



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

TempusUGR - Desarrollo de una Aplicación de Gestión de Horarios Universitarios Basada en Microservicios Interoperable con Servicios de Calendario Externos

Realizado por
Juan Miguel Acosta Ortega



Para la obtención del título de
Grado en Ingeniería Informática

Dirigido por
Juan Luis Jiménez Laredo

En el departamento de
Dpto. de Ingeniería de Computadores, Automática y Robótica

Convocatoria de junio, curso 2024/25

Aquí la dedicatoria del trabajo

Agradecimientos

Quiero agradecer a X por...

También quiero agradecer a Y por...

Resumen

Incluya aquí un resumen de los aspectos generales de su trabajo, en español.

Palabras clave: Palabra clave 1, palabra clave 2, ..., palabra clave N

Abstract

This section should contain an English version of the Spanish abstract.

Keywords: Keyword 1, keyword 2, ..., keyword N

Yo, Juan Miguel Acosta Ortega, alumno de la titulación Grado en Ingeniería INformática de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada, con DNI 54313742R, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Juan Miguel Acosta Ortega

Granada, 8 de junio de 2025

D. Juan Luis Jiménez Laredo, Profesor del Área de Arquitectura y Tecnología de Computadores del Departamento de Ingeniería de Computadores, Automática y Robótica de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *TempusUGR - Desarrollo de una Aplicación de Gestión de Horarios Universitarios Basada en Microservicios Interoperable con Servicios de Calendario Externos*, ha sido realizado bajo su supervisión por **Juan Miguel Acosta Ortega**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada, 8 de junio de 2025.

El director: Juan Luis Jiménez Laredo

Índice general

1	Introducción	1
1.1.	Estructura de la memoria	1
2	Descripción del problema	3
2.1.	Contexto y problemática	3
2.2.	Estimación del coste agregado	3
2.2.1.	Coste para el profesorado	4
2.2.2.	Coste para el estudiantado	4
2.2.3.	Estimación del coste agregado anual	5
2.3.	Solución propuesta	5
2.4.	Restricciones	6
2.5.	Objetivos del proyecto	6
2.5.1.	Objetivo Principal	6
2.5.2.	Objetivos Generales	7
2.5.3.	Objetivos Específicos	7
3	Estado del arte	8
3.1.	Contextualización	8
3.2.	Visualización y gestión de horarios académicos en la UGR	9
3.3.	Análisis comparativo de sistemas de planificación personalizada en educación superior	12
3.4.	Desarrollo de servicios web	15
3.4.1.	Arquitecturas de software	18
3.4.2.	Tecnologías de desarrollo	22
3.4.3.	Sistemas de almacenamiento de datos	25
3.4.4.	Autenticación y autorización	26
3.4.5.	Comunicación entre servicios	27
3.4.6.	Despliegue de sistemas	29
3.4.7.	Pruebas y calidad del software	30
4	Especificación de requisitos	32
4.1.	Recopilación de información	32
4.2.	Personas	33
4.2.1.	Personas del sistema	34
4.3.	Escenarios	36
4.3.1.	Escenarios del sistema	36
4.4.	Historias de usuario	38
4.4.1.	Estructura de una historia de usuario	38
4.4.2.	Historias de usuario	39
4.5.	Requisitos funcionales	50
4.5.1.	Gestión de usuarios	50
4.5.2.	Gestión de horarios académicos	51

4.6.	Requisitos no funcionales	52
4.6.1.	Rendimiento	52
4.6.2.	Usabilidad	52
4.6.3.	Seguridad	52
4.6.4.	Mantenibilidad	53
4.6.5.	Portabilidad	53
4.6.6.	Disponibilidad	53
4.7.	Requisitos de información	53
4.7.1.	Servicio de usuarios	54
4.7.2.	Servicio de horarios	54
4.7.3.	Servicio de suscripciones académicas	55
4.8.	Validación de los requisitos	56
5	Planificación, metodología y presupuesto del proyecto	57
5.1.	Cronograma del proyecto	57
5.2.	Metodología de desarrollo	58
5.2.1.	Roles y Responsabilidades en este Proyecto	58
5.2.2.	Proceso Scrum Implementado	59
5.2.3.	Justificación de la Metodología	59
5.2.4.	Gestión de Tareas y Seguimiento del Progreso	60
5.3.	Desarrollo y control de versiones	63
5.4.	Gestión de riesgos	64
5.5.	Presupuesto del proyecto	65
5.5.1.	Presupuesto en formato tabla	65
5.5.2.	Desglose de la información	65
6	Diseño del sistema: Arquitectura, tecnologías y decisiones clave	68
6.1.	Decisiones de diseño y arquitectura: Cimientos del sistema	68
6.1.1.	El Backend como pilar central	68
6.1.2.	Arquitectura de Microservicios: Flexibilidad y escalabilidad .	68
6.1.3.	Separación Backend-Frontend: Comunicación vía API REST .	69
6.2.	Backend: Diseño y tecnologías clave	70
6.2.1.	Servicios del Backend identificados	70
6.2.2.	Tecnologías y Frameworks del Backend	71
6.2.3.	Modelado de la base de datos	76
6.2.4.	Diseño de la API	81
6.3.	Frontend: Diseño y tecnologías clave	89
6.3.1.	Diseño de la Interfaz y Experiencia del Usuario (UI/UX) . . .	89
7	Implementación	91
7.1.	Sprint 0	91
7.1.1.	Estructura general de los servicios	91
7.2.	Sprint 1	92
7.2.1.	User Service	92
7.2.2.	Mail Service	96
7.2.3.	Auth Service	97

7.2.4. API Gateway	99
7.2.5. RabbitMQ	101
7.2.6. Primer hito del backend	104
7.3. Sprint 2	105
7.4. Sprint 3	109
7.5. Sprint 4	112
7.5.1. Frontend con Angular	113
7.6. Sprint 5	117
8 Despliegue del sistema	118
8.1. Configuración del servidor	118
8.2. Contenerización del sistema	119
8.2.1. Pasos para la contenerización del backend	119
8.2.2. Docker Compose	121
8.2.3. Pasos para la contenerización del frontend	122
8.3. Levantar el sistema	125
8.3.1. Paso del HTTP a HTTPS en las llamadas al backend	125
8.3.2. Pruebas de carga en un entorno real	126
8.3.3. Solución a la sincronización con Google Calendar	133
9 Conclusiones y trabajos futuros	134
9.1. Evaluación del proyecto	134
9.2. Dificultades y resolución	134
9.3. Mejoras posibles y trabajos futuros	134
Bibliografía	135
Anexo: Glosario	140

Índice de figuras

3.1. Comparación de horarios de diferentes grados: ETSIIT (arriba), ADE (centro) y Doble Grado en Nutrición Humana y Dietética (abajo).	10
3.2. Horario de la asignatura Bases Químicas de la Biología, impartida en el Grado en Biología.	11
3.3. Aplicación móvil de la Universidad de Almería (UAL App).	13
3.4. Aplicación My Study Life.	14
4.1. Persona 1: Alumno de la UGR	34
4.2. Persona 2: Profesor de la UGR	35
4.3. Persona 3: "Administrador" de la UGR	35
5.1. Gantt del proyecto.	57
5.2. Ejemplo de historia de usuario en Github Projects.	61
5.3. Tablero del 2º Sprint durante su desarrollo.	61
5.4. Segmento del "Roadmap" del proyecto.	62
5.5. Resumen de horas de desarrollo en Clockify.	63
6.1. Comunicación Backend-Frontend a través de API REST	70
6.2. Comparativa de velocidades entre lenguajes de programación	72
6.3. Arquitectura general del Backend	73
6.4. Modelo de datos del Servicio de Usuarios	76
6.5. Modelo de datos del Servicio de Horarios y Calendario	78
6.6. Modelo de datos del Servicio de Matriculaciones	80
6.7. Logo de TempusUGR	89
7.1. Diagrama de clases del servicio User Service realizado con PlantUML	93
7.2. Comunicación general entre los componentes del backend	95
7.3. Correo de activación de cuenta	97
7.4. Flujo de autenticación y generación de tokens JWT	99
7.5. Configuración de RabbitMQ en el servicio mail-service	103
7.6. Proceso de scrapping de asignaturas y horarios	105
7.7. Servicio de suscripciones académicas	107
7.8. Servidor de descubrimiento de servicios con Eureka	109
7.9. Pruebas unitarias e integración del servicio schedule-consumer-service	112
7.10. Página de inicio de sesión	114
7.11. Página principal del calendario	115
7.12. Página de sincronización con Google Calendar	115
7.13. Página de suscripciones a asignaturas	116
7.14. Página de eventos de grupo y facultad	116
8.1. Gráfica de la prueba de carga con 100 usuarios concurrentes.	127
8.2. Gráfica de la prueba de carga con 500 usuarios concurrentes.	129
8.3. Gráfica de la prueba de carga con 1000 usuarios concurrentes.	130

Índice de tablas

4.1. Estructura de una historia de usuario	39
4.2. Historia de usuario HU-1	39
4.3. Historia de usuario HU-2	40
4.4. Historia de usuario HU-3	40
4.5. Historia de usuario HU-4	41
4.6. Historia de usuario HU-5	41
4.7. Historia de usuario HU-6	42
4.8. Historia de usuario HU-7	42
4.9. Historia de usuario HU-8	42
4.10. Historia de usuario HU-9	43
4.11. Historia de usuario HU-10	43
4.12. Historia de usuario HU-11	44
4.13. Historia de usuario HU-12	44
4.14. Historia de usuario HU-13	45
4.15. Historia de usuario HU-14	46
4.16. Historia de usuario HU-15	47
4.17. Historia de usuario HU-16	48
4.18. Historia de usuario HU-17	48
4.19. Historia de usuario HU-18	49
4.20. Historia de usuario HU-19	49
4.21. Historia de usuario HU-20	50
5.1. Resumen de costes del proyecto.	65
6.1. Comparativa: Arquitectura Monolítica vs. Microservicios	69
8.1. Resultados de la prueba de carga con 100 usuarios concurrentes.	127
8.2. Resultados de la prueba de carga con 500 usuarios concurrentes.	128
8.3. Resultados de la prueba de carga con 1000 usuarios concurrentes.	130

1. Introducción

En la Universidad de Granada (UGR) no existe en la actualidad un sistema centralizado y personalizado de calendario académico para estudiantes y profesorado. Para confeccionar sus propios itinerarios, los usuarios deben acceder a diversas páginas abiertas en la web institucional, combinando manualmente la información de cada uno de los portales disponibles. Esta situación genera redundancias, posibles errores y un coste agregado en tiempo y esfuerzo.

Con el fin de dar respuesta a esta carencia, este proyecto propone la creación de un servicio de calendario automático y personalizado para la comunidad universitaria, adoptando una arquitectura de microservicios como solución de base. Mediante la descomposición del sistema en servicios independientes —cada uno responsable de una funcionalidad concreta como gestión de usuarios, suscripciones, calendario o autenticación— se facilita la comunicación síncrona y asíncrona entre componentes, la integración de un API Gateway para el enrutamiento y la seguridad, y la aplicación de configuraciones y buenas prácticas que aseguren la disponibilidad y el rendimiento del sistema. Este enfoque modular y distribuido aporta flexibilidad, mantenibilidad y despliegues ágiles, a la vez que confiere resiliencia al aislar posibles fallos en servicios específicos.

En resumen, este Trabajo de Fin de Grado se concibe como una oportunidad para implementar un sistema de calendario académico personalizado basado en microservicios, enfrentándose a retos técnicos reales y contribuyendo directamente a la comunidad universitaria. A lo largo del desarrollo se abordarán aspectos clave como el diseño de APIs claras y bien definidas, la persistencia de datos en múltiples servicios, la consistencia y tolerancia a fallos, y las comunicaciones internas del sistema. Además de consolidar conocimientos teóricos, se evaluará el impacto en los costes de tiempo y gestión para los casi 60 000 alumnos y 4 000 profesores de la UGR, demostrando cómo un servicio bien diseñado puede optimizar procesos administrativos esenciales.

1.1. Estructura de la memoria

La estructura de la memoria se divide en los siguientes capítulos:

- **Capítulo 1: Introducción.** En este capítulo se presenta el contexto general del trabajo, la motivación que ha llevado a su realización, y la estructura que seguirá la memoria.
- **Capítulo 2: Descripción del problema.** Este capítulo describe con detalle el problema principal que se pretende resolver, los retos existentes en la gestión de horarios académicos y las limitaciones de los enfoques tradicionales. También se identifican las necesidades de los usuarios y se justifica la necesidad de una solución basada en tecnologías web.

- **Capítulo 3: Estado del arte.** Se realiza una revisión exhaustiva de las soluciones existentes en el ámbito de los sistemas de gestión de horarios, tanto comerciales como académicos, así como en el desarrollo de sistemas de información web. Se analizan sus ventajas, limitaciones, y se identifican oportunidades de mejora que el presente trabajo busca aprovechar.
- **Capítulo 4: Especificación de requisitos.** Aquí se definen los distintos actores del sistema mediante personas y escenarios de uso. Se detallan los requisitos del sistema agrupados en requisitos funcionales, no funcionales y de información. Además, se presentan historias de usuario que ayudan a comprender cómo interactúan los usuarios con el sistema en distintos contextos.
- **Capítulo 5: Planificación, metodología y presupuesto del proyecto.** En este capítulo se describe la planificación temporal del proyecto, especificando las fases y entregas principales. Se justifica la elección de la metodología ágil (por ejemplo, Scrum) y se explica cómo se ha aplicado durante el desarrollo. También se presenta una estimación de costes y recursos necesarios, incluyendo un presupuesto detallado.
- **Capítulo 6: Diseño del sistema. Arquitectura, tecnologías y decisiones clave** Se expone el diseño de la arquitectura del sistema basada en microservicios, detallando la organización de los diferentes componentes, el diseño de la API REST, la estructura de las bases de datos utilizadas y el diseño de la interfaz de usuario. Se discuten las decisiones técnicas tomadas y se ilustran mediante diagramas e imágenes.
- **Capítulo 7: Implementación.** En este capítulo se documenta el desarrollo del sistema, dividido en los distintos sprints planificados. Se detallan los avances realizados, las herramientas utilizadas y las tecnologías integradas. También se explican los problemas encontrados durante la implementación y las soluciones adoptadas.
- **Capítulo 8: Despliegue del sistema.** Aquí se describe el proceso de despliegue del sistema en el servidor proporcionado por la Universidad de Granada (UGR). Se detallan aspectos como la configuración del servidor, la gestión de certificados SSL, la habilitación del protocolo HTTPS, el uso de dominios personalizados, y la realización de pruebas de carga en un entorno realista.
- **Capítulo 9: Conclusiones y trabajos futuros.** Se presentan las conclusiones obtenidas a partir del desarrollo del proyecto, destacando los logros alcanzados, las lecciones aprendidas y las dificultades superadas. Asimismo, se proponen líneas de trabajo futuro y posibles mejoras que podrían incorporarse al sistema en versiones posteriores.
- **Anexo: Glosario.** Este anexo recoge una recopilación de términos técnicos, abreviaturas y conceptos relevantes empleados a lo largo del documento, con el objetivo de facilitar la comprensión del lector no especializado.

2. Descripción del problema

La Universidad de Granada (UGR), con su amplia oferta formativa y su elevado número de usuarios —alrededor de 60 000 estudiantes y 3.500 profesores [1]— depende de una gestión de horarios académicos ágil y precisa para garantizar la correcta organización de sus actividades docentes. Sin embargo, en la práctica este proceso adolece de dispersión y falta de personalización: cada usuario debe rastrear múltiples portales en la web institucional para construir su propio itinerario, lo que no solo incrementa la probabilidad de errores y solapamientos, sino que también genera un coste oculto en tiempo y recursos cuya magnitud analizaremos en detalle en secciones posteriores.

2.1. Contexto y problemática

La Universidad de Granada (UGR) es una de las universidades más grandes de España, con una amplia oferta académica y un gran número de estudiantes y profesores. La gestión de horarios académicos es un aspecto crítico para el funcionamiento eficiente de la universidad. Sin embargo, la UGR podría mejorar en este campo en varios aspectos:

- **Falta de personalización:** Actualmente, los estudiantes y profesores tienen acceso a un horario general que no se adapta a sus necesidades específicas. Esto dificulta la planificación y organización de su tiempo.
- **Dificultad en la gestión de cambios:** Los cambios en los horarios y eventos académicos (tutorías, clases de recuperación, charlas, etc.) no se comunican de manera efectiva a los estudiantes y profesores, lo que puede llevar a confusiones y malentendidos.
- **Integración con servicios externos:** La falta de integración con servicios de calendario externos como Google Calendar limita la accesibilidad y la organización del horario académico.

2.2. Estimación del coste agregado

Más allá de los costes económicos directos asociados a software o personal administrativo, existe un coste “humano” o “invisible” que se manifiesta en el tiempo y el esfuerzo invertidos por los colectivos implicados en la realización de su calendario escolar personalizado. Esta sección busca cuantificar, de manera aproximada, dicho coste anual en una universidad con una población similar a la descrita anteriormente (aproximadamente 60.000 estudiantes y 3.500 profesores).

Para esta estimación, consideraremos el tiempo medio que, de forma no remunerada o fuera de sus funciones explícitas, dedican estudiantes y profesores a la consulta, ajuste y resolución de problemas relacionados con la confección de horarios cada cuatrimestre. La información se ha recopilado a través de encuestas realizadas a miembros del departamento de Ingeniería de Computadores, Automática y Robótica, y a estudiantes de distintos cursos de la ETSIIT.

2.2.1. Coste para el profesorado

El profesorado dedica un tiempo considerable a la revisión de horarios, la coordinación con otros docentes, la solicitud de cambios por solapamientos o la adaptación a nuevas asignaciones. Aunque gran parte de esta labor se gestiona a través de los coordinadores de titulación y jefes de departamento, la interacción individual sigue siendo significativa.

Para obtener una estimación del tiempo invertido por el profesorado, se realizó una consulta a los profesores del departamento de Ingeniería de Computadores, Automática y Robótica (ICAR). Las respuestas obtenidas revelan una dedicación variada en la configuración y ajuste de horarios. Excluyendo las respuestas que se referían a actividades diarias o eran valores atípicos, la mayoría de las estimaciones para la configuración inicial y ajustes esporádicos se sitúan entre los 15 y 45 minutos por cuatrimestre. Para esta estimación, se ha calculado una media ponderada de las respuestas, resultando en un promedio de **30 minutos (0.50 horas) por profesor y cuatrimestre** para las tareas de configuración y ajuste de horarios.

- **Población:** Aproximadamente 3.500 profesores.
- **Tiempo estimado por profesor y cuatrimestre:** 0.50 horas.
- **Coste anual por profesorado:** $(3.500 \text{ profesores}) \times 0.50 \text{ horas/cuatrimestre} \times 2 \text{ cuatrimestres/año} = 3.500 \text{ horas/año}$.

2.2.2. Coste para el estudiantado

El estudiantado es el colectivo numéricamente mayor y, por ende, el que acumula un mayor volumen de tiempo en la gestión de sus horarios. Los problemas comunes incluyen solapamientos entre asignaturas obligatorias u optativas, dificultades para encajar horarios por trabajo o estudios adicionales, y la necesidad de consultar repetidamente las plataformas hasta que los horarios se consolidan. Además los estudiantes de un Grado como puede ser el de Ingeniería Informática tienen un número muy inferior de grupos por asignatura comparado por ejemplo con los estudiantes de Medicina, lo que les obliga a realizar una mayor cantidad de consultas para encontrar un horario que se ajuste a sus necesidades.

Se ha realizado una encuesta anónima similar a alumnos de diferentes cursos de la ETSIIT. Las estimaciones obtenidas del estudiantado son consistentemente similares a las reportadas por el profesorado del departamento ICAR, reflejando

una dedicación significativa a la consulta y gestión de sus propios horarios. En base a estas encuestas, se estima una media de **45 minutos por estudiante y cuatrimestre**. Este tiempo engloba la consulta inicial y repetida de horarios, el análisis de solapamientos, la búsqueda de soluciones y la comunicación para reportar problemas.

- **Población:** Aproximadamente 60.000 estudiantes.
- **Tiempo estimado por estudiante y cuatrimestre:** 45 min.
- **Coste anual por estudiantado:** $(60.000 \text{ estudiantes}) \times 0.75 \text{ hora/cuatrimestre} \times 2 \text{ cuatrimestres/año} = 90.000 \text{ horas/año}$.

2.2.3. Estimación del coste agregado anual

Sumando el tiempo estimado para ambos colectivos, obtenemos una cuantificación del coste “humano” anual:

- **Total de horas anuales:** 90.000 horas (estudiantes) + 3.500 horas (profesores) = **93.500 horas/año**.

Para contextualizar esta cifra, podemos convertirla en un equivalente de jornadas laborales o incluso en un valor económico aproximado, aunque el objetivo principal es visibilizar el volumen de tiempo.

- **Equivalente en jornadas laborales (8 horas/día):** $93.500 \text{ horas/año} \div 8 \text{ horas/día} = 9.358 \text{ jornadas laborales/año}$.

Esta cifra de 93.500 horas anuales dedicadas por la comunidad universitaria a la gestión y resolución de problemas de horarios subraya la magnitud del **coste “humanO”** que puede pasarse por alto. Representa una cantidad significativa de tiempo y esfuerzo que podría ser redirigida hacia actividades más productivas académicamente o de mejora de la calidad de vida de la comunidad universitaria. La optimización de los procesos de generación de horarios, la mejora de la comunicación y la implementación de herramientas más eficientes tienen el potencial de mitigar sustancialmente este impacto.

2.3. Solución propuesta

Partiendo de la problemática observada, se propone la elaboración de un sistema de gestión personalizada de horarios académicos para los grados de la Universidad de Granada. Esto permitirá a los estudiantes y profesores acceder a su información horaria de manera centralizada y con comunicaciones efectivas sobre cambios y eventos académicos. Además, la integración con servicios de calendario externos como Google Calendar mejorará la accesibilidad y la organización del horario académico.

