar sus paralelos y asignar al coordinador y a los profesores (figura 38). La oferta se define eligiendo departamento, asignatura y semestre; el panel derecho agrega los paralelos, cada uno con su código y su profesor. Los campos de coordinador y profesor aceptan texto libre, de modo que puede registrarse a docentes que no tienen cuenta en el sistema.

OFERTAS Y PARALELOS

Ofertas de asignaturas
Crea asignaturas por semestre y gestiona sus paralelos.

PARALELOS
Agrega códigos de paralelo y profesores.

Departamento: Informática

Asignatura: INF285 — COMPUTACIÓN CIENTÍFICA

Semestre: 2026-2 (Prueba)

Selección oferta existente: Nueva oferta

☐ Asignatura coordinada

Coordinador (opcional): Buscar por nombre...

Crear oferta

Código: 01, A, ...

Profesor (opcional): Buscar o texto libre...

Nueva oferta

Agregar paralelo

Figura 38: Creación de una oferta de asignatura y de sus paralelos.
Fuente: Elaboración propia.

A.4.2 Importación masiva desde Excel

La carga de un semestre completo se realiza importando un archivo Excel con una fila por evaluación. La tabla 10 describe las columnas de la plantilla.

Columna	Obligatoria	Contenido
Semestre	Sí	Semestre de destino, por ejemplo 2026-1.
Departamento	Sí	Nombre del departamento que dicta la asignatura.
Nombre de Asignatura	Sí*	Nombre oficial de la asignatura.
Código de Asignatura	Sí*	Sigla oficial de la asignatura.
Paralelo	No	Código del paralelo; vacío para una evaluación coordinada de todos los paralelos.
Coordinador	No	Nombre del coordinador de la asignatura.
Profesor	No	Nombre del profesor del paralelo.
Nombre Evaluación	No	Por ejemplo Certamen 1.
Fecha Evaluación	Sí	Fecha de la evaluación.
Hora Evaluación	Sí	Hora de inicio.
Duración de Evaluación (min)	Sí	Duración; determina la hora de término.
Sala (opcional)	No	Sala o modalidad.
Comentario/Nota (opcional)	No	Nota libre.

Tabla 10: Columnas de la plantilla de importación, en el orden de la plantilla oficial. El analizador normaliza mayúsculas y tildes al leer el encabezado, de modo que admite variantes en su escritura. *Basta uno de los dos identificadores de la asignatura, aunque se recomienda incluir ambos.

Fuente: Elaboración propia.

El flujo de importación tiene tres etapas:

1. **Carga.** Seleccionar el archivo y el semestre de destino (figura 19). El sistema confirma cuántas filas reconoció (figura 20); ninguna se escribe todavía en el calendario.
2. **Revisión.** Las filas quedan agrupadas por asignatura, cada una con un estado (figura 21): *Lista* cuando todos sus datos coinciden con el catálogo, *Revisar* cuando hay más de una coincidencia posible y hay que elegir, y *Bloqueada* cuando falta un dato, indicando cuál. Una fila bloqueada se resuelve desde su formulario (figura 22), que permite completar el dato faltante o crear en el acto la asignatura o el paralelo ausentes del catálogo.
3. **Confirmación.** Al seleccionar las filas e importar, el sistema valida primero el conjunto y reporta cuáles se importarán y cuáles fallarían y por qué (figura 23), lo que permite corregir o descartar antes de escribir. El resultado final indica el detalle por fila (figuras 24, 25 y 26); solo las filas exitosas pasan al calendario, y una fila fallida puede corregirse y confirmarse después sin repetir la carga del archivo.

A.5 Administración global

El administrador global mantiene los objetos comunes a toda la institución desde el panel de administración:

- **Usuarios:** creación de cuentas y asignación de departamentos a los jefes de carrera (figura 39). El buscador ubica una cuenta existente para editarla; el formulario crea una nueva con una contraseña temporal y marca si es administrador global o personal académico. Los departamentos se asignan seleccionándolos de la lista, y un usuario puede administrar más de uno.
- **Semestres:** creación del semestre con las fechas de inicio y término que acotan las evaluaciones (figura 40). El semestre activo es el que el calendario público muestra por omisión, y uno oculto deja de ofrecerse en el selector.
- **Departamentos:** mantención del catálogo de departamentos y de sus códigos (figura 41). Un departamento puede tener más de un código, que es lo que permite reconocerlo en las siglas de asignatura durante la importación.
- **Mallas curriculares:** definición de los programas de estudio y de las asignaturas que componen cada semestre curricular, que alimentan el panel de mallas del calendario público. El listado (figura 42) muestra las mallas registradas con su código, departamento, número de semestres y visibilidad; *Editar* modifica esos datos y *Cursos* abre la asignación de asignaturas (figura 43). Cada fila de esa asignación une una asignatura del catálogo con el número de semestre que le corresponde dentro de la malla, y es esa correspondencia la que permite al calendario público agregar de una vez todas las asignaturas de un semestre curricular.