2.4. Restricciones

A fin de implementar la solución propuesta de manera eficaz y garantizar que responda a las necesidades detectadas en el contexto de la UGR, es imprescindible que el sistema cumpla una serie de restricciones técnicas y funcionales. Estas restricciones aseguran que la solución sea completa, segura, accesible, compatible, escalable e integrable con otros servicios, permitiendo así su adopción y correcto funcionamiento en el entorno universitario.

- **Completitud:** El sistema debe ser capaz de gestionar todos los grados y asignaturas de la UGR.
- **Seguridad:** El sistema debe manejar la mínima información privada posible para un funcionamiento normal.
- **Accesibilidad:** El sistema debe ser accesible desde cualquier dispositivo con conexión a Internet.
- **Compatibilidad:** El sistema debe ser compatible con los navegadores web más utilizados.
- **Escalabilidad:** El sistema debe ser capaz de manejar un gran número de usuarios y solicitudes simultáneas.
- **Integración:** El sistema debe ser capaz de integrarse con servicios de calendario externos como Google Calendar.

2.5. Objetivos del proyecto

En esta sección se detallan los objetivos que guiarán el desarrollo del sistema. Se dividen en un objetivo principal, que establece la meta general del proyecto, y objetivos generales y específicos, que desglosan las funcionalidades clave y las capacidades esperadas del sistema. Estos objetivos servirán como la hoja de ruta para el diseño, implementación y evaluación del proyecto, asegurando que el producto final cumpla con las necesidades de los usuarios y las expectativas de la UGR en cuanto a la organización académica.

2.5.1. Objetivo Principal

Desarrollar una aplicación backend basada en microservicios robusta, escalable y segura para la gestión personalizada de horarios académicos de la Universidad de Granada (UGR), que permita a los usuarios acceder a su información horaria de manera centralizada y personalizada, facilitando la integración con servicios de calendario externos como Google Calendar para mejorar la accesibilidad y la organización.

2.5.2. Objetivos Generales

1. Personalizar la visualización del horario para cada tipo de usuario según sus suscripciones.
2. Facilitar la gestión y comunicación de cambios de horario y eventos académicos (tutorías, clases de recuperación, charlas, etc.).
3. Permitir la integración con servicios de calendario externos para una mayor accesibilidad y sincronización de la información horaria.

2.5.3. Objetivos Específicos

1. Implementar un sistema de registro y autenticación seguro para usuarios (alumnos y profesores) utilizando correos electrónicos institucionales de la UGR.
2. Permitir a los alumnos y profesores suscribirse y revocar suscripciones a los grupos de asignaturas de los grados que cursan / imparten.
3. Generar y mostrar el horario personalizado de cada usuario en función de sus suscripciones, incluyendo información detallada de la asignatura, grupo, horario, profesores y aula.
4. Permitir a los profesores y administradores crear, modificar y eliminar eventos extra a las clases oficiales (tutorías, clases de recuperación, charlas, etc.) y notificar a los alumnos sobre estos eventos.
5. Permitir a los usuarios exportar su horario en formato iCalendar (.ics) para su importación en diversos sistemas de calendario.
6. Implementar la sincronización automática del horario de los usuarios con Google Calendar, reflejando los cambios en tiempo real.

3. Estado del arte

En esta sección se presenta una revisión del estado del arte en el ámbito de la planificación y gestión de horarios académicos, así como las tecnologías y paradigmas arquitectónicos utilizados en el desarrollo de sistemas de información basados en web. Se analizarán las limitaciones de los sistemas actuales, se compararán con soluciones más avanzadas implementadas en otras instituciones, y se explorarán las tecnologías y stacks utilizados en el desarrollo del sistema propuesto.

3.1. Contextualización

La planificación temporal y académica son pilares indispensables para un buen desempeño en el entorno universitario. Para los alumnos de centros con una estructura académica compleja, o profesores con varias horas de docencia en diferentes grupos, como la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación (ETSIIT) de la Universidad de Granada (UGR), o incluso varios grados, la capacidad de organizar y visualizar sus horarios de manera clara y personalizada se convierte en una necesidad notable.

La gestión de múltiples asignaturas, grupos de teoría y prácticas, seminarios, tutorías y actividades personales requiere de herramientas que vayan más allá de la simple presentación estática de información, y además de manera general para toda la institución.

Sin embargo, los sistemas tradicionales de visualización de horarios en muchas instituciones académicas presentan limitaciones significativas. De manera frecuente, la información se ofrece en formatos estáticos, como documentos PDF o imágenes, que dificultan la personalización, la interacción, la integración con las herramientas digitales que los estudiantes utilizan en su día a día, y en algunos casos una visibilidad accesible.

Esta falta de dinamismo y personalización puede generar confusión, dificultar la planificación y no aprovechar las ventajas que ofrecen las tecnologías actuales para una gestión académica más eficiente y adaptada a las necesidades individuales.

Este capítulo presenta una revisión del estado del arte que fundamenta la necesidad y el enfoque del proyecto. Se analiza la situación actual de la gestión y visualización de horarios en la UGR. Posteriormente, se realizará un análisis comparativo con sistemas más avanzados implementados en otras instituciones de educación superior. A continuación, se profundizará en los paradigmas arquitectónicos de backend y en tecnologías en el desarrollo de sistemas de información basados en web. Finalmente, se examinarán los diferentes stacks

tecnológicos, incluyendo Java y el ecosistema Spring para el desarrollo de microservicios, RabbitMQ para la comunicación asíncrona, la combinación de bases de datos MySQL y MongoDB bajo el principio de persistencia políglota, la librería Jsoup para la adquisición de datos mediante web scraping, y las tecnologías empleadas para el despliegue del sistema, como Docker.

3.2. Visualización y gestión de horarios académicos en la UGR

La Universidad de Granada, al igual que muchas otras universidades, descentraliza sus sedes, de modo que cada una de ellas tiene su propio sistema de gestión de la información. En este sentido, las facultades cuentan con una serie de sistemas de información propios que se encargan de la generación de horarios académicos, asignación de aulas y profesores a los grupos tanto de teoría como de prácticas de las distintas titulaciones y asignaturas. Esta información a su vez se le facilita a la Universidad de Granada para la centralización de la información.

Para acceder a la información de los horarios, los estudiantes y docentes pueden hacerlo de diferentes maneras:

- A través de la página propia de su facultad. Poniéndole de ejemplo a la ETSIIT, debemos acceder a la página oficial de la facultad [2] y buscar la información en la sección de “Calendario de exámenes” en caso de querer saber los días y rangos horarios de estos y visualizándolo con un pdf, o a “Calendario académico y horarios” y a “Grado en Ingeniería Informática” en caso de querer saber los horarios de los diferentes grupos del grado, presentado todo ello en un pdf contenedor de alrededor de 40 tablas.

De esta manera tendremos que buscar el año al que pertenece la asignatura de la que estamos matriculados y el grupo al que pertenecemos. De esta manera obtenemos su franja horaria y aula, pero no profesor que imparte la asignatura.

Sin embargo, el formato de las tablas cambia de un grado a otro 3.1, haciendo que el estudiante tenga que buscar la información de manera diferente en cada grado si está matriculado en más de uno, y obteniendo información diferente. En el caso del grado de Administración y Dirección de Empresas por ejemplo, no se muestra el aula en la que se imparte la clase, pero sí las asignaturas bilingües, y los profesores que las imparten.

Esta forma de visualización de horarios es inconsistente entre grados, y no es accesible para personas con discapacidad visual.

1º A Grado en Ingeniería Informática 1er. cuatrimestre											
	Lunes	Martes	Miércoles	Jueves	Viernes						
8:30-9:30											
9:30-10:30	ALEM 0,3	ALEM 0,3	FP 0,3	FPT 0,3	FS 0,3						
10:30-11:30	CA 0,3	FP 0,3	FS 0,3	CA 0,3	FP 0,3						
11:30-12:30	CA (A1) ALEM (A1)	FPT (A2) FP (A2)	FS (A3)	FP (A1) ALEM (A3)	FP (A2) CA (A3)	FS (A1) FPT (A2)	CA 0,3	FS (A1) ALEM (A2)	FP (A3)		
12:30-13:30	FP (A1) ALEM (A3)	FS (A2) FP (A2)	CA (A3)	FP (A2) CA (A3)	FS (A2) FP (A2)	CA 0,3	CA 0,3	FS (A2) CA (A2)	FP (A3)		
13:30-14:30											
15:30-16:30											
16:30-17:30											
17:30-18:30											
18:30-19:30											
19:30-20:30											
20:30-21:30											

1º A GADE. PRIMER SEMESTRE (D03)					
	LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES
8:30 a 9:30	MAT MAT E20	MAT MAT E20		FDAE	
9:30 a 10:30	MAT MAT E20	MAT MAT E20		FDAE	
10:30 a 11:30	IOF	IOF	IMK IMK (D03) (D25)	MAT MAT (D03)	
11:30 a 12:30	IOF	IOF	IMK IMK (D03) (D25)	MAT MAT (D03)	
12:30 a 13:30	EP EP (D03) (E20)	EP EP (D03) (E20)		FDAE	IMK IMK (D03) (D25)
13:30 a 14:30	EP EP (D03) (E20)	EP EP (D03) (E20)		FDAE	IMK IMK (D03) (D25)

Grado en Nutrición Humana y Dietética (Z.01)												Última actualización: 03/12/2014								
Asignatura	Genero	Tipo	Creditos	Horario	16	23	30	7	14	21	28	4	11	18	25	2	9	16	23	30
Bioquímica Metabólica (SQM)	1	FB	1,5	8:30 h 11:30 h 16:00 h				1	6											
Fisiología Humana (FH)	1	FB	1,5	8:30 h 11:30 h 16:00 h						1,2	1,2									
Microbiología (M)	1	Ob	1,5	8:30 h 11:30 h 16:00 h						1	3									
Nutrición I (N1)	1	FB	1,5	8:30 h 11:30 h 16:00 h				2	3	2	4									
Tecnología Culinaria (TC)	1	Ob	1,5	8:30 h 11:30 h 16:00 h		1	2	3	4	5										
Ampliación de Bromatología (ABRO)	2	Ob	1,5	8:30 h 11:30 h 16:00 h								1	3	2	4					
Fisiopatología (FP)	2	Ob	1,5	8:30 h 11:30 h 16:00 h										1	3	2	4	5		
Nutrición II (N2)	2	Ob	1,5	8:30 h 11:30 h 16:00 h									1	3	2	4	5			
Parasitología Alimentaria (PA)	2	Ob	1,5	8:30 h 11:30 h 16:00 h								1	3	2	4					
Toxicología Alimentaria (TOA)	2	Ob	1,5	8:30 h 11:30 h 16:00 h									1		2,3	4				

Figura 3.1: Comparación de horarios de diferentes grados: ETSIIT (arriba), ADE (centro) y Doble Grado en Nutrición Humana y Dietética (abajo).

- A través de la web grados UGR [3] se puede buscar la información de los horarios de las asignaturas de los diferentes grados de la Universidad de Granada. Para ello debemos seleccionar rama de conocimiento, grado, curso y asignatura. De esta manera obtenemos un horario semanal con las franjas horarias, aulas, profesores y fechas tanto de inicio como de fin. Este método nos proporciona una interfaz estándar y más información, pero también es más lento y tedioso para consultar por varias asignaturas o incluso grados.
- A través de las webs de cada departamento. Por ejemplo en la web del departamento de Ciencias de la Computación e Inteligencia Artificial [4] se

puede consultar la información de las asignaturas o profesores de este. Ofrece información adicional como asignaturas que imparte "x" profesor y su horario de tutorías y docencia.

Además para acceder a la información de períodos de actividad docente, exámenes finales, períodos de evaluación de convocatorias ... se ha de acceder a la web de la Secretaría General en la UGR [5] para consultar otro pdf.

En general la información de los horarios académicos de la Universidad de Granada es poco accesible, eficiente y consistente entre grados y facultades, lo que hace que el estudiante tenga que buscar la información de manera manual y tediosa. Además no hay manera de consultar de manera sencilla un calendario personal que incluya tanto los horarios de las asignaturas como los exámenes y períodos de evaluación, entre otros.

Pongamos el ejemplo de un estudiante matriculado en el primer curso del Grado de Biología en la Universidad de Granada con el estándar de cinco asignaturas en su primer cuatrimestre 3.2. Este estudiante tiene que buscar la información de los horarios de las asignaturas en la web de su facultad, en la web de la Universidad de Granada o en la web del departamento al que pertenezca cada asignatura. Suponemos que decide buscar su horario en la web de grados ugr, y una vez seleccionada la rama de conocimiento, grado, curso y asignatura, obtiene un horario semanal con las franjas horarias de todos los grupos de la asignatura, aulas, profesores y fechas tanto de inicio como de fin. Está matriculado por ende en la asignatura "Bases Químicas de la Biología" en el grupo "A" de teoría y en el grupo "2" de prácticas, por lo que tiene que buscar los sectores que pertenecen a su grupos para poder obtener su horario personalizado para esa materia.

La realidad con la que se encuentra el estudiante es con la siguiente:

Horario		LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES	
Hora	Día						
7:00							
8:00							
9:00		Grupo: 1 Grupo: 2 Grupo: 3 Grupo: 4 Grupo: 5 Grupo: 6 Grupo: 7	Grupo: 1 Grupo: 2 Grupo: 3 Grupo: 4 Grupo: 5 Grupo: 6 Grupo: 7	Grupo: 1 Grupo: 2 Grupo: 3 Grupo: 4 Grupo: 5 Grupo: 6 Grupo: 7	Grupo: 1 Grupo: 2 Grupo: 3 Grupo: 4 Grupo: 5 Grupo: 6 Grupo: 7	Grupo: 1 Grupo: 2 Grupo: 3 Grupo: 4 Grupo: 5 Grupo: 6 Grupo: 7	Grupo: 1 Grupo: 2 Grupo: 3 Grupo: 4 Grupo: 5 Grupo: 6 Grupo: 7
10:00		Grup: 7 7	Grup: 5 5	Grup: 7 7	Grup: 5 5	Grup: 7 7	Grup: 7 7
11:00		Grup: 6 6	Grup: 8 8	Grup: 5 5	Grup: 6 6	Grup: 7 7	Grup: 7 7
12:00		Gri: 8 8	Gri: 6 6	Gri: 8 8	Gri: 6 6	Gri: 8 8	Gri: 6 6
13:00				Grup: C C		Grup: C C	
14:00							
15:00					Grup: D D		
16:00		Grup: 7 7	Grup: 5 5	Grup: 3 3	Grup: 1 1	Grup: 3 3	Grup: 5 5
17:00				Grup: 5 5	Grup: 7 7	Grup: 1 1	Grup: 3 3
18:00		Grup: 4 4	Grup: 2 2	Grup: 6 6	Grup: 8 8	Grup: 2 2	Grup: 8 8
19:00		Grup: 6 6	Grup: 6 6	Grup: 7 7	Grup: 7 7	Grup: 5 5	Grup: 5 5
20:00							

Figura 3.2: Horario de la asignatura Bases Químicas de la Biología, impartida en el Grado en Biología.

El estudiante tiene que dedicar un tiempo considerable en buscar las franjas pertenecientes a sus grupos, puesto que no hay una sencilla visualización de los mismos. Además se requiere una búsqueda activa con el cursor para poder ver las franjas ocultas, y esta acción puede resultar tediosa cuando hay muchos sectores juntos, como en este caso.

Podemos concluir tras analizar la situación actual de aprovisionamiento de horarios académicos a los usuarios de la Universidad de Granada, que surge la necesidad de un sistema que permita la visualización de horarios académicos de manera sencilla, accesible y personalizada.

3.3. Análisis comparativo de sistemas de planificación personalizada en educación superior

Frente al modelo estático observado de manera generalizada en la Universidad de Granada, el panorama de la gestión de horarios en otras instituciones de educación superior y en el mercado de software educativo muestra una clara tendencia hacia sistemas más dinámicos, personalizados e integrados.

Existen diversas soluciones, desde módulos dentro de grandes sistemas ERP educativos hasta herramientas especializadas en la creación y gestión de horarios y planificadores académicos, pasando por aplicaciones de seguimiento del tiempo adaptadas al ámbito educativo. El análisis de estas herramientas revela un conjunto de características comunes y avanzadas que definen el estado del arte en este dominio:

- Por un lado ciertas universidades han desarrollado sistemas internos que permiten a los estudiantes acceder a sus horarios de manera personalizada, integrando información sobre asignaturas, grupos, aulas y profesores. Estos sistemas suelen ofrecer una interfaz gráfica intuitiva y accesible, permitiendo a los usuarios visualizar su horario de manera clara y sencilla.

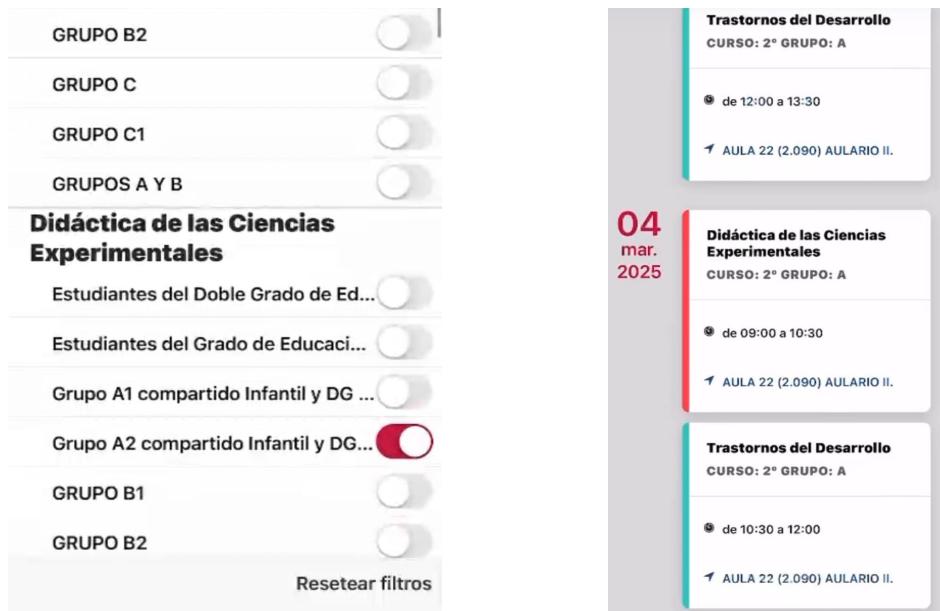


Figura 3.3: Aplicación móvil de la Universidad de Almería (UAL App).

Exponiendo un ejemplo, la Universidad de Almería (UAL) ha implementado en su aplicación móvil multiplataforma “UAL App” 3.3, la posibilidad de, seleccionando las asignaturas y grupos en los que se está matriculado, obtener una lista de las actividades ordenadas por hora según el día de la semana.

De esta manera en la misma aplicación que los estudiantes usan para consultar sus notas, expediente académico, días festivos, etc. pueden consultar su horario académico de manera rápida en el mismo ecosistema.

- Por otro lado, y de manera externa a las universidades, existen aplicaciones de gestión de horarios y planificación personal que permiten a los estudiantes integrar sus horarios académicos con otras actividades personales, como trabajos, eventos sociales o compromisos familiares.

Estas aplicaciones suelen ofrecer funciones avanzadas de recordatorios, notificaciones y sincronización con calendarios digitales, lo que facilita la organización del tiempo y la gestión de tareas. Un ejemplo representativo de este tipo de sistemas es 'My Study Life' [6], una aplicación multiplataforma que permite a los estudiantes gestionar sus horarios académicos, tareas y exámenes de manera integrada 3.4. En este caso el sistema en sí no cuenta con los datos internos de la universidad, sino que el estudiante tiene que introducir manualmente los datos de sus asignaturas y grupos, sin embargo, ofrece una interfaz intuitiva y fácil de usar, permitiendo a los estudiantes visualizar su horario de manera clara y sencilla. Además de la posibilidad de añadir tareas y exámenes, la aplicación permite establecer recordatorios y notificaciones para ayudar a los estudiantes a mantenerse organizados y cumplir con sus plazos, y posee widgets personalizados para la pantalla de inicio de los dispositivos móviles e incluso aplicaciones para smartwatch, lo que consigue una integración total con el ecosistema del usuario.

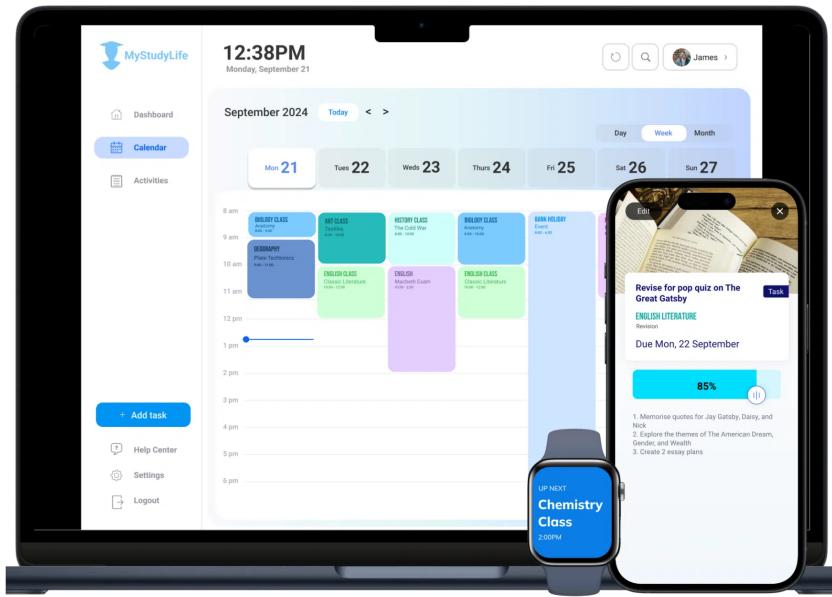


Figura 3.4: Aplicación My Study Life.

De manera general, y de uso más extendido, existen aplicaciones de gestión de tiempo y productividad que permiten a los usuarios organizar su tiempo de manera más eficiente como lo son Google Calendar [7] o Microsoft Outlook [8]. Estas aplicaciones permiten a los usuarios crear eventos, establecer recordatorios y sincronizar sus calendarios con otros dispositivos y aplicaciones. Sin embargo, no están específicamente diseñadas para la gestión de horarios académicos y pueden carecer de algunas funciones avanzadas que ofrecen otras aplicaciones más especializadas. Sin embargo también son usados para, sincronizando calendarios de sistemas externos, centralizar la información de los horarios académicos y otras actividades personales en un solo lugar, lo que facilita la gestión del tiempo y la planificación de tareas.

- Por último, existen sistemas de gestión de horarios y planificación académica que se integran con plataformas de aprendizaje en línea y sistemas de gestión del aprendizaje LMS, como Moodle [9] o Blackboard. Estos sistemas permiten a los estudiantes acceder a su horario académico y a la información relacionada con sus cursos de manera centralizada, facilitando la gestión de tareas, exámenes y actividades académicas. Un ejemplo de este tipo de sistemas es el módulo de planificación académica de Moodle que permite a los estudiantes visualizar su horario académico y gestionar sus tareas y exámenes de manera integrada con la plataforma de aprendizaje.

Este módulo ofrece una interfaz gráfica intuitiva y accesible, permitiendo a los estudiantes personalizar su horario académico y acceder a la información relacionada con sus cursos de manera centralizada. Además, el módulo de planificación académica de Moodle permite a los estudiantes establecer recordatorios y notificaciones para ayudarles a mantenerse organizados y

cumplir con sus plazos.

Sin embargo, este tipo de sistemas suelen estar limitados a las plataformas de aprendizaje en línea y no ofrecen la misma flexibilidad y personalización que otras aplicaciones de gestión de horarios y planificación personal.

3.4. Desarrollo de servicios web

El desarrollo de servicios web ha evolucionado significativamente en los últimos años, impulsado por la creciente demanda de aplicaciones distribuidas y la necesidad de integrar sistemas heterogéneos. En este contexto, se han desarrollado diferentes paradigmas arquitectónicos y tecnologías que permiten la creación de servicios web eficientes y escalables.

Esta evolución ha llevado a una clara distinción de responsabilidades en el desarrollo de aplicaciones web, consolidando los conceptos de **Frontend** y **Backend** como pilares fundamentales.

El **frontend**, también conocido como el "lado del cliente", es la parte de la aplicación con la que el usuario interactúa directamente. Abarca la interfaz de usuario (**UI**), la experiencia de usuario (**UX**) y toda la lógica que se ejecuta en el navegador web del cliente. Las tecnologías predominantes en el desarrollo frontend incluyen HTML para la estructura, CSS para la presentación y JavaScript para la interactividad.

En los últimos años, frameworks y bibliotecas de JavaScript como React, Angular y Vue.js han ganado una enorme popularidad, permitiendo la creación de interfaces de usuario dinámicas, complejas y reutilizables. Estos frameworks facilitan la gestión del estado de la aplicación en el cliente y la comunicación asíncrona con el servidor, mejorando la fluidez y la reactividad de las aplicaciones web modernas.

Por otro lado, el **backend**, o "lado del servidor", es el motor que impulsa la aplicación. Se encarga de la lógica de negocio, el procesamiento de datos, la autenticación de usuarios, la gestión de bases de datos y la comunicación con otros sistemas o servicios. Es invisible para el usuario final, pero crucial para el funcionamiento de la aplicación. Existe una amplia variedad de lenguajes y frameworks para el desarrollo backend, como Node.js (con Express.js o NestJS), Python (con Django o Flask), Java (con Spring), Ruby (con Ruby on Rails), PHP (con Laravel o Symfony) y C# (con .NET). La elección de la tecnología backend suele depender de factores como los requisitos de rendimiento, la escalabilidad, la experiencia del equipo de desarrollo y el ecosistema existente. El backend también es responsable de la seguridad de la aplicación, implementando medidas para proteger los datos y prevenir accesos no autorizados.

La comunicación entre el frontend y el backend se ha estandarizado en gran medida a través de las **Interfaces de Programación de Aplicaciones (API)**.

Las APIs son vitales para la creación de experiencias digitales modernas, ya

que simplifican como los sistemas se comunican, ofreciendo flexibilidad e independencia a una empresa. El mundo de las APIs está en constante evolución, y cada vez más empresas están adoptando este enfoque para integrar sus sistemas y servicios. Las APIs permiten a los desarrolladores acceder a funcionalidades específicas de una aplicación o servicio sin necesidad de conocer su implementación interna, lo que facilita la creación de aplicaciones complejas y la integración de diferentes sistemas.

Son cuatro tecnologías las que se han estandarizado como las más utilizadas para la creación de APIs: REST, SOAP, GraphQL y gRPC.

1. **REST - El Estándar Atemporal**

- **Ventajas:** Maduro y ampliamente adoptado, simple y flexible, sin estado, múltiples tipos de medios.
- **Limitaciones:** Sobre-recuperación y sub-recuperación, complejidad de versionado, capacidades de tiempo real limitadas.
- **Mejores Casos de Uso:** APIs públicas, operaciones CRUD simples, requisitos de tiempo real limitados.
- **Consideraciones:** Versionado y escalabilidad para grandes bases de usuarios, impacto de la sobre/sub-recuperación.

2. **SOAP - El Clásico Empresarial**

- **Ventajas:** Protocolo estandarizado, seguridad robusta (WS-Security), transacciones y confiabilidad.
- **Limitaciones:** Verbosidad, complejidad, menor flexibilidad que REST.
- **Mejores Casos de Uso:** Sistemas empresariales, integraciones complejas, requisitos de seguridad estrictos.
- **Consideraciones:** Complejidad del protocolo y herramientas, justificación del uso frente a REST.

3. **GraphQL - El Orquestador Dinámico**

- **Ventajas:** Obtención de datos impulsada por el cliente (reduce la sobre-recuperación), relaciones de datos complejas eficientes, actualizaciones en tiempo real (subscriptions), esquema flexible.
- **Limitaciones:** Mayor complejidad del servidor, curva de aprendizaje, ecosistema de herramientas en evolución.
- **Mejores Casos de Uso:** Aplicaciones de una sola página (SPAs), estructuras de datos complejas, actualizaciones y suscripciones en tiempo real.
- **Consideraciones:** Complejidad del servidor e impacto en el rendimiento, documentación y herramientas para desarrolladores.

4. **gRPC - El Conducto de Alto Rendimiento**

- **Ventajas:** Alto rendimiento (HTTP/2, Protocol Buffers), soporte de streaming, fuertemente tipado, herramientas maduras.
- **Limitaciones:** Curva de aprendizaje más pronunciada, menos flexible, adopción menos extendida.
- **Mejores Casos de Uso:** Comunicación entre microservicios de alto rendimiento, escenarios de streaming, operaciones intensivas en datos.
- **Consideraciones:** Complejidad de adopción de gRPC y Protocol Buffers, justificación del esfuerzo de desarrollo por las ganancias de rendimiento.

Las **APIs REST (Representational State Transfer)**, estrategia a seguir para la creación del backend del proyecto, se han convertido en el paradigma dominante para diseñar estas interfaces debido a su simplicidad, escalabilidad y flexibilidad. Una API REST define un conjunto de reglas y convenciones para que los sistemas puedan comunicarse a través del protocolo HTTP.

La creación de una API REST que sirva al frontend implica varios pasos clave:

- **Definición de Recursos:** Se identifican los recursos que la API expondrá (por ejemplo, usuarios, productos, pedidos). Cada recurso tiene una URI (Uniform Resource Identifier) única.
- **Uso de Métodos HTTP [10]:** HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs. Cada uno de ellos implementan una semántica diferente, pero algunas características similares son compartidas por un grupo de ellos: ej. un request method puede ser safe, idempotent, o cacheable.

Los más usados son :

- GET: Solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
- HEAD: Similar a GET, pero no devuelve el cuerpo de la respuesta, solo los encabezados. Se utiliza para obtener metadatos.
- POST: Se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
- PUT: Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
- PATCH: Aplica modificaciones parciales a un recurso.
- OPTIONS: Describe las opciones de comunicación para el recurso de destino. Permite al cliente conocer las capacidades del servidor.
- DELETE: Borra un recurso en específico.

- **Representación de Datos:** Los datos intercambiados entre el cliente y el servidor suelen estar en formato JSON (JavaScript Object Notation) debido a su ligereza y facilidad de parseo por parte de los navegadores y la mayoría de los lenguajes de programación. XML también puede ser utilizado, aunque es menos común en APIs modernas orientadas a frontend.
- **Statelessness (Ausencia de Estado):** Cada petición del cliente al servidor debe contener toda la información necesaria para que el servidor la entienda y procese. El servidor no almacena ningún estado de la sesión del cliente entre peticiones. Esto simplifica el diseño del servidor y mejora la escalabilidad.
- **Uso de Códigos de Estado HTTP:** Se utilizan códigos de estado HTTP para indicar el resultado de una petición (por ejemplo, 200 OK, 201 Created, 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error).

El backend desarrolla estos endpoints de la API REST, implementando la lógica necesaria para cada operación. El frontend, a su vez, realiza peticiones HTTP (utilizando APIs del navegador como Fetch o bibliotecas como Axios) a estas URLs para enviar y recibir datos, actualizando dinámicamente la interfaz de usuario sin necesidad de recargar la página completa. Este desacoplamiento entre frontend y backend permite que ambos puedan desarrollarse, probarse, desplegarse y escalarse de forma independiente, facilitando la colaboración entre equipos y la adopción de diferentes tecnologías para cada capa. Además, una API REST bien diseñada puede servir no solo a una aplicación web, sino también a aplicaciones móviles u otros servicios, promoviendo la reutilización y la interoperabilidad entre sistemas heterogéneos, tal como se mencionaba al inicio.

La tendencia hacia arquitecturas de microservicios en el backend ha reforzado aún más la importancia de las APIs REST bien definidas, ya que cada microservicio suele exponer su funcionalidad a través de una API. En este contexto, herramientas como OpenAPI (anteriormente Swagger) para la definición y documentación de APIs, y soluciones de API Gateway para la gestión, seguridad y monitorización del tráfico de las APIs, se han vuelto indispensables en el desarrollo de servicios web modernos y robustos.

3.4.1. Arquitecturas de software

La elección de la arquitectura backend es una decisión fundamental en el desarrollo de cualquier aplicación web, impactando directamente en su escalabilidad, mantenibilidad, flexibilidad y velocidad de desarrollo. Para un sistema de gestión de horarios académicos como el propuesto, que potencialmente puede crecer en complejidad y número de usuarios, la comparación entre los enfoques monolítico y de microservicios es particularmente relevante.

Si bien la arquitectura monolítica ha sido tradicionalmente un punto de partida, la creciente complejidad de los sistemas y la necesidad de agilidad han impulsado la adopción y evolución de diversos paradigmas arquitectónicos. A continuación, se presenta una exposición de diferentes arquitecturas de software [11] implementadas

en sistemas backend, analizando sus características y su posición en el espectro que va desde el monolito hasta los microservicios.

Arquitectura en Capas (Layered Architecture / N-Tier Architecture) [12]

Descripción: Este es uno de los patrones arquitectónicos más establecidos y fundamentales. Consiste en la organización del código en capas horizontales, donde cada capa posee una responsabilidad específica y se comunica, por lo general, únicamente con las capas adyacentes (superior e inferior). Las capas típicas suelen ser:

- *Capa de Presentación (o Interfaz de Usuario):* Gestiona la interacción con el usuario o sistemas cliente. En el contexto de un backend, esta capa a menudo se materializa como la API que gestiona las solicitudes HTTP.
- *Capa de Aplicación (o Lógica de Negocio):* Contiene la lógica de negocio central y orquesta las tareas y flujos de trabajo.
- *Capa de Dominio (o Modelo de Negocio):* Representa las entidades, los objetos de valor y las reglas inherentes al dominio del negocio.
- *Capa de Acceso a Datos (o Persistencia):* Encargada de la comunicación con los sistemas de almacenamiento de datos, como bases de datos.
- *Capa de Infraestructura:* Provee servicios técnicos transversales, tales como logging, monitorización, y comunicación de red.

Posicionamiento y Evolución: Una aplicación monolítica frecuentemente se estructura internamente siguiendo una arquitectura en capas. Aunque este patrón no descompone el sistema en servicios desplegables de forma independiente, sí promueve una modularización interna y una clara separación de responsabilidades (Separation of Concerns), lo cual es un primer paso crucial para gestionar la complejidad y facilitar la mantenibilidad de un sistema antes de considerar arquitecturas más distribuidas.

Arquitectura Orientada a Servicios (SOA - Service-Oriented Architecture) [13]

Descripción: SOA es un paradigma de diseño que estructura una aplicación como una colección de servicios que se comunican entre sí. Estos servicios encapsulan funcionalidades de negocio discretas y pueden ser accedidos a través de la red. A menudo, los servicios en SOA son de grano más grueso en comparación con los microservicios. La comunicación se estandarizó frecuentemente mediante protocolos como SOAP (Simple Object Access Protocol) sobre HTTP, y es común el uso de un Enterprise Service Bus (ESB) para la mediación, el enrutamiento y la transformación de mensajes entre servicios.

Características Clave: Fomenta la reutilización de servicios a nivel empresarial, la interoperabilidad entre sistemas heterogéneos y el descubrimiento dinámico de servicios.

Posicionamiento y Evolución: SOA representó un avance significativo respecto a los monolitos, permitiendo una descomposición más formal y orientada a negocio. Puede considerarse un precursor importante de la arquitectura de microservicios. Sin embargo, SOA a menudo implicaba una mayor sobrecarga en términos de estándares, una gobernanza más centralizada y, en ocasiones, cuellos de botella debidos al ESB, aspectos que la arquitectura de microservicios busca simplificar o descentralizar.

Arquitectura Dirigida por Eventos (EDA - Event-Driven Architecture) [14]

Descripción: En una EDA, el flujo de la aplicación es determinado por la ocurrencia de eventos. Los eventos son notificaciones que representan un cambio de estado significativo o un suceso relevante dentro del sistema (por ejemplo, "PedidoRealizado", "InventarioBajo"). Los componentes del sistema, denominados productores de eventos, publican estos eventos en un canal o bus de eventos (gestionado por un bróker de mensajes como Apache Kafka, RabbitMQ o Google Cloud Pub/Sub). Otros componentes, los consumidores de eventos, se suscriben a los eventos que les conciernen y reaccionan a ellos de forma asíncrona.

Estilos Principales: Se distinguen dos topologías principales: el *mediador de eventos*, donde un componente central orquesta el flujo de eventos, y el *bróker de eventos*, que facilita una mayor desacoplamiento entre publicadores y suscriptores.

Posicionamiento y Evolución: EDA es altamente compatible y a menudo se utiliza en conjunción con la arquitectura de microservicios para lograr una comunicación asíncrona, resiliente y escalable. Permite un desacoplamiento profundo entre servicios, mejorando la tolerancia a fallos y la capacidad de respuesta del sistema. También se emplea para modernizar sistemas monolíticos, permitiendo integrar nuevas funcionalidades de forma reactiva o desacoplar módulos existentes.

Arquitectura "Serverless" y Nativa de la Nube [15]

Descripción: Este enfoque se centra en la ejecución de código sin necesidad de gestionar servidores o infraestructura subyacente. Los desarrolladores escriben funciones que se ejecutan en respuesta a eventos, y el proveedor de la nube (como AWS Lambda, Azure Functions o Google Cloud Functions) se encarga de la infraestructura, escalabilidad y disponibilidad. Las aplicaciones nativas de la nube están diseñadas para aprovechar al máximo las capacidades de la nube, como el escalado automático, la alta disponibilidad y los servicios gestionados.

Características Clave: Despliegue basado en funciones, escalabilidad automática, pago por uso (se paga solo por el tiempo de ejecución), y enfoque en eventos y microservicios.

Posicionamiento y Evolución: La arquitectura serverless es una evolución natural hacia una mayor abstracción y simplificación del desarrollo backend. Permite a los equipos centrarse en la lógica de negocio sin preocuparse por la infraestructura subyacente. Sin embargo, puede introducir desafíos relacionados con el estado, la latencia y el control sobre el entorno de ejecución. Este enfoque es especialmente adecuado para aplicaciones que requieren escalabilidad extrema y una alta disponibilidad, como sistemas de gestión de horarios académicos que pueden experimentar picos de carga durante períodos específicos (por ejemplo, al inicio del semestre).

Arquitectura de Microservicios [16]

Descripción: Esta arquitectura estructura una aplicación como una suite de pequeños servicios independientes, cada uno enfocado en una capacidad de negocio específica y bien delimitada (Bounded Context). Cada microservicio es autónomo, lo que implica que puede ser desarrollado, desplegado, escalado y gestionado de forma independiente. Típicamente, cada servicio posee su propia base de datos para asegurar un bajo acoplamiento y se comunica con otros servicios a través de APIs ligeras y bien definidas, comúnmente utilizando HTTP/REST, gRPC, o a través de mensajería asíncrona.

Características Clave: Despliegue independiente y continuo, escalabilidad granular (se escalan solo los servicios que lo necesitan), flexibilidad tecnológica (posibilidad de usar diferentes stacks tecnológicos por servicio), aislamiento de fallos (un fallo en un servicio no debería derribar todo el sistema), y alineación con equipos de desarrollo pequeños y autónomos (Conway's Law).

Posicionamiento y Evolución: La arquitectura de microservicios se considera una evolución directa y una alternativa robusta para superar las limitaciones intrínsecas de los sistemas monolíticos, especialmente en contextos de alta complejidad, crecimiento rápido y necesidad de agilidad. Aborda desafíos como la dificultad para escalar, la complejidad en el mantenimiento, la lentitud en la adopción de nuevas tecnologías, los ciclos de despliegue prolongados y el alto acoplamiento que caracterizan a las grandes aplicaciones monolíticas.

Conclusión: Progresión y Criterios de Selección

El panorama de arquitecturas backend ha evolucionado desde la simplicidad inicial de los sistemas **monolíticos**, pasando por la modularización interna de la **arquitectura en capas**, hacia la descomposición en servicios con **SOA**.

Posteriormente, paradigmas como la **arquitectura dirigida por eventos** han ganado tracción para mejorar el desacoplamiento y la resiliencia, mientras que enfoques como la **arquitectura basada en el espacio** y los **principios nativos de la nube** abordan la escalabilidad extrema y la eficiencia en entornos cloud.

Finalmente, la **arquitectura de microservicios**, arquitectura a adoptar en el backend del sistema, emerge como un enfoque predominante para construir sistemas complejos, distribuidos y altamente escalables, ofreciendo un alto grado de agilidad y autonomía. No obstante, es crucial destacar que no existe una arquitectura universalmente superior. La selección de la arquitectura más adecuada debe ser el resultado de un análisis cuidadoso de los requisitos específicos del proyecto, el contexto del negocio, las capacidades del equipo de desarrollo, las proyecciones de escalabilidad y los compromisos (trade-offs) inherentes a cada patrón. En muchos sistemas del mundo real, es común encontrar combinaciones pragmáticas de estos patrones arquitectónicos.

3.4.2. Tecnologías de desarrollo

En el ámbito del desarrollo de software, la elección de las tecnologías adecuadas es crucial para el éxito de un proyecto. En este apartado, se presentan las principales tecnologías que se han considerado para el desarrollo del sistema de gestión de horarios académicos.

Tecnologías y lenguajes backend

1. **Java con Spring Framework (Spring Boot y Spring Cloud)**[17]: Java es un lenguaje de programación robusto, maduro y ampliamente adoptado en el desarrollo de aplicaciones empresariales a gran escala. Su ecosistema es vasto, con una gran comunidad y un fuerte enfoque en el rendimiento y la seguridad. Spring Boot simplifica drásticamente la creación de aplicaciones basadas en Spring, ofreciendo auto-configuración, servidores web embebidos (como Tomcat o Jetty) y una gestión de dependencias simplificada, lo que lo convierte en una opción popular para el desarrollo rápido de microservicios listos para producción. De hecho, se considera el estándar de facto para microservicios en Java.

Para arquitecturas de microservicios, Spring Cloud complementa a Spring Boot proporcionando un conjunto de herramientas y patrones para construir sistemas distribuidos resilientes y escalables. Esto incluye soluciones para el descubrimiento de servicios (permitiendo que los servicios se encuentren dinámicamente en la red), balanceo de carga (distribuyendo las solicitudes entre múltiples instancias de un servicio), pasarelas API (un punto de entrada único para todas las solicitudes de los clientes), interruptores de circuito (para prevenir fallos en cascada) y gestión de configuración distribuida. Dada la potencial complejidad de un sistema de gestión de horarios universitarios, especialmente si se opta por microservicios, la madurez y el soporte integral de Spring Cloud para estos patrones son altamente relevantes.