USUARIOS
Gestiona usuarios
Busca, edita o crea usuarios y asigna roles de departamento.

Limpiar

Buscar por nombre o email
Profesor Test Informática 1 [Buscar]

Nombre completo
Nombre y apellido

Email
usuario@dominio.cl

Contraseña
Contraseña temporal

☐ Administrador global ☐ Profesor / TA

DEPARTAMENTOS DE ADMINISTRACIÓN
El usuario podrá gestionar exámenes e importaciones en estos departamentos.

Defider Dirección General de Docencia Estudios Humanísticos Física Industrias Informática
Ingeniería Comercial Ingeniería Eléctrica Ingeniería Mecánica Ingeniería Química y Ambiental
Matemática Obras Civiles Química

Crear usuario

Figura 39: Administración de usuarios y asignación de departamentos.

Fuente: Elaboración propia.

SEMESTRES

Gestión de semestres

Crea y edita los semestres disponibles.

Selección semestre

Nuevo semestre

2026-2 (Prueba)

2026-1 (Ejemplo)

Nombre

2025-1, 2025-2

Inicio

mm / dd / yyyy

Término

mm / dd / yyyy

☐ Activo

☐ Oculto

Crear semestre

Figura 40: Gestión de semestres.
Fuente: Elaboración propia.

DEPARTAMENTOS

Registro de departamentos

Crea departamentos y administra sus códigos.

Selección existente

Nuevo departamento

Defider

Dirección General de Docencia

Estudios Humanísticos

Física

Industrias

Nombre del departamento

Informática, Matemáticas...

Agregar código (INF, MAT...)

Agregar

Crear departamento

Figura 41: Registro de departamentos y de sus códigos.
Fuente: Elaboración propia.

MALLAS

Gestión de mallas

Crea y edita mallas curriculares y sus asignaturas por semestre.

Código

Nombre

Departamento

Semestres

Estado

Acciones

IC

Ing. Civil

Obras Cíviles

10

Visible

Editar Cursos

ICI

Ing. Civil Informática

Informática

10

Visible

Editar Cursos

ICI-ANT

Ing. Civil Informática (antiguo)

Informática

11

Visible

Editar Cursos

ICIND

Ing. Civil Industrial

Industrias

10

Visible

Editar Cursos

ICM

Ing. Civil Mecánica

Ingeniería Mecánica

10

Visible

Editar Cursos

ICQ

Ing. Civil Química

Ingeniería Química y Ambiental

10

Visible

Editar Cursos

Figura 42: Listado de mallas curriculares registradas.
Fuente: Elaboración propia.

Cursos — ICI-ANT ×

Asigna cursos al programa especificando el semestre en la malla.

SEMESTRE	ASIGNATURA	
1	DEW100 — EDUCACION FISICA I (DAMAS Y VARONES)	
1	FIS100 — INTRODUCCION A LA FISICA	
1	IWI131 — PROGRAMACION	
1	MAT021 — MATEMATICA I	
2	DEW101 — EDUCACION FISICA II (DAMAS Y VARONES)	
2	FIS110 — FISICA GENERAL I	
2	IWG101 — INTRODUCCION A LA INGENIERIA	
2	MAT022 — MATEMATICA II	
2	QUI010 — QUIMICA Y SOCIEDAD	
3	FIS130 — FISICA GENERAL III	
3	INF134 — ESTRUCTURAS DE DATOS	
3	INF152 — ESTRUCTURAS DISCRETAS	
3	INF260 — TEORÍA DE SISTEMAS	
3	MAT023 — MATEMATICAS III	
4	FIS120 — FISICA GENERAL II	

Cancelar Guardar cursos

Figura 43: Asignación de asignaturas a los semestres curriculares de una malla.
Fuente: Elaboración propia.