2. **.NET Core[18]**: .NET Core (ahora parte de .NET 5 y versiones posteriores) es un framework de desarrollo de aplicaciones multiplataforma, de código abierto y de alto rendimiento mantenido por Microsoft. Es una opción sólida para construir microservicios, con excelente soporte para la creación de APIs RESTful, contenedores Docker y despliegue en plataformas de orquestación como Kubernetes. Ofrece características como escalabilidad, un modelo de entrega continua, herramientas para operaciones CRUD, soporte para comunicación síncrona y asíncrona entre servicios, y la implementación de patrones de diseño avanzados como CQRS (Command and Query Responsibility Segregation) y Event Sourcing. También se integra bien con tecnologías de caché como Redis y proporciona mecanismos de seguridad robustos mediante OAuth2 y OpenID Connect. Para equipos con experiencia en el ecosistema Microsoft o que buscan una alternativa de alto rendimiento a Java, .NET Core es una opción muy competente.
3. **Node.js con Express.js[19]**: Node.js es un entorno de ejecución para JavaScript del lado del servidor, construido sobre el motor V8 de Chrome. Su principal característica distintiva es su modelo de E/S (Entrada/Salida) asíncrono y sin bloqueo, orientado a eventos. Esto lo hace particularmente eficiente para aplicaciones que manejan un gran número de conexiones concurrentes y operaciones de E/S intensivas, como aplicaciones en tiempo real o APIs que actúan como fachadas para otros servicios. Express.js es un framework web minimalista y flexible para Node.js, ampliamente utilizado para construir APIs RESTful y microservicios ligeros. Su simplicidad y el vasto ecosistema de paquetes disponibles a través de npm (Node Package Manager) permiten un desarrollo rápido. En el contexto de un sistema de gestión de horarios, Node.js con Express podría ser adecuado para microservicios específicos que se beneficien de su naturaleza asíncrona, como un servicio de notificaciones en tiempo real o una pasarela API ligera.
4. **Python con Django/Flask[20][21]**: Python es un lenguaje de programación conocido por su sintaxis clara, legibilidad y alta productividad del desarrollador. Para el desarrollo web, existen dos frameworks principales: Django y Flask. Django es un framework de “baterías incluidas” que proporciona muchas funcionalidades listas para usar, como un ORM (Object-Relational Mapper), un panel de administración y un sistema de autenticación. Esto puede acelerar el desarrollo de aplicaciones web completas. Flask, por otro lado, es un micro-framework que proporciona las herramientas esenciales para construir aplicaciones web, ofreciendo mayor flexibilidad y dejando más decisiones de diseño al desarrollador.
Para microservicios, Flask es a menudo la opción preferida debido a su ligereza y minimalismo, permitiendo construir servicios pequeños y enfocados sin el overhead de Django. Sin embargo, Django podría considerarse si un microservicio específico se beneficia significativamente de sus características integradas. Ambos frameworks cuentan con comunidades maduras y un amplio soporte.
5. **Consideraciones para la Elección del Stack Backend**: La elección del stack

tecnológico para el backend está intrínsecamente ligada a la arquitectura general seleccionada (monolito o microservicios) y, de manera crucial, a la experiencia y familiaridad del equipo de desarrollo. Si se adopta una arquitectura de microservicios, frameworks con un robusto soporte para patrones de sistemas distribuidos, como Spring Cloud para el ecosistema Java, ofrecen ventajas considerables en términos de gestión y resiliencia. Por otro lado, si la velocidad de desarrollo inicial para un monolito o para microservicios más simples es prioritaria, alternativas como Python con Flask o Node.js con Express pueden permitir un arranque más rápido. La disponibilidad de desarrolladores con experiencia en un stack tecnológico particular, por ejemplo, Java en entornos corporativos o universitarios, puede ser un factor determinante, incluso si otro stack pudiera parecer marginalmente superior desde una perspectiva puramente técnica.

Además, la “madurez” de un lenguaje y framework, como es el caso de Java y Spring, no solo implica estabilidad del código base, sino también la existencia de un vasto ecosistema de herramientas, bibliotecas probadas y soluciones documentadas para problemas comunes. Este ecosistema reduce el riesgo inherente al desarrollo y puede acortar los tiempos de desarrollo a largo plazo al evitar la necesidad de “reinventar la rueda”. Para un sistema potencialmente complejo como la gestión de horarios académicos, que podría requerir integraciones con sistemas universitarios preexistentes, autenticación robusta y una gestión de datos sofisticada, la amplitud y profundidad de un ecosistema maduro pueden ser más beneficiosas que un framework más nuevo o ligero que exija más integraciones manuales o el desarrollo de componentes básicos desde cero.

Tecnologías y lenguajes frontend

En el contexto de desarrollo web orientado al cliente, la elección de tecnologías y lenguajes es fundamental para garantizar una experiencia de usuario fluida y eficiente. A continuación, se presentan las principales tecnologías y lenguajes considerados para el desarrollo del frontend del sistema de gestión de horarios académicos.

En cuanto a lenguajes de programación, **JavaScript**[22] es el lenguaje de programación más utilizado en el desarrollo frontend. Es un lenguaje interpretado y orientado a objetos que permite la creación de aplicaciones web interactivas y dinámicas. JavaScript se ejecuta en el navegador del cliente, lo que permite la manipulación del DOM (Document Object Model) y la interacción con el usuario sin necesidad de recargar la página.

El **HTML (HyperText Markup Language)**[23] es el lenguaje de marcado utilizado para estructurar el contenido de las páginas web. HTML define la estructura y el contenido de una página, incluyendo texto, imágenes, enlaces y otros elementos multimedia. Junto con CSS (Cascading Style Sheets), que se utiliza para definir la presentación y el diseño visual de las páginas web, HTML forma la base del desarrollo frontend.

El CSS[24] es un lenguaje de estilo utilizado para describir la presentación de un documento HTML. CSS permite definir el diseño, los colores, las fuentes y otros aspectos visuales de una página web. Junto con HTML y JavaScript, CSS forma el trinomio fundamental del desarrollo frontend.

Respecto a frameworks y bibliotecas, existen varias opciones populares que facilitan el desarrollo frontend:

1. **React[25]**: Es una biblioteca de JavaScript desarrollada por Facebook para construir interfaces de usuario. React se basa en componentes reutilizables y permite la creación de aplicaciones web dinámicas y escalables. Su enfoque basado en el estado y el ciclo de vida de los componentes facilita la gestión de la interactividad y la actualización eficiente del DOM.
2. **Angular[26]**: Es un framework de desarrollo web desarrollado por Google que permite la creación de aplicaciones web de una sola página (SPA). Angular utiliza TypeScript, un superconjunto de JavaScript, y ofrece una arquitectura basada en componentes, inyección de dependencias y un sistema de enrutamiento robusto.
3. **Vue.js[27]**: Es un framework progresivo para construir interfaces de usuario. Vue.js es fácil de integrar con otras bibliotecas o proyectos existentes y se centra en la capa de vista. Su enfoque reactivo y su sistema de componentes lo hacen adecuado para aplicaciones pequeñas y grandes.

3.4.3. Sistemas de almacenamiento de datos

Los sistemas de almacenamiento de datos son fundamentales en el desarrollo de aplicaciones web, ya que permiten la persistencia y gestión eficiente de la información. En el contexto del sistema de gestión de horarios académicos, se han considerado varias opciones para el almacenamiento de datos.

1. **Bases de Datos Relacionales (RDBMS)**: Estas bases de datos utilizan un modelo tabular para almacenar datos y son ideales para aplicaciones que requieren transacciones complejas y relaciones entre datos. Ejemplos populares incluyen MySQL[28], PostgreSQL[29] y Microsoft SQL Server[30]. Estas bases de datos son adecuadas para el sistema de gestión de horarios, ya que permiten la creación de relaciones entre entidades como grados, asignaturas, grupos y clases.

La ventaja de utilizar una base de datos relacional es la capacidad de realizar consultas complejas y garantizar la integridad referencial de los datos. Sin embargo, pueden presentar limitaciones en términos de escalabilidad horizontal (agregar más servidores para manejar cargas de trabajo) y flexibilidad en la estructura de datos.

2. **Bases de Datos NoSQL**: Estas bases de datos son ideales para aplicaciones que requieren alta escalabilidad y flexibilidad en la estructura de datos. Existen varios tipos de bases de datos NoSQL, como bases de datos

orientadas a documentos (MongoDB)[31], bases de datos clave-valor (Redis)[32], bases de datos en columna (Cassandra)[33] y bases de datos orientadas a grafos (Neo4j)[34]. Las bases de datos NoSQL son adecuadas para aplicaciones que manejan grandes volúmenes de datos no estructurados o semi-estructurados.

La ventaja de utilizar una base de datos NoSQL es la capacidad de escalar horizontalmente y manejar grandes volúmenes de datos. Sin embargo, pueden presentar limitaciones en términos de transacciones complejas y consistencia de datos.

3. **Bases de Datos en Memoria:** Estas bases de datos almacenan datos en la memoria RAM, lo que permite un acceso extremadamente rápido. Son ideales para aplicaciones que requieren baja latencia y alto rendimiento, como sistemas de caché o análisis en tiempo real. Ejemplos populares incluyen Redis y Memcached[35].

La ventaja de utilizar una base de datos en memoria es la velocidad de acceso a los datos. Sin embargo, pueden presentar limitaciones en términos de persistencia de datos y capacidad de almacenamiento.

4. **Bases de Datos en la Nube:** Estas bases de datos son ofrecidas como servicios en la nube y permiten a las empresas escalar y gestionar sus datos sin preocuparse por la infraestructura subyacente. Ejemplos populares incluyen Amazon RDS[36], Google Cloud SQL[37] y Azure Cosmos DB[38].

La ventaja de utilizar una base de datos en la nube es la escalabilidad y la facilidad de gestión. Sin embargo, pueden presentar limitaciones en términos de control sobre la infraestructura y costos a largo plazo.

5. **Bases de Datos Híbridas:** Estas bases de datos combinan características de bases de datos relacionales y NoSQL, permitiendo a las aplicaciones aprovechar lo mejor de ambos mundos. Ejemplos populares incluyen Amazon Aurora[39] y Google Cloud Spanner[40].

La ventaja de utilizar una base de datos híbrida es la flexibilidad y la capacidad de manejar diferentes tipos de datos. Sin embargo, pueden presentar limitaciones en términos de complejidad y costos.

3.4.4. Autenticación y autorización

La autenticación y autorización son componentes críticos en el desarrollo de aplicaciones web, especialmente en sistemas que manejan datos sensibles o requieren control de acceso granular. En el contexto del sistema de gestión de horarios académicos, se han considerado varias opciones para implementar la autenticación y autorización.

1. **Autenticación basada en formularios[41]:** Este es el método más común de autenticación en aplicaciones web. Los usuarios ingresan sus credenciales (nombre de usuario y contraseña) en un formulario, que se envía al servidor para su validación. Si las credenciales son correctas, el servidor crea una sesión y devuelve un token de sesión al cliente. Este token se utiliza para

autenticar las solicitudes posteriores.

La ventaja de este método es su simplicidad y facilidad de implementación. Sin embargo, puede presentar limitaciones en términos de seguridad (por ejemplo, ataques de fuerza bruta) y experiencia del usuario (por ejemplo, necesidad de recordar contraseñas).

2. **Autenticación basada en tokens[42]:** Este método utiliza tokens (como [JWT](#) - JSON Web Tokens) para autenticar a los usuarios. Después de que el usuario ingresa sus credenciales, el servidor genera un token firmado y lo envía al cliente. Este token se incluye en las solicitudes posteriores para autenticar al usuario. La ventaja de este método es que no requiere mantener sesiones en el servidor, lo que facilita la escalabilidad y la interoperabilidad entre diferentes servicios, lo que casa a la perfección con el concepto de API REST anteriormente mencionado.
Sin embargo, puede presentar limitaciones en términos de seguridad (por ejemplo, tokens robados) y complejidad de implementación (por ejemplo, gestión de la expiración de tokens).
3. **Autenticación basada en OAuth2[43]:** OAuth2 es un protocolo de autorización que permite a los usuarios otorgar acceso limitado a sus recursos a aplicaciones de terceros sin compartir sus credenciales. Este método es ampliamente utilizado por plataformas como Google, Facebook y Twitter para permitir el inicio de sesión único ([SSO](#)) en aplicaciones de terceros.
La ventaja de este método es su flexibilidad y capacidad para integrar múltiples proveedores de identidad. Sin embargo, puede presentar limitaciones en términos de complejidad de implementación y dependencia de terceros.
4. **Autenticación multifactor (MFA)[44]:** Este método combina múltiples factores de autenticación (por ejemplo, contraseña y código enviado por SMS) para aumentar la seguridad. La ventaja de este método es su capacidad para prevenir accesos no autorizados incluso si las credenciales son comprometidas. Sin embargo, puede presentar limitaciones en términos de experiencia del usuario (por ejemplo, necesidad de ingresar múltiples factores) y complejidad de implementación.
5. **Autenticación basada en LDAP[45]:** LDAP (Lightweight Directory Access Protocol) es un protocolo utilizado para acceder y gestionar servicios de directorio. Este método permite autenticar a los usuarios utilizando un servidor LDAP, que almacena información sobre usuarios y grupos. La ventaja de este método es su capacidad para integrar múltiples sistemas y aplicaciones. Sin embargo, puede presentar limitaciones en términos de complejidad de implementación y dependencia de terceros.

3.4.5. Comunicación entre servicios

La comunicación entre servicios es un aspecto fundamental en el desarrollo de aplicaciones distribuidas, especialmente en arquitecturas de microservicios.

Existen varias opciones para implementar la comunicación entre servicios, cada una con sus ventajas y desventajas.

EL paso de información entre servicios puede realizarse de diferentes maneras, dependiendo de la arquitectura y los requisitos del sistema. A continuación, se presentan las principales opciones para la comunicación entre servicios:

- **Comunicación síncrona:** Este enfoque implica que un servicio realiza una solicitud a otro servicio y espera una respuesta antes de continuar. Los protocolos más comunes para la comunicación síncrona son HTTP/REST y gRPC. La ventaja de este enfoque es su simplicidad y facilidad de implementación. Sin embargo, puede presentar limitaciones en términos de latencia y disponibilidad, ya que un fallo en un servicio puede afectar a otros servicios que dependen de él. Tecnologías comunes (descritas en la sección 3.4):
 - [HTTP/REST\[46\]](#)
 - [SOAP\[47\]](#)
 - [gRPC\[48\]](#)
 - [GraphQL\[49\]](#)
- **Comunicación asíncrona:** Este enfoque permite que un servicio envíe un mensaje a otro servicio sin esperar una respuesta inmediata. Los mensajes se envían a través de un sistema de mensajería (como RabbitMQ, Apache Kafka o Amazon SQS) y pueden ser procesados en paralelo por los servicios receptores. La ventaja de este enfoque es su capacidad para manejar cargas de trabajo variables y mejorar la resiliencia del sistema. Sin embargo, puede presentar limitaciones en términos de complejidad de implementación y gestión de errores. Tecnologías comunes:
 - [RabbitMQ\[50\]](#): RabbitMQ es un sistema de mensajería de código abierto que implementa el protocolo AMQP (Advanced Message Queuing Protocol). Permite la comunicación asíncrona entre servicios mediante el uso de colas de mensajes, lo que facilita la desacoplación y la escalabilidad. Tiene la capacidad de manejar grandes volúmenes de mensajes y ofrece características como confirmaciones de entrega, enrutamiento avanzado y soporte para múltiples protocolos de mensajería, como MQTT y STOMP.
 - [Apache Kafka\[51\]](#): Kafka es una plataforma de mensajería distribuida diseñada para manejar flujos de datos en tiempo real. Utiliza un modelo de publicación/suscripción y permite la transmisión de mensajes entre productores y consumidores a través de temas (topics). Kafka es altamente escalable y tolerante a fallos, lo que lo convierte en una opción popular para aplicaciones que requieren procesamiento de eventos en tiempo real. Posee la ventaja de permitir la persistencia de mensajes, lo que significa que los mensajes pueden ser almacenados y procesados

posteriormente, lo que es útil para la recuperación ante fallos y el análisis de datos históricos.

- **Amazon SQS[52]:** Amazon Simple Queue Service (SQS) es un servicio de mensajería completamente gestionado que permite la comunicación asíncrona entre servicios en la nube de Amazon Web Services (AWS). SQS permite a los desarrolladores enviar, recibir y eliminar mensajes entre componentes de aplicaciones distribuidas. Ofrece características como escalabilidad automática, alta disponibilidad y seguridad integrada. SQS es ideal para aplicaciones que requieren desacoplamiento entre componentes y procesamiento asíncrono de mensajes. Además, se integra fácilmente con otros servicios de AWS, lo que facilita la construcción de arquitecturas distribuidas en la nube.

3.4.6. Despliegue de sistemas

El despliegue de sistemas es un aspecto crítico en el desarrollo de aplicaciones web, ya que implica la implementación y gestión de la infraestructura necesaria para ejecutar la aplicación. Además involucra desde la configuración de servidores y redes hasta la gestión de bases de datos y servicios de almacenamiento. En el contexto del sistema de gestión de horarios académicos, se han considerado varias opciones para el despliegue del sistema.

- **Despliegue en servidores físicos:** Este enfoque implica la instalación y configuración de la aplicación en servidores físicos dedicados. Aunque ofrece un alto grado de control sobre la infraestructura, puede ser costoso y difícil de escalar. Además, requiere una gestión constante del hardware y el software.
- **Despliegue en máquinas virtuales:** Este enfoque utiliza hipervisores para crear máquinas virtuales (VM) que ejecutan la aplicación. Las VM permiten una mayor flexibilidad y escalabilidad en comparación con los servidores físicos, pero pueden presentar limitaciones en términos de rendimiento y gestión de recursos.
- **Despliegue en contenedores:** Este enfoque utiliza tecnologías de contenedorización (como Docker) para empaquetar la aplicación y sus dependencias en contenedores ligeros y portátiles. Los contenedores permiten un despliegue rápido y eficiente, así como una mayor escalabilidad y flexibilidad. Además, se integran bien con plataformas de orquestación como Kubernetes.
- **Despliegue en la nube:** Este enfoque utiliza servicios en la nube (como Amazon Web Services, Google Cloud Platform o Microsoft Azure) para alojar la aplicación. La computación en la nube permite una escalabilidad casi infinita, alta disponibilidad y gestión simplificada de la infraestructura. Además, ofrece servicios adicionales como bases de datos gestionadas, almacenamiento y análisis de datos.
- **Despliegue híbrido:** Este enfoque combina elementos de despliegue en

servidores físicos, máquinas virtuales y la nube. Permite a las organizaciones aprovechar lo mejor de cada enfoque, adaptándose a sus necesidades específicas y requisitos de seguridad.

Tecnologías de despliegue

En cuanto a despliegue de servicios, existen varias tecnologías y herramientas que facilitan la implementación y gestión de aplicaciones en diferentes entornos. A continuación, se presentan algunas de las tecnologías más relevantes para este tipo de sistemas pueden ser:

- **Docker[53]:** Docker es una plataforma de contenedorización que permite empaquetar aplicaciones y sus dependencias en contenedores ligeros y portátiles. Los contenedores son independientes del sistema operativo subyacente, lo que facilita el despliegue y la escalabilidad de aplicaciones en diferentes entornos. Docker es ampliamente utilizado en arquitecturas de microservicios y DevOps.
- **Kubernetes[54]:** Kubernetes es un sistema de orquestación de contenedores que automatiza la implementación, escalado y gestión de aplicaciones en contenedores. Proporciona características como balanceo de carga, recuperación ante fallos y gestión de secretos, lo que lo convierte en una opción popular para gestionar aplicaciones distribuidas y microservicios.
- **Terraform[55]:** Terraform es una herramienta de infraestructura como código (IaC) que permite definir y gestionar la infraestructura mediante archivos de configuración. Terraform es compatible con múltiples proveedores de nube y permite crear, modificar y eliminar recursos de forma programática, facilitando la gestión de la infraestructura en entornos complejos.
- **Ansible[56]:** Ansible es una herramienta de automatización de TI que permite gestionar la configuración, el aprovisionamiento y la implementación de aplicaciones en servidores físicos, máquinas virtuales o contenedores. Utiliza un enfoque declarativo y se basa en archivos YAML para definir las configuraciones deseadas.
- **Jenkins[57]:** Jenkins es una herramienta de integración continua (CI) y entrega continua (CD) que permite automatizar el proceso de construcción, prueba y despliegue de aplicaciones. Jenkins se integra con múltiples herramientas y servicios, lo que facilita la implementación continua en diferentes entornos.

3.4.7. Pruebas y calidad del software

La calidad del software es un aspecto fundamental en el desarrollo de aplicaciones web, ya que garantiza que el sistema cumpla con los requisitos funcionales y no funcionales, así como con las expectativas de los usuarios. En el contexto del sistema de gestión de horarios académicos, se han considerado varias opciones para garantizar la calidad del software.

- **Pruebas unitarias:** Estas pruebas se centran en verificar el comportamiento de componentes individuales del sistema, como funciones o métodos. Las pruebas unitarias son fundamentales para garantizar que cada componente funcione correctamente y cumpla con los requisitos especificados. Herramientas populares para pruebas unitarias incluyen JUnit (Java), NUnit (.NET), PyTest (Python) y Jest (JavaScript).
- **Pruebas de integración:** Estas pruebas verifican la interacción entre diferentes componentes del sistema, asegurando que funcionen correctamente juntos. Las pruebas de integración son esenciales para identificar problemas de comunicación y dependencias entre componentes. Herramientas populares para pruebas de integración incluyen Postman (para APIs REST), TestNG (Java) y Mocha (JavaScript).
- **Pruebas funcionales:** Estas pruebas evalúan el comportamiento del sistema desde la perspectiva del usuario, asegurando que cumpla con los requisitos funcionales especificados. Las pruebas funcionales pueden ser manuales o automatizadas, y herramientas populares incluyen Selenium (para aplicaciones web), Cucumber (para pruebas basadas en comportamiento) y TestComplete.
- **Pruebas de seguridad:** Estas pruebas evalúan la seguridad del sistema, identificando vulnerabilidades y asegurando que cumpla con las mejores prácticas de seguridad. Las pruebas de seguridad son fundamentales para proteger los datos sensibles y garantizar la integridad del sistema. Herramientas populares para pruebas de seguridad incluyen OWASP ZAP, Burp Suite y Nessus.
- **Pruebas de usabilidad:** Estas pruebas evalúan la experiencia del usuario al interactuar con el sistema, asegurando que sea fácil de usar y cumpla con las expectativas de los usuarios. Las pruebas de usabilidad pueden ser manuales o automatizadas, y herramientas populares incluyen Hotjar, Crazy Egg y UserTesting.
- **Pruebas de carga:** Estas pruebas evalúan la capacidad del sistema para manejar un número específico de usuarios simultáneos y medir su rendimiento bajo diferentes condiciones de carga. Las pruebas de carga son esenciales para garantizar que el sistema sea escalable y responda adecuadamente a las demandas de los usuarios. Herramientas populares para pruebas de carga incluyen Apache JMeter, Gatling y LoadRunner.

4. Especificación de requisitos

El primer paso para el desarrollo de un sistema es la definición de los requisitos que este debe cumplir. Estos son las características y funcionalidades que el sistema debe tener para satisfacer las necesidades de los usuarios finales y cumplir con los objetivos del proyecto.

4.1. Recopilación de información

Para definir estos, se han utilizado diferentes técnicas de recopilación de información, como entrevistas, encuestas y análisis de documentos existentes.

Añadir a esta sección correos y peticiones al CSIRC, y propuesta al vicerrectorado de transformación digital de la UGR.

– Juanmi

- **Análisis de documentos existentes:** Se han revisado documentos y recursos existentes relacionados con la UGR, como el sitio web oficial, guías académicas y normativas internas. Esto ha permitido comprender mejor el contexto y la forma en la que los usuarios acceden a la información.
- **Encuestas:** Se ha realizado una encuesta a los miembros del departamento ICAR, y a compañeros estudiantes de la ETSIIT para obtener información sobre como confeccionan su horario y cuanto tiempo les toma con los sistemas actuales. Estas encuestas han permitido identificar las funcionalidades más valoradas y las áreas de mejora.
- **Entrevistas:** Se han llevado a cabo numerosas entrevistas con perfiles variados dentro de la institución para entender mejor las necesidades actuales de los usuarios:
 - **Jesús García Miranda - Profesor titular y Secretario de la ETSIIT:** (3 de febrero de 2025) En este punto del proyecto no se tenía muy claro cómo se generaba la información de horario escolar, y se necesitaba una visión general de cómo se gestionaba la información en la UGR. Jesús ofreció documentos y recursos para clarificar cómo se generaba y clasificaba esta información para su posterior entrega a la UGR. Además hablamos con varios integrantes encargados de la administración de estos procesos que nos aportaron más información sobre cómo se gestionaba la información de horarios y asignaturas. De esta manera se comprendió que estos procesos no eran igual en cada sede de la UGR, y que cada facultad tenía su propio sistema de gestión de horarios y asignaturas.

Fue una entrevista enriquecedora que ayudó a definir mejor el contexto

y las necesidades del sistema. Además comprendimos que lo ideal para nuestro sistema sería obtener la información de horarios y asignaturas directamente de la UGR, y no tener que gestionarla nosotros mismos.

- **Alberto Guillén Perales - Profesor titular en la ETSIIT y Director del Centro de Producción de Recursos para la Universidad Digital (Ceprud):** (25 de febrero de 2025) Esta reunión se planteó para comprobar la posibilidad de acceder a los siguientes recursos:

- Información del calendario académico de la UGR. (Información pública expuesta en varias páginas web de la UGR, como la página oficial de Grados UGR).
- Información de matriculaciones y asignaturas de los alumnos. (Información privada, no accesible públicamente).
- Acceso a la autenticación de la UGR. (Información privada, no accesible públicamente, y con proceso de solicitud establecido).

En esta reunión se concluye que en el tiempo restante del proyecto es complicado el acceso a cualquiera de estos recursos mediante un proceso de solicitud formal, ya que los procesos burocráticos de la UGR pueden demorarse varios meses.

El resto de la conversación se centró en alternativas para obtener la información necesaria para el desarrollo del sistema, y se decide construir un sistema de autenticación y autorización propio basado en los correos institucionales, y en utilizar técnicas de web scraping para obtener la información de horarios y asignaturas de los grados de la UGR.

Fue una reunión muy productiva que ayudó a definir mejor el contexto y las necesidades del sistema. Además comprendimos que lo ideal para nuestro sistema sería obtener la información de horarios y asignaturas directamente por nuestra cuenta y no depender de factores externos.

- **Juan Luis Jiménez Laredo - Profesor titular en la ETSIIT y Director del TFG “TempusUGR”:** (Reuniones semanales desde febrero de 2025 hasta junio de 2025) Durante estas reuniones se ha ido revisando el avance del proyecto, y se han ido definiendo y acotando los requisitos del sistema. Gracias a su punto de vista como profesor, y experiencia en el desarrollo de sistemas, se han podido identificar las funcionalidades más importantes y las áreas de mejora.

4.2. Personas

Las personas son representaciones ficticias de los usuarios finales del sistema. Estas se crean a partir de la investigación y el análisis de los usuarios reales, y se

utilizan para comprender mejor sus necesidades, comportamientos, inquietudes, objetivos, etc

Las personas ayudan a guiar el diseño y desarrollo del sistema, asegurando que se satisfacen las necesidades de los usuarios.

4.2.1. Personas del sistema

En este apartado se muestran las personas que se han definido en el sistema en las figuras 4.1, 4.2 y 4.3. Estas personas representan a los diferentes tipos de usuarios que interactuarán con el sistema, y se han creado a partir de la investigación y el análisis de los usuarios reales.

PERSONA #1: MANUEL RODRÍGUEZ JIMÉNEZ



DEMOGRAFÍA

Edad: 21
Género: Hombre
Profesión: Estudiante de Ingeniería Informática.
Nacionalidad: Perú, Ayacucho
Localización: Granada , Granada

METAS - INTERESES

- Acabar la carrera en 6 años.
- Viajar por el mundo.
- Hacer más deporte.

APASIONADO - SOCIABLE - INCANSABLE

" Si nuestra relación fuera un lenguaje de programación sería c++, porque tengo un puntero a tu corazón ".

PAIN POINTS - PREOCUPACIONES

- Tiene varias asignaturas de diferentes años, ver su horario es tedioso.

ESCENARIO

Necesita saber los horarios de los grupos de las asignaturas en los que está matriculado.

Figura 4.1: Persona 1: Alumno de la UGR

PERSONA #2: MARÍA COBOS MERINO



SERIA - INQUIETA - INTROVERTIDA

"El conocimiento no es un destino, sino un viaje constante. No teman las preguntas difíciles, abracen la curiosidad y nunca dejen de aprender."

Bio: Profesora titular en la UGR. Este año imparte clase en varias asignaturas tanto de grado como de máster.

DEMOGRAFÍA

Edad: 46
Género: Mujer
Profesión: Profesora titular en la Universidad de Granada.
Nacionalidad: España
Localización: Jaén, Baeza

METAS - INTERESES

- Mejorar su metodología de enseñanza.
- Tener menos carga de investigación.
- Organizar charlas de empresas en sus clases de forma extraescolar.

PAIN POINTS - PREOCUPACIONES

- Le es difícil organizar su horario al comienzo del cuatrimestre.

ESCENARIO

Quiere tener centralizado el horario de los grupos a los que imparte clase. Quiere comunicar de manera efectiva cuándo vienen empresas a dar charlas.

Figura 4.2: Persona 2: Profesor de la UGR

PERSONA #3: SERGIO HERNÁNDEZ POMÁRES



AMIGABLE - EMPÁTICO - RISUEÑO

"Cada día es una nueva oportunidad para aprender, crecer y acercarte un paso más a la mejor versión de ti mismo."

Bio: Secretario de la Facultad de Ciencias en la UGR. Además imparte clases en la misma sede.

DEMOGRAFÍA

Edad: 44
Género: Hombre
Profesión: Profesor titular en la Universidad de Granada.
Nacionalidad: España
Localización: Granada, Granada

METAS - INTERESES

- Promover una enseñanza orientada al estudiante.
- Facilitar el paso por la facultad.

PAIN POINTS - PREOCUPACIONES

- No consigue comunicar de manera efectiva eventos de su facultad.
- Quiere hacer saber a todos cuando imparte clases de recuperación.

ESCENARIO

Quiere tener centralizado el horario de los grupos a los que imparte clase y comunicar de manera efectiva cuándo son sus clases de recuperación.

Figura 4.3: Persona 3: "Administrador" de la UGR

4.3. Escenarios

Un escenario es una descripción narrativa de cómo un usuario interactúa con un sistema para lograr un objetivo específico. Los escenarios son herramientas útiles para comprender y comunicar los requisitos del sistema, ya que proporcionan un contexto claro y detallado sobre cómo se espera que funcione el sistema en situaciones del mundo real.

¿Por qué son importantes los escenarios?

- Ayudan a identificar y definir los requisitos del sistema de manera más clara y comprensible.
- Proporcionan un contexto para las decisiones de diseño y desarrollo, asegurando que se alineen con las necesidades del usuario.
- Facilitan la comunicación entre los miembros del equipo de desarrollo y los interesados, ya que son más fáciles de entender que los requisitos técnicos.
- Permiten identificar posibles problemas o desafíos en la interacción del usuario con el sistema antes de que se implemente.

4.3.1. Escenarios del sistema

Escenario 1: Manuel, el alumno organizado en medio del caos

Situación Actual: Manuel es estudiante de tercer año del Grado de Química en la Facultad de Ciencias de la UGR. Este curso, su horario es un verdadero rompecabezas: tiene asignaturas de primero, segundo y tercero. Para complicar aún más las cosas, varios grupos han cambiado de aula a última hora. Manuel se encuentra constantemente revisando múltiples documentos, correos electrónicos y tablones de anuncios para intentar confeccionar un horario coherente y no perderse ninguna clase. La incertidumbre sobre dónde y cuándo tiene cada asignatura le genera estrés y dificulta su planificación semanal.

Con el Sistema: Manuel accede a su panel personalizado. Allí, visualiza de forma clara y unificada su horario, con todas las asignaturas de los diferentes cursos que tiene matriculadas. La información de las aulas está completamente actualizada, reflejando los últimos cambios. Además, puede filtrar por formato mensual, semanal y diario, facilitando la consulta. Con una url para importar en un sistema de calendario externo, sincroniza este horario personalizado con su Google Calendar, teniendo toda su planificación académica integrada en su calendario digital habitual. Ya no tiene que preocuparse por buscar información dispersa o por posibles cambios de última hora, ya que la aplicación se encarga de mantener su horario al día.

Escenario 2: María, la profesora conectada con sus alumnos

Situación Actual: La profesora María imparte varias asignaturas en la UGR y utiliza el SWAD para comunicar anuncios importantes a sus alumnos, como clases de recuperación o charlas de profesionales invitados. Sin embargo, se da cuenta de que muchos estudiantes no acceden regularmente a la plataforma o no revisan los mensajes con la frecuencia necesaria, perdiéndose información valiosa. Para aquellos que sí ven el mensaje, recordar la fecha y hora del evento implica tener que buscarlo nuevamente en el SWAD. Esto dificulta la participación de los alumnos en actividades complementarias importantes para su formación.

Con el Sistema: María puede crear eventos directamente asociados a sus grupos de asignatura. Al hacerlo, el sistema automáticamente envía una notificación por correo electrónico a todos los alumnos inscritos en ese grupo, informándoles del evento con todos los detalles relevantes (fecha, hora, lugar, descripción). Además, este evento se añade automáticamente al calendario personal de cada alumno dentro de la aplicación, integrado con sus clases oficiales. María también puede sincronizar su propio calendario de eventos y clases con su Google Calendar, teniendo una visión completa de su agenda académica. De esta manera, la información importante llega directamente a los alumnos, aumentando la visibilidad y la participación en las actividades propuestas.

Escenario 3: Sergio, el administrador eficiente con información clara

Situación Actual: Sergio, el secretario de la ETSIIT, necesita tener una visión clara del horario de clases que se imparten en la facultad para diversas tareas de gestión y organización. Además, la facultad organiza regularmente seminarios de empresas, talleres y otros eventos de interés para los estudiantes. Actualmente, la comunicación de estos eventos se realiza principalmente por correo electrónico masivo, lo que a menudo resulta intrusivo y puede pasar desapercibido entre la gran cantidad de mensajes que reciben los alumnos. Sergio necesita una forma más efectiva y menos invasiva de informar sobre estos eventos a nivel de facultad.

Con el Sistema: Sergio puede visualizar el horario de todas las clases de la ETSIIT de forma organizada y sencilla a través de la interfaz del sistema. Además, tiene la capacidad de crear eventos a nivel de facultad (seminarios, talleres, etc.). Estos eventos se integran directamente en el calendario de todos los alumnos de la ETSIIT dentro de la aplicación, apareciendo junto con sus horarios de clase habituales. Aunque no se envía una notificación por correo electrónico para evitar la sobrecarga de información, los alumnos pueden ver fácilmente estos eventos al consultar su horario personalizado. De esta manera, la información importante a nivel de facultad está siempre accesible y visible para los estudiantes, mejorando la comunicación y la participación sin recurrir a métodos intrusivos.

4.4. Historias de usuario

Una historia de usuario es una explicación general e informal de una función de software escrita desde la perspectiva del usuario final. Su propósito es articular cómo proporcionará una función de software valor al cliente. [58].

Estas no usan un lenguaje técnico y preciso para definir y acotar los requisitos de un sistema, sino que se enfocan en el usuario final y en cómo este interactuará con el sistema. Por lo tanto, las historias de usuario son una herramienta de comunicación entre el equipo de desarrollo y el cliente.

En **Scrum** las historias de usuario son una parte fundamental del proceso de desarrollo de software. En este marco de trabajo, las historias de usuario son utilizadas para definir los requisitos del sistema y son la base para la planificación y estimación de las tareas a realizar.

¿Por qué son importantes las historias de usuario?

- Centran la atención en el usuario final.
- Permiten la colaboración y comunicación entre el equipo de desarrollo y el cliente.
- Fomentan soluciones creativas y flexibles.

4.4.1. Estructura de una historia de usuario

Las historias de usuario siguen una estructura general simple y clara.

Como [tipo de usuario], **quiero** [realizar una acción], **para** [obtener un beneficio].

- **Como**: describe el tipo de usuario que está interactuando con el sistema.
- **Quiero**: describe la acción que el usuario desea realizar.
- **Para**: describe el beneficio que el usuario obtendrá al realizar la acción.

Además de esta estructura general, las historias de usuario pueden incluir otros elementos como criterios de aceptación, prioridad, estimación de esfuerzo, entre otros.

Para se ha definido la siguiente estructura para las historias de usuario:

ID	Identificador único de la historia de usuario.	Nombre	Nombre de la historia de usuario.
Descripción		Descripción general de la historia de usuario.	
Estimación		Estimación del esfuerzo necesario para completar la historia de usuario. Basado en Planning Poker.	
Prioridad		Acción que el usuario desea realizar. Desde P3 (baja) hasta P0 (alta).	
Criterios de aceptación		Conjunto de condiciones que deben cumplirse para considerar la historia de usuario como completada.	

Tabla 4.1: Estructura de una historia de usuario

4.4.2. Historias de usuario

ID	HU-1	Nombre	Iniciar sesión
Descripción		Como usuario he de poder iniciar sesión en el sistema.	
Estimación		3	
Prioridad		P0	
Criterios de aceptación		<ul style="list-style-type: none"> ■ Para poder iniciar sesión ha de insertar su correo y contraseña. ■ Sólo se puede iniciar sesión con correos de la UGR. 	

Tabla 4.2: Historia de usuario HU-1

ID	HU-2	Nombre	Registrarse
Descripción		Como usuario he de poder registrarme en el sistema.	
Estimación		5	
Prioridad		P0	
Criterios de aceptación		<ul style="list-style-type: none"> ■ El alumno sólo se puede registrar con su correo institucional de la UGR. ■ El alumno debe insertar nickname, correo y contraseña. ■ La contraseña del alumno ha de ser mayor o igual a 9 caracteres, conteniendo esta una mayúscula y un número como mínimo. ■ El registro se ha de completar mediante un link mandado por mail. 	

Tabla 4.3: Historia de usuario HU-2

ID	HU-3	Nombre	Modificar nickname
Descripción		Como usuario puedo modificar mi nickname.	
Estimación		2	
Prioridad		P2	
Criterios de aceptación		<ul style="list-style-type: none"> ■ El alumno no puede cambiar su nickname a otro que exista. ■ El alumno no puede modificar su correo electrónico. 	

Tabla 4.4: Historia de usuario HU-3

ID	HU-4	Nombre	Modificar contraseña
Descripción		Como usuario he de poder modificar la contraseña de acceso.	
Estimación		3	
Prioridad		P1	
Criterios de aceptación		<ul style="list-style-type: none"> ■ Para poder modificar la contraseña ha de insertar la contraseña anterior. ■ Se ha de insertar la nueva contraseña 2 veces, siendo esta mayor o igual a 9 caracteres, y conteniendo una mayúscula y un número como mínimo. 	

Tabla 4.5: Historia de usuario HU-4

ID	HU-5	Nombre	Darse de baja
Descripción		Como usuario he de poder darme de baja del sistema.	
Estimación		1	
Prioridad		P2	
Criterios de aceptación		<ul style="list-style-type: none"> ■ Para poder completar la baja ha de escribir su contraseña en un campo de texto. 	

Tabla 4.6: Historia de usuario HU-5

ID	HU-6	Nombre	Cambiar rol
Descripción	Como administrador he de poder actualizar mi rol a profesor, y viceversa.		
Estimación	2		
Prioridad	P1		
Criterios de aceptación	<ul style="list-style-type: none"> ■ Para poder cambiar el rol a profesor he de ser administrador. ■ Para poder cambiar el rol a profesor he de ser administrador. 		

Tabla 4.7: Historia de usuario HU-6

ID	HU-7	Nombre	Seleccionar grados
Descripción	Como usuario he de poder seleccionar el grado o grados que estoy cursando.		
Estimación	3		
Prioridad	P0		
Criterios de aceptación	<ul style="list-style-type: none"> ■ Se pueden seleccionar un máximo de 4 grados. 		

Tabla 4.8: Historia de usuario HU-7

ID	HU-8	Nombre	Eliminar grado
Descripción	Como usuario he de poder eliminar un grado que ya no esté cursando.		
Estimación	2		
Prioridad	P1		
Criterios de aceptación	<ul style="list-style-type: none"> ■ Si el usuario está suscrito a grupos de asignatura de ese grado, se le recordará que también se revocarán sus suscripciones a estos. 		

Tabla 4.9: Historia de usuario HU-8

ID	HU-9	Nombre	Suscribirse a grupos de asignatura
Descripción		Como usuario he de poder suscribirme a los grupos de asignaturas a las que quiero hacer seguimiento.	
Estimación		4	
Prioridad		P0	
Criterios de aceptación		<ul style="list-style-type: none"> ■ Para hacer seguimiento a un grupo en concreto, el usuario deberá estar cursando el grado al que pertenece. 	

Tabla 4.10: Historia de usuario HU-9

ID	HU-10	Nombre	Revocar suscripción a grupo de asignatura
Descripción		Como usuario he de poder revocar una suscripción a un grupo de asignatura.	
Estimación		2	
Prioridad		P1	
Criterios de aceptación		<ul style="list-style-type: none"> ■ El usuario sólo puede revocar suscripciones de grupos a los que está suscrito. 	

Tabla 4.11: Historia de usuario HU-10

ID	HU-11	Nombre	Ver horario de grupos de asignatura
Descripción		Como usuario he de poder obtener la información de mi horario personalizado conforme a las suscripciones.	
Estimación		5	
Prioridad		P0	
Criterios de aceptación		<ul style="list-style-type: none"> ■ El horario de cada clase debe mostrar la asignatura, grupo, hora de inicio y fin, profesores del grupo, y aula. ■ Las clases sólo deben mostrarse en el rango de fechas en las que se imparten. 	

Tabla 4.12: Historia de usuario HU-11

ID	HU-12	Nombre	Crear evento puntual a nivel de grupo de asignatura
Descripción		Como profesor / administrador he de poder crear eventos puntuales a nivel de grupo (clases de recuperación, extra, charlas...).	
Estimación		3	
Prioridad		P0	
Criterios de aceptación		<ul style="list-style-type: none"> ■ Se debe especificar la fecha, hora de inicio, hora de fin y tipo de evento. ■ La clase extra no debe coincidir con otra clase existente en horario y aula. ■ El usuario debe ser un profesor o administrador. 	

Tabla 4.13: Historia de usuario HU-12

ID	HU-13	Nombre	Eliminar evento puntual a nivel de grupo de asignatura
Descripción		Como profesor / administrador he de poder eliminar los eventos que he creado (clases de recuperación, extra, charlas ...).	
Estimación		2	
Prioridad		P1	
Criterios de aceptación		<ul style="list-style-type: none"> ■ Se debe especificar la fecha, hora de inicio, hora de fin y tipo de evento. ■ La clase extra no debe coincidir con otra clase existente en horario y aula. ■ El usuario debe ser un profesor o administrador. 	

Tabla 4.14: Historia de usuario HU-13

ID	HU-14	Nombre	Exportar horario a calendario estándar
Descripción		Como usuario he de poder exportar mi horario para usarlo en otros sistemas estándar de calendario.	
Estimación		4	
Prioridad		P0	
Criterios de aceptación		<ul style="list-style-type: none"> ■ El usuario podrá exportar su horario en formatos compatibles con sistemas estándar de calendario (.ics). ■ La exportación debe incluir todas las asignaturas y eventos del usuario. ■ Se debe permitir elegir un rango de fechas para la exportación. ■ El archivo generado debe poder descargarse y ser importable en Google Calendar, Outlook, Apple Calendar, etc. 	

Tabla 4.15: Historia de usuario HU-14

ID	HU-15	Nombre	Sincronizar calendario con Google Calendar
Descripción		Como usuario he de poder sincronizar mi calendario con Google Calendar.	
Estimación		5	
Prioridad		P3	
Criterios de aceptación		<ul style="list-style-type: none"> ■ El usuario podrá vincular su cuenta con Google Calendar mediante OAuth. ■ Los eventos de su horario deben sincronizarse automáticamente con Google Calendar. ■ Se deben reflejar en Google Calendar los cambios realizados en el horario del usuario. ■ El usuario debe poder desactivar la sincronización en cualquier momento. 	

Tabla 4.16: Historia de usuario HU-15

ID	HU-16	Nombre	Ver alertas de clases extra de grupos de asignatura
Descripción		Como alumno he de poder recibir alertas referentes a "clases extra" de mis grupos (clases de recuperación, extra, charlas...).	
Estimación		3	
Prioridad		P1	
Criterios de aceptación		<ul style="list-style-type: none"> ■ Las clases extra aparecerán, como las clases, en la vista del horario. ■ Si el evento es en la semana actual se debe mostrar en una lista de alertas. ■ Se debe mandar un correo electrónico a los alumnos que afecte y tengan las notificaciones activadas. 	

Tabla 4.17: Historia de usuario HU-16

ID	HU-17	Nombre	Crear evento a nivel de facultad
Descripción		Como administrador he de poder crear eventos a nivel de facultad (charlas, conferencias, exámenes...).	
Estimación		5	
Prioridad		P1	
Criterios de aceptación		<ul style="list-style-type: none"> ■ Se debe especificar la fecha, hora de inicio, hora de fin y tipo de evento. ■ El evento no debe coincidir con otra clase existente en horario y aula. ■ El usuario debe ser un administrador. 	

Tabla 4.18: Historia de usuario HU-17

ID	HU-18	Nombre	Eliminar evento a nivel de facultad
Descripción		Como profesor / administrador he de poder eliminar los eventos que he creado a nivel de facultad (charlas, conferencias, exámenes ...).	
Estimación		3	
Prioridad		P2	
Criterios de aceptación		<ul style="list-style-type: none"> ■ El usuario debe ser un administrador. 	

Tabla 4.19: Historia de usuario HU-18

ID	HU-19	Nombre	Activar / desactivar alertas por correo electrónico
Descripción		Como alumno debo poder desactivar / activar las notificaciones por correo electrónico referente a los eventos a nivel de grupo / facultad	
Estimación		3	
Prioridad		P2	
Criterios de aceptación		<ul style="list-style-type: none"> ■ Tras desactivar las alertas, el usuario no recibirá más correos electrónicos referente a eventos a nivel de grupo / facultad. ■ El usuario podrá activar las alertas en cualquier momento. 	

Tabla 4.20: Historia de usuario HU-19

ID	HU-20	Nombre	Suscripción a todas las asignaturas que imparte un profesor
Descripción		Como profesor he de poder buscar mi nombre, y aceptar suscribirme a las asignaturas a las que imparto clase en la UGR	
Estimación		5	
Prioridad		P2	
Criterios de aceptación		<ul style="list-style-type: none"> ■ El nombre a buscar debe ser un profesor reflejado en la web Grados UGR. ■ El usuario se suscribirá a todas las asignaturas que imparte el profesor. 	

Tabla 4.21: Historia de usuario HU-20

4.5. Requisitos funcionales

A partir de las historias de usuario, junto a sus criterios de aceptación, se han extraído los siguientes requisitos funcionales:

4.5.1. Gestión de usuarios

- **RF-1) Gestión de usuarios:** El sistema debe poder registrar usuarios para futuros inicio de sesión y seguimiento de su información de suscripciones a grupos de asignaturas.
 - **RF-1.1) Inicio de sesión:** El sistema debe permitir el inicio de sesión de usuarios mediante correo electrónico institucional y contraseña.
 - **RF-2.2) Registro de usuarios:** El sistema debe tener un proceso de registro de usuario.
 - **RF-2.3) Completar el registro:** Para completar el registro el sistema debe mandar un mail para confirmar si el usuario se trata de un alumno o de un profesor.
 - **RF-2.4) Cambio de nickname:** El sistema debe permitir cambiar el nickname al usuario por uno no usado.
 - **RF-2.5) Cambio de contraseña de acceso:** El sistema debe permitir el cambio de la contraseña de acceso.
 - **RF-2.6) Dar de baja:** El sistema debe permitir al usuario darse de baja con el objetivo de que no le lleguen más correos relacionados.

- **RF-2.7) Cambio de rol:** EL sistema debe poder facilitar el cambio de rol de profesor a administrador, y viceversa.
- **RF-2.8) Activación / desactivación de alertas:** El sistema debe permitir activar o desactivar las alertas por correo electrónico.
- **RF-2.9) Suscripción a los grupos que imparte un profesor:** El sistema debe permitir al profesor suscribirse a los grupos de asignaturas que imparte.

4.5.2. Gestión de horarios académicos

- **RF-2) Gestión de horarios académicos:** El sistema debe poder obtener la información relacionada con el horario académico de todos los grados de la UGR, para así poder identificar los horarios personalizados de alumnos y docentes a través de un sistema de suscripción a grupos de asignatura.
 - **RF-2.1) Recopilación de horarios:** El sistema debe recopilar la información de horarios académicos de todos los grados de la UGR.
 - **RF-2.2) Grados del alumno / profesor:** El sistema debe recoger el grado/ grados académicos que está cursando / impartiendo el alumno / profesor.
 - **RF-2.3) Asignaturas del alumno / profesor:** El sistema debe recoger las asignaturas que está cursando / impartiendo el alumno / profesor.
 - **RF-2.4) Grupos del alumno / profesor:** El sistema debe recoger los grupos de las asignaturas que está cursando / impartiendo el alumno / profesor.
 - **RF-2.5) Eliminar grados del alumno / profesor:** El sistema debe poder eliminar el grado/ grados académicos que está cursando / impartiendo el alumno / profesor.
 - **RF-2.6) Eliminar asignaturas del alumno / profesor:** El sistema debe poder eliminar las asignaturas que está cursando / impartiendo el alumno / profesor.
 - **RF-2.7) Eliminar grupos del alumno / profesor:** El sistema debe poder eliminar los grupos de las asignaturas que está cursando / impartiendo el alumno / profesor.
 - **RF-2.8) Horario personalizado:** El usuario ha de poder acceder a la información de horario académico de los grupos de asignaturas a los que esté suscrito.
 - **RF-2.9) Crear clases extra:** El sistema debe permitir al profesor / administrador crear clases extra a las oficiales.
 - **RF-2.10) Eliminar clases extra:** El sistema debe permitir al profesor / administrador eliminar clases extra a las oficiales.

- **RF-2.11) Exportar horario a estándar:** El sistema debe poder exportar el horario en formato estándar (.ics).
- **RF-2.12) Sincronizar con Google calendar:** El sistema deberá poder sincronizarse con Google Calendar.
- **RF-2.13) Alertas sobre clases extra:** El sistema debe poder mandar alertas sobre clases extra a los alumnos de ese grupo.
- **RF-2.14) Crear eventos a nivel de facultad:** El sistema debe poder crear eventos a nivel de facultad.
- **RF-2.15) Alertas de cambios en asignaturas suscritas:** El sistema debe poder mandar alertas de cambios en las asignaturas suscritas.
- **RF-2.16) Eliminar eventos a nivel de facultad:** El sistema debe poder eliminar eventos a nivel de facultad.

4.6. Requisitos no funcionales

4.6.1. Rendimiento

- **RNF-1.1) Tiempo de respuesta:** El sistema debe responder a las solicitudes de los usuarios en un tiempo máximo de 2 segundos.
- **RNF-1.2) Capacidad de usuarios concurrentes:** El sistema debe soportar un mínimo de 1000 usuarios concurrentes sin degradación significativa del rendimiento.
- **RNF-1.3) Recuperación ante fallos:** El sistema debe ser capaz de recuperarse de fallos en menos de 5 minutos.

4.6.2. Usabilidad

- **RNF-2.1) Interfaz intuitiva:** La interfaz de usuario debe ser fácil de usar y comprender, incluso para usuarios sin experiencia técnica.
- **RNF-2.2) Accesibilidad:** El sistema debe cumplir con las pautas de accesibilidad web (WCAG) para garantizar que sea utilizable por personas con discapacidades.
- **RNF-2.3) Compatibilidad con dispositivos:** El sistema debe ser compatible con cualquier navegador web, y a cualquier resolución.

4.6.3. Seguridad

- **RNF-3.1) Autenticación segura:** El sistema debe implementar un mecanismo de autenticación y autorización basado en JWT.

- **RNF-3.2) Protección de datos:** El sistema debe proteger los datos de usuario confidenciales (como contraseñas y correos electrónicos) mediante cifrado y otras medidas de seguridad.
- **RNF-3.3) Autorización:** El sistema debe controlar el acceso a las funciones del sistema según los roles de usuario (administrador, profesor, alumno).

4.6.4. Mantenibilidad

- **RNF-4.1) Modularidad:** El sistema debe estar diseñado de forma modular para facilitar el mantenimiento y la actualización.
- **RNF-4.2) Documentación:** El sistema debe estar debidamente documentado para facilitar la comprensión y el mantenimiento del código.
- **RNF-4.3) Pruebas:** El sistema debe incluir pruebas unitarias y de integración para garantizar la calidad del código.
- **RNF-4.4) Descubrimiento:** El sistema ha de tener un servicio de descubrimiento de servicios para facilitar la extensión del sistema.
- **RNF-4.5) Configuración:** El sistema ha de contar con un servidor de configuración para centralizarla.

4.6.5. Portabilidad

- **RNF-5.1) Independencia de plataforma:** El sistema debe ser independiente de la plataforma, lo que significa que debe poder ejecutarse en diferentes sistemas operativos y entornos de servidor.

4.6.6. Disponibilidad

- **RNF-6.1) Tiempo de actividad:** El sistema debe tener un tiempo de actividad del 95 %.
- **RNF-6.2) Recuperación ante fallos:** El sistema debe poder recuperarse de fallos de hardware o software sin pérdida de datos.

4.7. Requisitos de información

El sistema debe recopilar y almacenar la siguiente información de los diferentes servicios:

4.7.1. Servicio de usuarios

No se recopila demasiada información del usuario, sólo la necesaria para poder identificarlo y autenticarlo. La información que se recopila es la siguiente:

- **Nickname:** Nombre de usuario único.
- **Correo electrónico:** Correo electrónico institucional (necesario para clasificar al usuario como alumno o profesor).
- **Contraseña:** Contraseña de acceso al sistema.

Esta corresponde con la información de entrada que ofrecen los usuarios al registrarse. El sistema no almacena la contraseña, sino un hash de la misma, para evitar que un posible ataque a la base de datos comprometa la seguridad de los usuarios. Tras el procesamiento de la información, el sistema almacena la siguiente información:

- **ID:** Identificador único del usuario.
- **Rol:** Rol del usuario.
- **Notificaciones :** Información sobre si el usuario tiene activadas o desactivadas las notificaciones.

4.7.2. Servicio de horarios

El sistema recopila la información de horarios académicos de todos los grados de la UGR. Esta información es pública y se obtiene de la página "grados.ugr.es". La información que se recopila es la siguiente:

- **Grado:** Información referente al grado.
 - **Facultad :** Nombre de la facultad a la que pertenece el grado.
 - **Campo :** Campo de conocimiento al que pertenece el grado.
 - **Nombre :** Nombre del grado.
 - **Url :** Url de la página del grado.
- **Asignatura:** Información referente a la asignatura.
 - **Curso académico :** Curso académico al que pertenece la asignatura.
 - **Departamento :** Departamento al que pertenece la asignatura.
 - **Nombre :** Nombre de la asignatura.
 - **Semestre :** Semestre al que pertenece la asignatura.
 - **Tipo :** Tipo de asignatura (obligatoria, optativa, etc.).
 - **Url :** Url de la página de la asignatura.
 - **Año :** Año en el que se imparte la asignatura.

- **Grupo** : Información referente al grupo.
 - **Nombre** : Nombre del grupo.
 - **Profesores** : Profesores que imparten la asignatura.
- **Clase** : Información referente a la clase.
 - **Aula** : Aula en la que se imparte la clase.
 - **Fecha de inicio** : Fecha de inicio de la clase.
 - **Fecha de fin** : Fecha de fin de la clase.
 - **Día** : Día de la semana en el que se imparte la clase.
 - **Hora de inicio** : Hora de inicio de la clase.
 - **Hora de fin** : Hora de fin de la clase.

4.7.3. Servicio de suscripciones académicas

El sistema recopila la información de las suscripciones académicas de los usuarios y de los eventos (clases extra, charlas, días festivos ...). Esta información es privada y se obtiene de la base de datos del sistema. La información que se recopila es la siguiente:

- **Suscripción**: Información referente a la suscripción.
 - **ID** : Identificador único de la suscripción.
 - **ID del usuario** : Identificador único del usuario.
 - **Nombre del grupo** : Nombre del grupo al que está suscrito el usuario.
 - **Nombre de la asignatura** : Nombre de la asignatura a la que está suscrito el usuario.
 - **Nombre del grado** : Nombre del grado al que está suscrito el usuario.
- **Evento**: Información referente al evento.
 - **ID** : Identificador único del evento.
 - **ID del usuario creador** : Identificador único del usuario que ha creado el evento.
 - **Facultad** : Facultad a la que pertenece el evento.
 - **Grado** : Grado al que pertenece el evento.
 - **Asignatura** : Asignatura al que pertenece el evento.
 - **Grupo** : Grupo al que pertenece el evento.
 - **Tipo** : Tipo de evento (a nivel de grupo / facultad).
 - **Fecha** : Fecha del evento.

- **Hora de inicio** : Hora de inicio del evento.
- **Hora de fin** : Hora de fin del evento.
- **Día** : Día de la semana en el que se imparte el evento.
- **Aula** : Aula en la que se imparte el evento.
- **Título** : Título del evento.
- **Profesor** : Profesor que imparte el evento.

4.8. Validación de los requisitos

La validación de los requisitos ha sido un proceso clave para garantizar que el sistema desarrollado cumpla con las expectativas y necesidades definidas. Este proceso se ha llevado a cabo mediante reuniones semanales con el Product Manager, quien en este caso ha sido el director del TFG, D. Juan Luis Jiménez Laredo.

Durante estas reuniones, se han abordado los siguientes aspectos:

- **Revisión de requisitos:** Se han revisado los requisitos funcionales y no funcionales definidos, asegurando que sean claros, completos y alineados con los objetivos del proyecto.
- **Validación de historias de usuario:** Se han analizado las historias de usuario propuestas, verificando que reflejen correctamente las necesidades de los usuarios finales y que incluyan criterios de aceptación adecuados.
- **Validación de implementaciones:** Se han presentado los incrementos de software desarrollados durante cada sprint, evaluando si cumplen con los requisitos y las historias de usuario previamente validadas.
- **Retroalimentación:** Se ha recibido retroalimentación por parte del Product Manager, lo que ha permitido realizar ajustes y mejoras tanto en los requisitos como en las implementaciones.

Este enfoque iterativo e incremental ha asegurado que el desarrollo del sistema se mantenga alineado con las expectativas del proyecto, minimizando riesgos y garantizando la calidad del producto final.

5. Planificación, metodología y presupuesto del proyecto

En este apartado se presenta la planificación del proyecto, incluyendo el cronograma de trabajo, la metodología de desarrollo utilizada y la gestión de riesgos.

5.1. Cronograma del proyecto

Antes del comienzo del desarrollo del proyecto, se realizó una planificación inicial que incluía la definición de los sprints y las tareas a realizar en cada uno de ellos de manera general, definiendo hitos, no tareas específicas. Esta planificación se ha seguido a lo largo del desarrollo, aunque ha habido ajustes en función de los avances y los resultados obtenidos.

La realización del cronograma se ha llevado a cabo haciendo uso de la herramienta **GantPRO**[59], que permite la creación de diagramas de Gantt de manera sencilla y efectiva. A continuación, se presenta el diagrama de Gantt del proyecto en la figura 5.1, que muestra las diferentes fases y tareas a realizar en cada sprint.

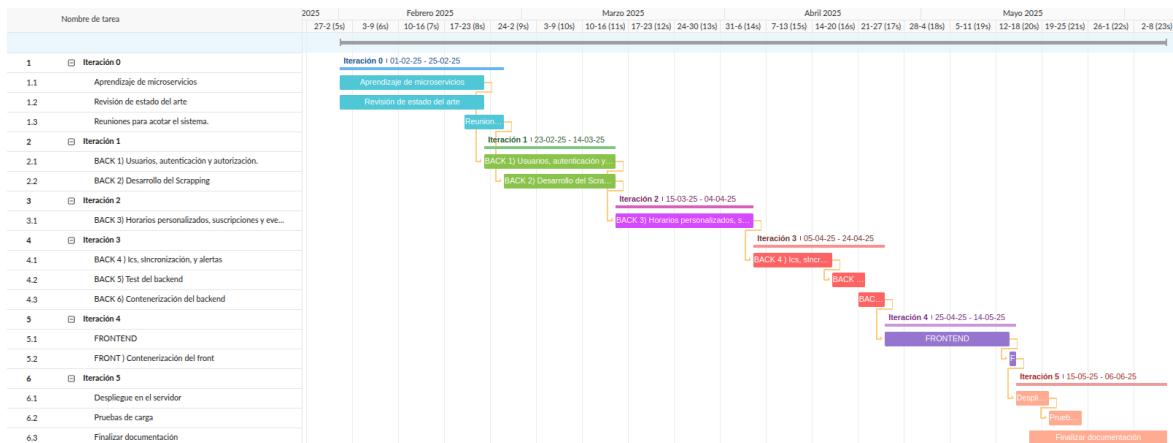


Figura 5.1: Gantt del proyecto.

El cronograma del proyecto se ha dividido en 5 sprints, cada uno con una duración de 3 semanas. Como se muestra en la figura 5.1, cada sprint ha tenido un conjunto de tareas generales a realizar, que se han ido completando a lo largo del desarrollo.

1. **Sprint 0:** En este sprint se paraleliza por un lado el aprendizaje técnico acerca de microservicios, docker y el framework de desarrollo backend Spring Boot, a la vez que se acota el sistema y se recaban los requisitos iniciales del sistema.

2. **Sprint 1:** En este sprint se comienza el desarrollo del backend implementando los servicios relativos a los usuarios, y la autenticación y autorización basadas en las credenciales de la UGR junto al servicio de mensajería (notificaciones). Además se implementa el scrapping de la web de "Grados UGR" para obtener los horarios de los grados.
3. **Sprint 2:** En este sprint se continúa el backend implementando las funcionalidades relativas a suscripciones, horarios personalizados y creación de eventos.
4. **Sprint 3:** En este sprint se desarrolla la parte del backend relacionada con generación de archivos ics, sincronización con sistemas de calendarios externos y alertas. Además se realizan tests y la contenerización del sistema.
5. **Sprint 4:** En este sprint se desarrolla el frontend del sistema, implementando la interfaz de usuario y la comunicación con el backend.
6. **Sprint 5:** En este último sprint se realiza el despliegue en el servidor de la UGR, se realizan pruebas de carga y se finaliza el proyecto para su entrega. Además se añade la funcionalidad extra de búsqueda de suscripciones por nombre de profesor, de modo que podemos suscribirnos directamente a los grupos de asignaturas que imparte este.

En todos los sprints se realizarán además tareas de documentación y pruebas, además del seguimiento y registro de horas dedicadas a cada tarea.

5.2. Metodología de desarrollo

Para la gestión y desarrollo del proyecto, se ha optado por la metodología ágil **Scrum**. Esta metodología se caracteriza por su enfoque iterativo e incremental, permitiendo una adaptación flexible a los cambios y una entrega temprana de valor.

5.2.1. Roles y Responsabilidades en este Proyecto

Dada la naturaleza individual de este proyecto, los roles tradicionales de Scrum se han adaptado de la siguiente manera:

- **Equipo de Desarrollo y Scrum Master:** El autor de este TFG ha asumido ambos roles. Esto implica la responsabilidad de llevar a cabo el desarrollo del software, así como de facilitar el proceso Scrum, asegurando que se sigan las prácticas y principios de la metodología. Se ha encargado de la planificación, ejecución y revisión de cada sprint, así como de la identificación y resolución de impedimentos.
- **Product Owner:** El rol de Product Owner ha sido desempeñado tanto por el director del TFG, D. Juan Luis Jiménez Laredo, como por el autor del sistema. En esta función, ambos han sido los responsables de definir la visión del

producto, priorizar el Backlog del Producto y asegurar que el desarrollo se alinee con las necesidades y expectativas del proyecto. Los dos participaron activamente en la definición de los requisitos y en la validación de los incrementos de software.

5.2.2. Proceso Scrum Implementado

El proceso Scrum se ha implementado siguiendo los siguientes pasos clave:

- **Backlog del Producto:** Se ha definido un Backlog del Producto inicial, compuesto por las funcionalidades y tareas necesarias para completar el TFG.
- **Sprints:** El desarrollo se ha dividido en 5 sprints de duración 3 semanas cada uno. Cada sprint ha tenido como objetivo la entrega de un incremento de software funcional y potencialmente entregable.
- **Planificación del Sprint:** Al inicio de cada sprint, se ha llevado a cabo una reunión de planificación en la que, junto con el Product Owner, se han seleccionado los elementos del Backlog del Producto que se abordarían durante el sprint. Se han estimado las tareas y se ha definido el Sprint Backlog.
- **Desarrollo del Sprint:** Durante el sprint, el autor ha trabajado en el desarrollo de las tareas asignadas, siguiendo las prácticas de desarrollo y asegurando la calidad del código.
- **Reunión Diaria (Daily Scrum):** Aunque adaptada a la naturaleza individual del proyecto, se ha realizado una reflexión diaria sobre el progreso, los impedimentos y las tareas a realizar. Esto ha permitido mantener un seguimiento constante del avance.
- **Revisión del Sprint (Sprint Review):** Al finalizar cada sprint, se ha llevado a cabo una revisión del sprint. Dado que el autor es también el equipo de desarrollo, esta revisión ha consistido en una **introspección personal y un análisis de los resultados del sprint**, evaluando las metas alcanzadas y el incremento de software desarrollado. Se ha realizado una autoevaluación del progreso y la calidad del trabajo.
- **Retrospectiva del Sprint (Sprint Retrospective):** La retrospectiva del sprint se ha realizado en colaboración con el Product Owner (D. Juan Luis Jiménez Laredo). En esta reunión, se ha analizado el sprint finalizado, identificando qué se ha hecho bien, qué se podría mejorar y qué acciones concretas se podrían implementar para el siguiente sprint. Esta colaboración ha permitido obtener una perspectiva externa y valiosa para la mejora continua del proceso.

5.2.3. Justificación de la Metodología

La elección de la metodología Scrum se justifica por las siguientes razones:

- **Flexibilidad:** Permite adaptarse a los cambios en los requisitos y a los aprendizajes obtenidos durante el desarrollo. En concreto este sistema dependía en etapas tempranas de desarrollo del posible acceso a datos oficiales de la UGR, sistemas de autenticación internos, datos de matriculaciones, etc. Es por ello que la flexibilidad de Scrum ha sido clave para ajustar el plan a medida que se han ido conociendo más detalles.
- **Entrega Temprana de Valor:** Facilita la entrega de incrementos funcionales de software de forma regular, lo que permite obtener retroalimentación temprana y ajustar el rumbo del proyecto si es necesario.
- **Transparencia:** El uso de herramientas como GitHub Projects y la realización de las reuniones Scrum promueven la transparencia en el progreso del proyecto.
- **Adaptabilidad a un Proyecto Individual:** Aunque tradicionalmente Scrum se aplica a equipos, su estructura iterativa y adaptable se ajusta bien a un proyecto individual como un TFG, permitiendo una organización eficiente del trabajo y una gestión del tiempo efectiva.

Es importante destacar que, dada la naturaleza individual del proyecto, se ha realizado una adaptación de los roles y las ceremonias de Scrum para ajustarse a las necesidades y recursos disponibles. Sin embargo, se han mantenido los principios fundamentales de la metodología para asegurar una gestión eficaz del desarrollo.

5.2.4. Gestión de Tareas y Seguimiento del Progreso

Para la gestión de las tareas y el seguimiento del progreso del proyecto, se ha utilizado **GitHub Projects**^[60]. Esta herramienta ha permitido:

- **Creación de un Backlog del Producto:** Se ha creado un backlog del producto en GitHub Projects, donde se han definido las historias de usuario y las tareas necesarias para el desarrollo del sistema. Este backlog ha sido la base para la planificación de los sprints y la gestión de las tareas.

Cada tarea creada en este ha representado una historia de usuario o una tarea aparte a realizar (reuniones, investigación, etc.). Cada tarea ha sido asignada a un sprint y se ha estimado el tiempo necesario para su realización usando la técnica de **“Planning Poker”**. Esta técnica ha permitido una estimación más precisa y consensuada entre el Product Owner y el equipo de desarrollo. Además cada historia de usuario llevaba una serie de criterios de aceptación que se han ido marcando a medida que se iban cumpliendo. En la figura 5.2 se muestra un ejemplo de una historia de usuario creada en GitHub Projects, donde se puede ver la descripción, los criterios de aceptación y el estado de la tarea.



Figura 5.2: Ejemplo de historia de usuario en Github Projects.

- **Creación de Tableros por Sprint:** Se han configurado tableros de proyecto en GitHub Projects, utilizando las funcionalidades de “Iteraciones” para representar cada sprint. Esto ha facilitado la visualización del trabajo en curso para cada sprint, permitiendo un seguimiento claro del progreso y el estado de las tareas.

Antes del comienzo de cada sprint se revisa el product backlog, se seleccionan las tareas a realizar y se crea el tablero correspondiente poniendo todas las tareas en estado “To do”. Además también se revisan las prioridades de estas y se cambian si el proyecto lo requiere. Durante el desarrollo del sprint, las tareas se van moviendo a los diferentes estados según su avance (“Backlog”, “Todo”, “In progress”, “Testing”, “Done”). Se refleja un ejemplo de Sprint en la figura 5.3.

Figura 5.3: Tablero del 2º Sprint durante su desarrollo.

- **Visualización de las tareas en el tiempo:** La herramienta ha permitido visualizar el progreso de las tareas en el tiempo a través de un roadmap, lo

que ha facilitado la identificación de posibles retrasos y la toma de decisiones para ajustar el plan si es necesario.

Además este cronograma se ha ido actualizando a medida que se iban completando las tareas, permitiendo una visualización clara del progreso del proyecto, como se muestra en la figura 5.4, y una escenificación de las tareas realizadas en el tiempo.

Enlace al roadmap del proyecto [61].

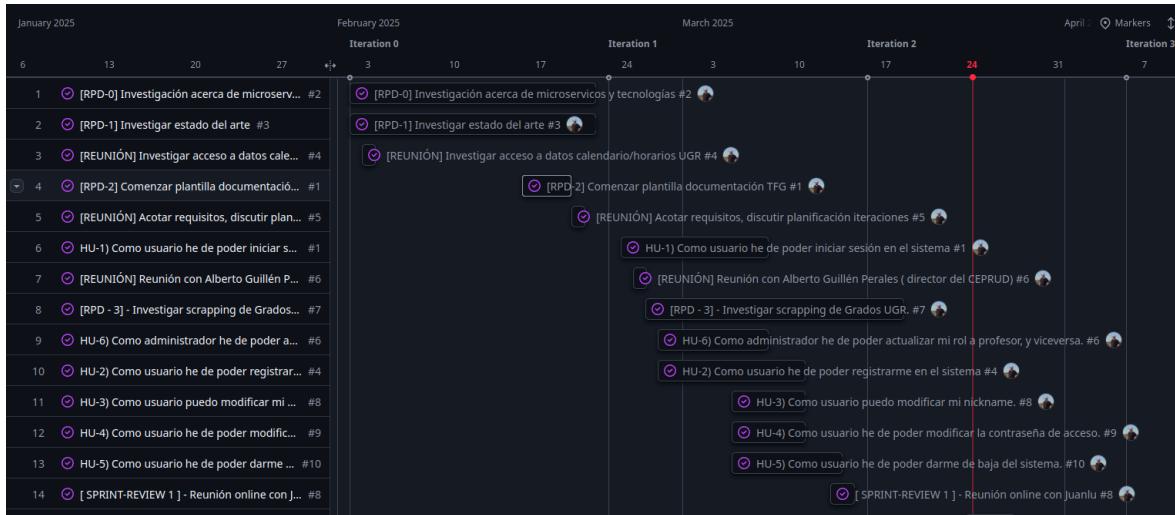


Figura 5.4: Segmento del "Roadmap" del proyecto.

Para la gestión del tiempo dedicado a cada tarea, se ha utilizado la funcionalidad de “**Time Tracking**” Clockify. Esta funcionalidad permite registrar el tiempo dedicado a cada tarea y generar informes sobre el progreso del proyecto. Además, se ha utilizado la técnica de “**Pomodoro**” para gestionar el tiempo de trabajo, lo que ha permitido mantener un enfoque constante y evitar la fatiga.

Clockify[62] es una herramienta de seguimiento del tiempo que permite registrar el tiempo dedicado a cada tarea y generar informes sobre el progreso del proyecto, como se muestra en la figura 5.5. Esta herramienta ha sido utilizada para llevar un control detallado del tiempo invertido en cada tarea, lo que ha facilitado la gestión del tiempo y la identificación de posibles retrasos. Además nos facilita un total de horas dedicadas al desarrollo del proyecto, por lo que facilita demostrar el esfuerzo realizado en el mismo.

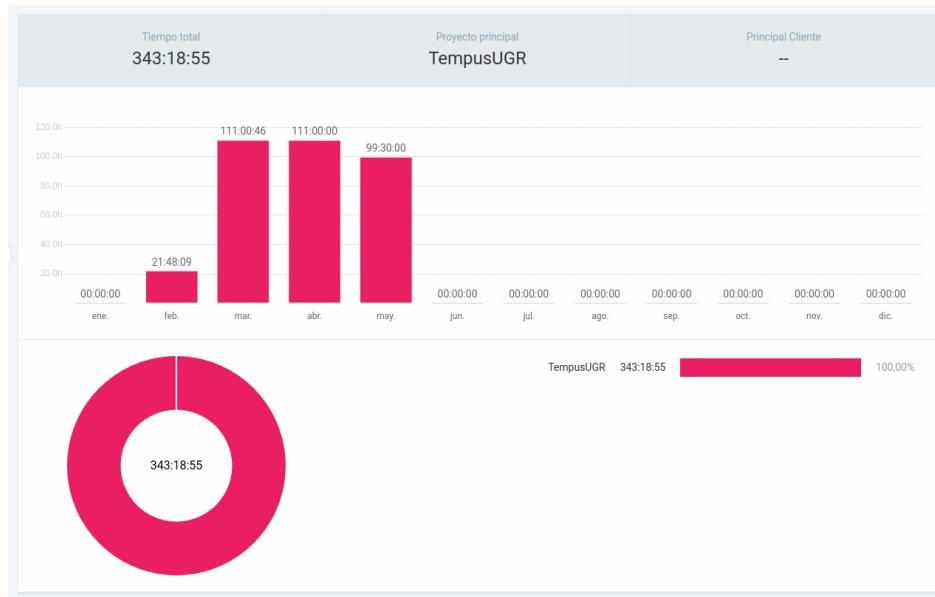


Figura 5.5: Resumen de horas de desarrollo en Clockify.

Este resumen de horas son las correspondientes a los sprints del 1 al 5, e incluye las horas dedicadas a tareas de desarrollo, investigación, reuniones y documentación. En total, y sumando 28.5 horas del curso de microservicios, 5 horas de investigación inicial, y otras 5 horas de reuniones el Sprint 0, se han dedicado un total de 376.5 horas al desarrollo del proyecto, superando así las 300 horas mínimas para superar el TFG.

5.3. Desarrollo y control de versiones

El desarrollo del proyecto se ha realizado utilizando el sistema de control de versiones **Git** y la plataforma de alojamiento de código **GitHub**. Esta combinación ha permitido llevar un control detallado de los cambios realizados en el código, facilitar la colaboración y mantener una trazabilidad completa del desarrollo.

Cada servicio independiente del backend ha supuesto un repositorio independiente, puesto que son programas independientes que se comunican a través de la red. Además se ha creado un repositorio para el frontend y otro para la documentación en latex.

En este último repositorio se ha hecho uso de “**Github Actions**” para automatizar el proceso de compilación y despliegue de la documentación, lo que ha permitido mantenerla actualizada de manera continua. Cada vez que se realiza un cambio en la documentación, se ejecuta un flujo de trabajo que compila el código LaTeX y genera el PDF final, asegurando que siempre se disponga de la versión más reciente.

Además cuando el director de este proyecto ha reflejado aluna corrección mediante comentarios en latex, se ha hecho uso de la funcionalidad de “**Pull Requests**” de

GitHub para revisar y aplicar estos cambios de manera controlada. Esto ha permitido mantener un flujo de trabajo ordenado y garantizar que todas las modificaciones se integren correctamente en la documentación final.

5.4. Gestión de riesgos

En todo proyecto de desarrollo de software, es fundamental identificar y gestionar los riesgos que pueden afectar al éxito del mismo. A continuación se presentan los principales riesgos identificados en este proyecto, junto con las estrategias de mitigación implementadas:

- **Riesgo de cambios en los requisitos:** Dado que desde el principio del proyecto se trabajó con incertidumbre respecto a la información de la que se podía disponer (información de los horarios académicos, matriculaciones del alumnado, autenticación institucional ...) y la posibilidad de acceso a estos datos, se ha optado por una metodología ágil (Scrum) que permite adaptarse a los cambios en los requisitos de manera flexible. Además, se ha mantenido una comunicación constante con el Product Owner para ajustar el backlog del producto según sea necesario.

Además se ha ido ajustando y equilibrando las tareas a realizar entre los sprints, de manera que se realizaban las tareas más prioritarias que eran más difícil que cambiaron con el paso del tiempo, y se dejó para el final ciertas tareas que no estaban tan definidas.

- **Riesgo de problemas técnicos:** Durante el desarrollo del proyecto, se han presentado diversos problemas técnicos relacionados con la implementación de microservicios, la integración de APIs y la contenerización del sistema. Para mitigar este riesgo, se ha realizado una investigación exhaustiva sobre las tecnologías utilizadas y se han seguido buenas prácticas de desarrollo. Además, se ha mantenido una documentación detallada del proceso de desarrollo para facilitar la resolución de problemas.

En caso de que se presentaran problemas técnicos que no pudieran resolverse, se ha mantenido una comunicación constante con el director del TFG para buscar soluciones y alternativas.

- **Imposibilidad de cumplir con los plazos establecidos:** Dada la naturaleza individual del proyecto, existe el riesgo de no poder cumplir con los plazos establecidos en el cronograma. Para mitigar este riesgo, se ha realizado una planificación detallada de las tareas y se ha mantenido un seguimiento constante del progreso. Además, se han establecido hitos intermedios para evaluar el avance del proyecto y realizar ajustes si es necesario.
- **Acceso a la información necesaria:** Durante el desarrollo del proyecto, se ha dependido de la disponibilidad de información externa (horarios académicos, autenticación institucional, etc.). Para mitigar este riesgo, se ha mantenido una comunicación constante con el Product Owner y se han explorado alternativas

en caso de que no se pudiera acceder a la información necesaria. Además, se ha optado por implementar un sistema de scrapping para obtener los horarios académicos de la web de “Grados UGR” como una solución alternativa, y se ha mantenido persistencia de los datos en la base de datos del sistema para evitar depender de la disponibilidad de la web.

5.5. Presupuesto del proyecto

En esta sección se presenta el presupuesto del proyecto, que incluye los costes de personal, adquisición de equipamiento informático, costes de infraestructura y operación del servidor, y gastos de suministros durante el desarrollo.

5.5.1. Presupuesto en formato tabla

Concepto	Importe (€)
Costes de Personal	5.647,50
Adquisición de Equipamiento Informático	968,25
Costes de Infraestructura y Operación del Servidor	260,00
Gastos de Suministros (Desarrollo)	200,00
Total Presupuestado	7.075,75

Tabla 5.1: Resumen de costes del proyecto.

5.5.2. Desglose de la información

El desglose de la información del presupuesto se presenta a continuación, detallando cada uno de los conceptos incluidos en el presupuesto del proyecto.

Costes de Personal

Horas trabajadas: El desarrollo del proyecto ha constado de **376,5 horas** de trabajo, distribuidas entre tareas y sprints según la planificación del proyecto.

Tarifa horaria aplicada: Aunque la plataforma Glassdoor [63] indica que el salario medio de un desarrollador fullstack junior es de **23.062 €/año** (aproximadamente **11 €/hora**), se ha considerado una tarifa de **15 €/hora** debido a la experiencia y cualificación del desarrollador responsable.

Cálculo del coste de personal:

- Total: $376,5 \text{ horas} \times 15 \text{ €/hora} = 5.647,50 \text{ €}$
- Coste mensual estimado (4 meses): $5.647,50 \text{ €} \div 4 = 1.411,88 \text{ €/mes}$

Gastos de Suministros (Desarrollo) Los gastos de suministros incluyen los costes derivados del uso de recursos básicos durante el desarrollo del proyecto, tales como Internet y electricidad.

Durante el desarrollo (4 meses):

■ **Internet:**

- Conexión de alta velocidad necesaria para tareas de desarrollo y pruebas.
- Coste estimado: 30 €/mes
- Total: $30 \text{ €} \times 4 \text{ meses} = 120,00 \text{ €}$

■ **Electricidad:**

- Energía eléctrica para los equipos informáticos utilizados durante el desarrollo.
- Coste estimado: 20 €/mes
- Total: $20 \text{ €} \times 4 \text{ meses} = 80,00 \text{ €}$

Coste total de suministros (desarrollo): 120,00 € (Internet) + 80,00 € (Electricidad) = 200,00 €

Adquisición de Equipamiento Informático Para la fase de despliegue del proyecto y para disponer del hardware adecuado, se ha adquirido el siguiente equipamiento informático. Los costes detallados provienen del presupuesto A25 1091 de DOCUMEDIA, S.L. con fecha 04-03-2025, destinado a la UNIVERSIDAD DE GRANADA -INGENIERIA COMPUTADORES.

- PC MINI ASUS NUC 13 PRO RNUC13ANKI700021 (CPU i7-1360P): 626,00 €
- Memoria RAM KINGSTON SODIMM DDR4 16GB 3200MHZ CL22 (2 unidades): 64,46 €
- SSD WD 1TB M.2 BLACK SN770 PCI-E NVME 4.0: 70,25 €
- DATA SWITCH KVM AISENS 2P HDMI + TECLADO + RATON USB (2 PC): 40,50 €

El subtotal de estos componentes (Base Imponible) asciende a 800,21 €. A este importe se le aplica un IVA del 21 % (168,04 €), resultando en un total de 968,25 €.

Especificaciones principales del servidor resultante:

- CPU: Intel Core i7-1360P (integrada en ASUS NUC 13 Pro)
- RAM: 32 GB DDR4 (2 x 16GB SODIMM)
- Almacenamiento: SSD 1TB M.2 NVMe PCIe 4.0
- Conectividad: Red de alta velocidad (Ethernet 1 Gbps)

Esta máquina estará en funcionamiento 24/7 para asegurar la disponibilidad continua del sistema.

Costes de Infraestructura y Operación del Servidor Estos costes cubren la conexión inicial del servidor a la red universitaria y su operación continua durante un periodo estimado de 4 meses.

- **Conexión a la red del CSIRC:** El departamento de Ingeniería de Computadores (ICAR) ha sufragado el coste de conexión del servidor a la red del Centro de Servicios de Informática y Redes de Comunicaciones (CSIRC), con un importe de **120,00 €**.
- **Electricidad (Servidor):** Estimación del coste por funcionamiento continuo del servidor (24/7).
 - Estimación mensual: 25 €/mes
 - Periodo considerado: 4 meses
 - Total: $25 \text{ €} \times 4 \text{ meses} = 100,00 \text{ €}$
- **Conectividad (Servidor):** Estimación del coste adicional por uso intensivo de red para servir peticiones de clientes.
 - Estimación mensual adicional: 10 €/mes
 - Periodo considerado: 4 meses
 - Total: $10 \text{ €} \times 4 \text{ meses} = 40,00 \text{ €}$

Coste total de infraestructura y operación del servidor (4 meses): 120,00 € (Red CSIRC) + 100,00 € (Electricidad) + 40,00 € (Conectividad) = **260,00 €**.

6. Diseño del sistema: Arquitectura, tecnologías y decisiones clave

Punto a revisar
– Juanmi

Este capítulo profundiza en el **diseño del sistema**, abarcando desde su **arquitectura general** y la **elección de tecnologías y frameworks**, hasta el **diseño de la base de datos** y la **API**. También se detallan el **diseño de la interfaz de usuario (UI)** y la **experiencia de usuario (UX)**, elementos cruciales para garantizar una aplicación intuitiva y accesible.

6.1. Decisiones de diseño y arquitectura: Cimientos del sistema

Esta sección es fundamental, ya que explica la **arquitectura general del sistema**, definiendo cómo se organizan sus componentes y cómo interactúan. La elección arquitectónica es decisiva para la **escalabilidad, mantenibilidad y robustez** de la aplicación a largo plazo.

6.1.1. El Backend como pilar central

Desde el inicio del proyecto, el **diseño del backend** se conceptualizó como la **piedra angular del sistema**. Dada la complejidad de la lógica de negocio y la gestión de datos, se priorizó una **infraestructura robusta y escalable** capaz de manejar eficientemente las operaciones principales. Esta decisión asegura la **estabilidad, rendimiento y seguridad**, independientemente de la interfaz de usuario.

6.1.2. Arquitectura de Microservicios: Flexibilidad y escalabilidad

Para cumplir con los exigentes requisitos de escalabilidad, flexibilidad y mantenibilidad a largo plazo, se optó por una **arquitectura basada en microservicios** para el backend. Esta elección permite descomponer la aplicación en **componentes independientes y débilmente acoplados**, cada uno con una funcionalidad de negocio específica.

Tabla 6.1: Comparativa: Arquitectura Monolítica vs. Microservicios

Característica	Arquitectura Monolítica	Arquitectura de Microservicios
Velocidad de Desarrollo Inicial	✓✓✓	✓
Escalabilidad Independiente		✓✓✓
Mantenibilidad a Largo Plazo	✓	✓✓✓
Flexibilidad Tecnológica (Políglota)		✓✓✓
Resiliencia (Aislamiento de Fallos)	✓	✓✓✓
Despliegues Rápidos y Frecuentes	✓	✓✓✓
Complejidad Operacional	✓✓✓	✓
Coste Inicial de Infraestructura	✓✓✓	✓
Adaptabilidad a Cambios	✓	✓✓✓
Independencia de Equipos	✓	✓✓✓

Comparada con una arquitectura monolítica, los microservicios ofrecen ventajas significativas. Mientras que un monolito puede ser más sencillo de implementar inicialmente, su mantenimiento y escalabilidad eficiente se complican a medida que la aplicación crece. Los microservicios, en cambio, fomentan una mayor **modularidad y reutilización**, facilitando el **desarrollo ágil y la integración continua**. Cada microservicio puede ser **desarrollado, desplegado y escalado de forma independiente**, lo que agiliza la implementación de nuevas funcionalidades y la adaptación a cambios en los requisitos.

6.1.3. Separación Backend-Frontend: Comunicación vía API REST

Una decisión arquitectónica fundamental, coherente con los microservicios, es la **completa separación entre el backend y el frontend**. Esto implica que el backend funciona como una entidad autónoma, sin conocimiento directo de la presentación al usuario. Esta independencia ofrece ventajas esenciales:

- **Flexibilidad de Desarrollo:** Permite que equipos de frontend y backend trabajen en paralelo con distintas tecnologías, acelerando el ciclo de desarrollo.

- **Reusabilidad de Servicios:** Un mismo backend puede soportar múltiples interfaces de usuario (web, móvil), evitando la duplicidad de lógica de negocio.
- **Escalabilidad Horizontal Independiente:** Facilita el escalado individual de cada componente (frontend o backend) según la demanda, optimizando los recursos.
- **Mantenibilidad Mejorada:** Cambios en una capa (ej. refactorización del frontend) no afectan directamente a la otra, reduciendo riesgos y simplificando el mantenimiento.
- **Seguridad Robusta:** Permite implementar medidas de seguridad específicas y más robustas para cada componente, especialmente en el backend, que maneja datos sensibles.

La comunicación exclusiva entre el backend y el frontend se realiza mediante una **API REST (Representational State Transfer)** [6.1](#). La elección de REST se basa en su **simplicidad, naturaleza sin estado y amplia adopción**, lo que facilita la integración e interoperabilidad. La API REST define un conjunto de *endpoints* y utiliza un formato estandarizado, generalmente **JSON (JavaScript Object Notation)**, para el intercambio de datos.

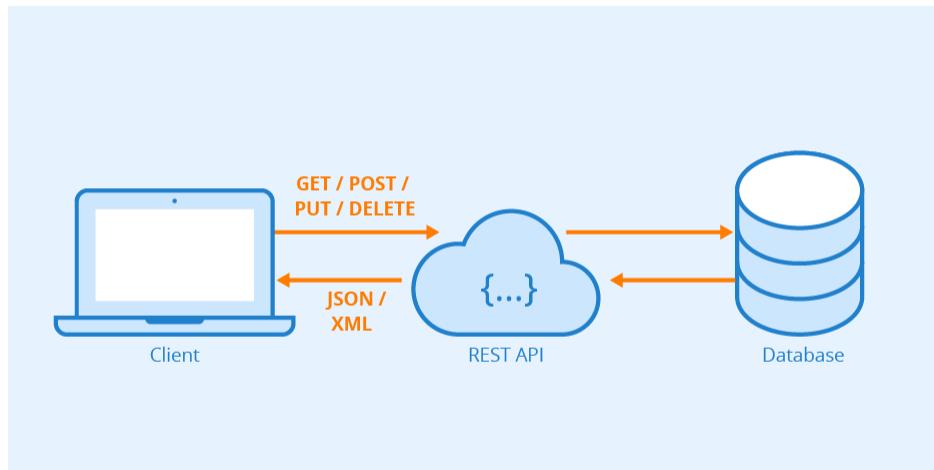


Figura 6.1: Comunicación Backend-Frontend a través de API REST

6.2. Backend: Diseño y tecnologías clave

Una vez definidos los requisitos del sistema, se procedió a diseñar el **backend** de la aplicación, siguiendo la arquitectura de microservicios, donde cada servicio gestiona una funcionalidad específica.

6.2.1. Servicios del Backend identificados

Los servicios identificados para cumplir con los requisitos del sistema son:

- **Servicio de Autenticación y Autorización (Auth Service):** Gestiona el acceso, registro, inicio de sesión y permisos de usuarios.
- **Servicio de Gestión de Usuarios (User Service):** Responsable de la creación, actualización y eliminación de perfiles de usuario.
- **Servicio de Gestión de Horarios y Calendario (Schedule Consumer Service):** Obtiene y administra el horario académico desde el sistema de la UGR.
- **Servicio de Notificaciones (Mail Service):** Envía notificaciones a los usuarios (creación de eventos, registro completado, reseteo de contraseña, etc.).
- **Servicio de Matriculaciones (Academic Subscription Service):** Gestiona las matriculaciones de usuarios en asignaturas y grupos, actualiza el horario personalizado y crea eventos adicionales.

Cada servicio se implementa como una aplicación independiente para una mayor flexibilidad y escalabilidad. La comunicación entre ellos se realiza tanto vía **API REST** como mediante **eventos a través de RabbitMQ**, lo que crea una arquitectura más robusta y desacoplada. Un **API Gateway** actúa como punto de entrada para todas las solicitudes del frontend, enrutándolas, gestionando la autenticación/autorización y aportando una capa adicional de seguridad.

6.2.2. Tecnologías y Frameworks del Backend

Para el desarrollo del backend, se eligió el stack tecnológico de **Java con Spring Boot**, que ofrece ventajas significativas para microservicios:

- **Spring Boot:** Facilita la creación de aplicaciones Java independientes y productivas con mínima configuración.
- **Spring Security:** Proporciona un marco completo para una autenticación y autorización robustas.
- **Spring Data JPA:** Simplifica el acceso a bases de datos relacionales mediante JPA.
- **Spring Cloud:** Ofrece herramientas para construir aplicaciones distribuidas (descubrimiento de servicios, configuración centralizada, gestión de circuitos).

Spring Framework es una tecnología líder en Java, con un vasto ecosistema de herramientas y bibliotecas que facilitan la creación de aplicaciones robustas y escalables. La elección de Java como lenguaje de programación capitaliza un lenguaje maduro, ampliamente adoptado, con una gran comunidad, un ecosistema rico y compatibilidad con diversas plataformas y sistemas operativos, además de su velocidad de ejecución [64] en comparación con otros lenguajes usados en backend, reflejado en la figura 6.2.

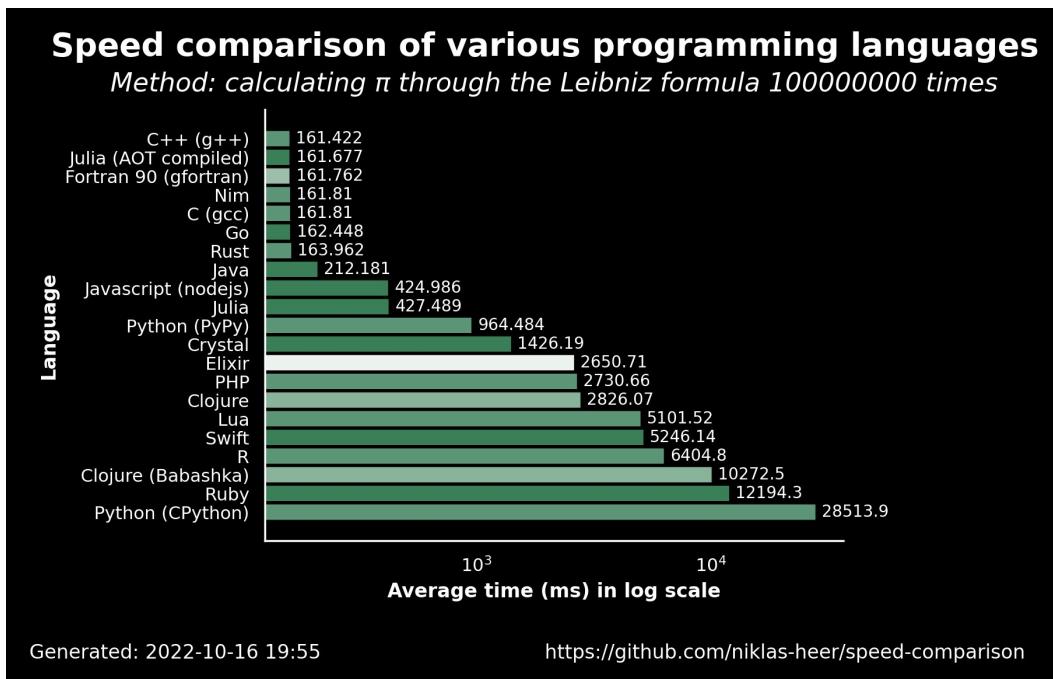


Figura 6.2: Comparativa de velocidades entre lenguajes de programación

Para el descubrimiento de servicios en esta arquitectura de microservicios, se integró **Eureka**, que permite a los servicios registrarse y descubrirse entre sí en la red. Sus ventajas incluyen:

- **Descubrimiento de Servicios:** Facilita la comunicación y el enrutamiento de solicitudes entre microservicios.
- **Escalabilidad:** Permite añadir o eliminar instancias de servicios sin reconfiguración manual.
- **Resiliencia:** Ofrece mecanismos para manejar fallos de servicios, manteniendo la aplicación funcional.
- **Configuración Centralizada:** Simplifica la administración y el despliegue al centralizar la configuración de los servicios.

Como base de datos relacional para “User Service” y “Schedule Consumer Service” se optó por **MySQL**, un sistema de gestión de bases de datos ampliamente utilizado y confiable, que ofrece robustez, escalabilidad y un sólido soporte para transacciones. MySQL es ideal para aplicaciones que requieren integridad referencial y consultas complejas, lo que lo convierte en una excelente opción para manejar los datos de usuarios y horarios académicos.

La comunicación asíncrona y desacoplada entre servicios se logra mediante **RabbitMQ**, un sistema de mensajería ampliamente utilizado que proporciona alta disponibilidad, escalabilidad y fiabilidad en la entrega de mensajes, siendo una excelente elección para la arquitectura de microservicios. Se eligió RabbitMQ sobre alternativas como Kafka o ActiveMQ por su simplicidad, facilidad de uso, amplia adopción y compatibilidad.

Finalmente, Docker se seleccionó para la **contenedorización de los servicios**, permitiendo una fácil implementación y escalabilidad. Docker proporciona un entorno aislado y reproducible para cada servicio, simplificando el despliegue en diferentes entornos (desarrollo, pruebas, producción) y mejorando la portabilidad.

Tras la selección de tecnologías y la definición de las interacciones entre los servicios, se establece la siguiente arquitectura general del backend en la figura 6.3:

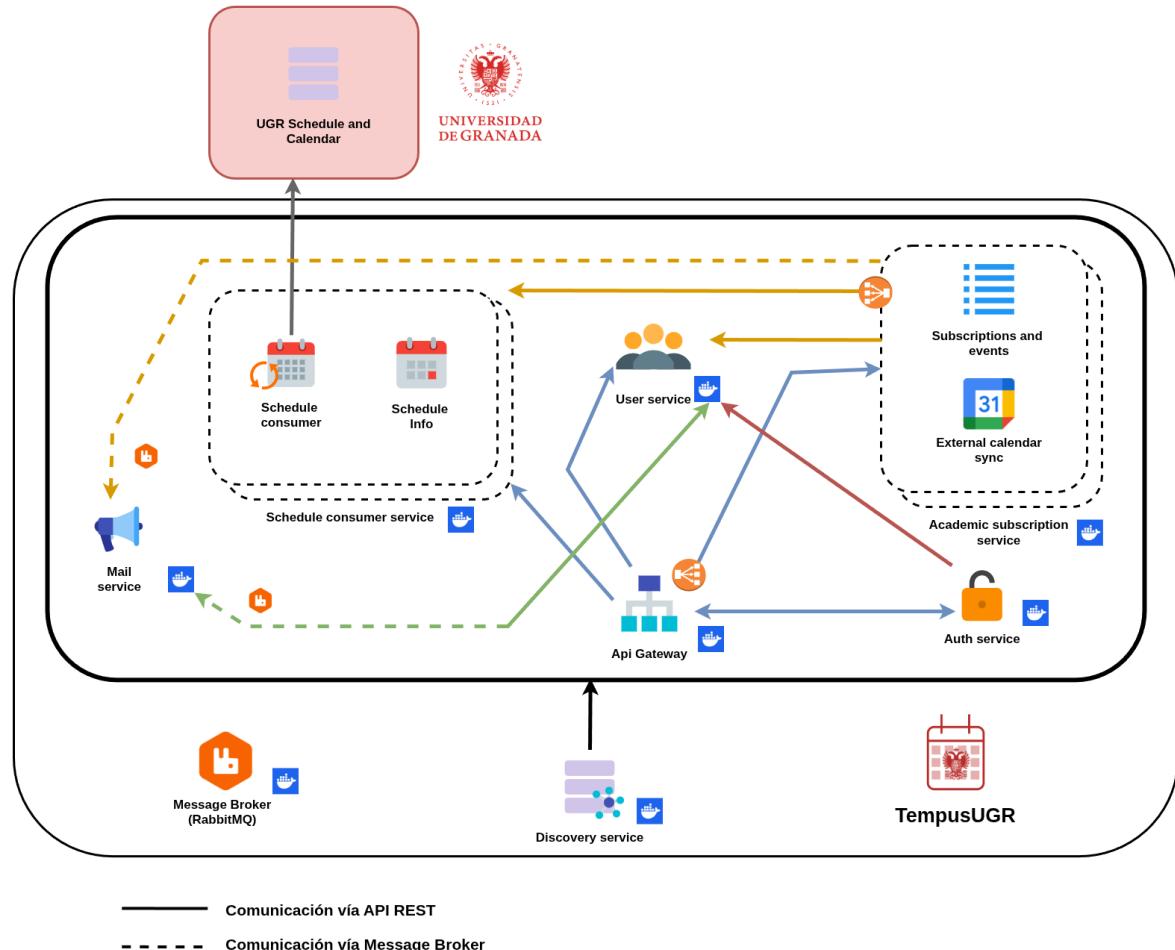


Figura 6.3: Arquitectura general del Backend

Comunicaciones entre servicios

Como se observa en el diagrama, las comunicaciones se distinguen por dos tipos principales:

- **Comunicación vía API REST (Líneas Sólidas Negras y de Colores):** Este es el método principal para las interacciones síncronas, donde un servicio realiza una solicitud directa a otro y espera una respuesta inmediata. El **API Gateway** actúa como el punto de entrada para todas las solicitudes externas (provenientes de TempusUGR) y las enruta a los servicios correspondientes.

- **API Gateway ↔ Auth Service:** El **API Gateway** se comunica con el

Auth Service para gestionar la autenticación y autorización de los usuarios en cada solicitud entrante, asegurando que solo usuarios válidos y con permisos adecuados accedan a los recursos del sistema.

- **API Gateway ↔ User Service:** Las solicitudes relacionadas con la gestión de usuarios (ej., creación, consulta, modificación de perfiles) son enrutadas desde el **API Gateway** al **User Service**.
- **API Gateway ↔ Academic Subscription Service:** Las peticiones para gestionar suscripciones académicas, eventos personalizados y la sincronización con sistemas de calendario externos (ej., Google Calendar) se dirigen desde el **API Gateway** al **Academic Subscription Service**.
- **API Gateway ↔ Schedule Consumer Service:** Las solicitudes para obtener información sobre horarios académicos (ej., buscar horarios de titulaciones, asignaturas) son enrutadas desde el **API Gateway** al **Schedule Consumer Service**.
- **Academic Subscription Service ↔ User Service:** El **Academic Subscription Service** necesita comunicarse con el **User Service** para obtener información detallada de los usuarios, como sus identificadores o datos de perfil, cuando gestiona las suscripciones o eventos vinculados a ellos.
- **Academic Subscription Service ↔ Schedule Consumer Service:** El **Academic Subscription Service** interactúa con el **Schedule Consumer Service** para obtener la información de las asignaturas matriculadas por los usuarios, como parte del proceso de construcción de sus horarios personalizados o la validación de suscripciones.
- **Comunicación vía Message Broker (RabbitMQ - Líneas Discontinuas):** Para la comunicación asíncrona y el desacoplamiento entre servicios, se utiliza un **Message Broker (RabbitMQ)**. Este patrón es ideal para notificaciones, procesamiento en segundo plano y cuando un servicio necesita informar a otros sin esperar una respuesta inmediata.
 - **Academic Subscription Service → Message Broker → Mail Service:** Cuando se producen eventos significativos en el **Academic Subscription Service** (ej., creación de una nueva suscripción, adición de un evento personalizado), este publica un mensaje en el Message Broker. El **Mail Service** consume estos mensajes para enviar notificaciones por correo electrónico a los usuarios implicados de forma asíncrona.
 - **User Service → Message Broker → Mail Service:** De manera similar, cuando se realizan acciones relacionadas con el perfil de usuario en el **User Service** (ej., registro de un nuevo usuario, reseteo de contraseña), este publica mensajes en el Message Broker. El **Mail Service** los consume para enviar comunicaciones relevantes a los usuarios, como correos de bienvenida o confirmaciones.

Además de estos mecanismos de comunicación explícitos, es importante

destacar el papel del **Discovery Service (Eureka)**. Aunque no se muestra directamente en el flujo de datos del diagrama, es fundamental para que los microservicios puedan encontrarse y comunicarse entre sí de manera dinámica y resiliente, sin necesidad de direcciones IP o puertos codificados rígidamente.

Esta combinación de comunicación síncrona (API REST a través de API Gateway) y asíncrona (RabbitMQ) permite construir una arquitectura **robusta, escalable y mantenible**, donde los servicios pueden evolucionar de forma independiente y reaccionar a los eventos del sistema de manera eficiente y desacoplada.

Obtención de horarios académicos

El **Servicio de Horarios y Calendario (Schedule Consumer Service)** es el encargado principal de recolectar la información de horarios académicos. Para ello, se emplea una técnica de **web scraping** dirigida a la web oficial de grados de la Universidad de Granada (UGR). Este proceso consiste en la extracción automatizada de datos estructurados desde páginas web.

El flujo de obtención de horarios sigue los siguientes pasos:

1. **Identificación de URLs:** El servicio está configurado con las URLs de las páginas web de grados de la UGR que contienen la información de los horarios.
2. **Raspado de datos:** Se utiliza un algoritmo de web scraping para navegar por estas páginas, identificar los elementos HTML que contienen la información relevante (ej., tablas de horarios, nombres de asignaturas, grupos, profesores, aulas y fechas) y extraerlos de forma programática.
3. **Procesamiento y Normalización:** Los datos extraídos, que inicialmente pueden estar en un formato inconsistente o no estructurado, son procesados y normalizados para ajustarse al modelo de datos definido en la base de datos del **Schedule Consumer Service** (entidades Grade, Subject, Subject_group, Class_info). Este paso es crucial para asegurar la coherencia y la integridad de la información.
4. **Almacenamiento:** Una vez normalizados, los datos se persisten en la base de datos **MySQL** asociada al servicio.

Esta estrategia de web scraping permite mantener la base de datos de horarios académicos actualizada con la información publicada por la UGR, garantizando que el sistema TempusUGR opere con los datos más recientes y precisos para la gestión de horarios y matriculaciones.

Balanceo de Carga

Como se refleja en la arquitectura, el sistema se ha preparado para levantar varias instancias de los servicios “Academic Subscription Service” y “Schedule

Consumer Service". Esto permite distribuir la carga de trabajo entre múltiples instancias, mejorando la capacidad de respuesta y la disponibilidad del sistema. El balanceo de carga se gestiona a través del **API Gateway** y **Academic Subscription Service**, que enrutan las solicitudes entrantes a la instancia adecuada del servicio, basándose en criterios como la disponibilidad, la carga actual o el tipo de solicitud.

EL algoritmo de balanceo de carga utilizado es el **Round Robin**, que distribuye las solicitudes entrantes de manera equitativa entre todas las instancias disponibles. Este enfoque es simple y efectivo, especialmente en escenarios donde las instancias tienen capacidades similares y se espera una carga uniforme.

6.2.3. Modelado de la base de datos

Una vez definidos los requisitos de información del sistema, y los sistemas de gestión de bases de datos a usar en cada servicio, se procedió a diseñar el modelo de datos 6.4 para cada uno de los servicios del backend. A continuación se muestran los diagramas de entidad-relación (ER) para cada uno de estos:

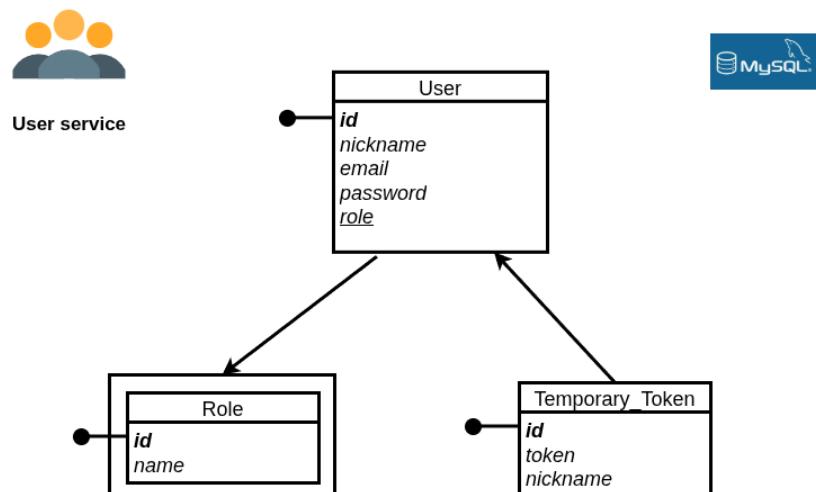


Figura 6.4: Modelo de datos del Servicio de Usuarios

A continuación, se detalla la estructura y propósito de cada entidad:

- **Entidad User:** Esta entidad representa a los usuarios del sistema y es el núcleo del servicio. Almacena la información esencial para la identificación y autenticación.
 - **id:** Clave primaria única para cada usuario.
 - **nickname:** Nombre de usuario único, utilizado para la identificación en el sistema.
 - **email:** Dirección de correo electrónico del usuario, también única y utilizada para comunicaciones y recuperación de cuenta.

- **password:** Contraseña del usuario, almacenada de forma segura (normalmente como un hash).
- **role:** Hace referencia al rol del usuario, estableciendo una relación con la entidad **Role**.
- **Entidad Role:** Define los distintos roles o perfiles que un usuario puede tener dentro del sistema, lo que permite implementar un control de acceso basado en roles (RBAC).
 - **id:** Clave primaria única para cada rol.
 - **name:** Nombre del rol (ROLE_INACTIVE, ROLE_STUDENT, ROLE_TEACHER, ROLE_ADMIN).

La relación entre **User** y **Role** es de uno a muchos (un rol puede ser asignado a múltiples usuarios), o de muchos a muchos si un usuario pudiera tener varios roles (aunque el diagrama sugiere una relación de uno a muchos a través del campo **role** en **User**). Para mayor claridad en el futuro, se podría especificar el tipo de relación.

- **Entidad Temporary-Token:** Esta entidad se utiliza para gestionar tokens de un solo uso, comúnmente empleados para procesos como la recuperación de contraseña o la verificación de correo electrónico.
 - **id:** Clave primaria única para cada token temporal.
 - **token:** El valor único del token generado.
 - **nickname:** Hace referencia al **nickname** del usuario al que está asociado el token, estableciendo una relación directa con la entidad **User**.

Esta relación implica que cada token temporal está asociado a un usuario específico.

Este diseño de base de datos 6.5 proporciona una estructura sólida para el **Servicio de Gestión de Usuarios**, permitiendo una gestión eficiente y segura de la información de los usuarios, sus permisos y los mecanismos de autenticación adicionales.

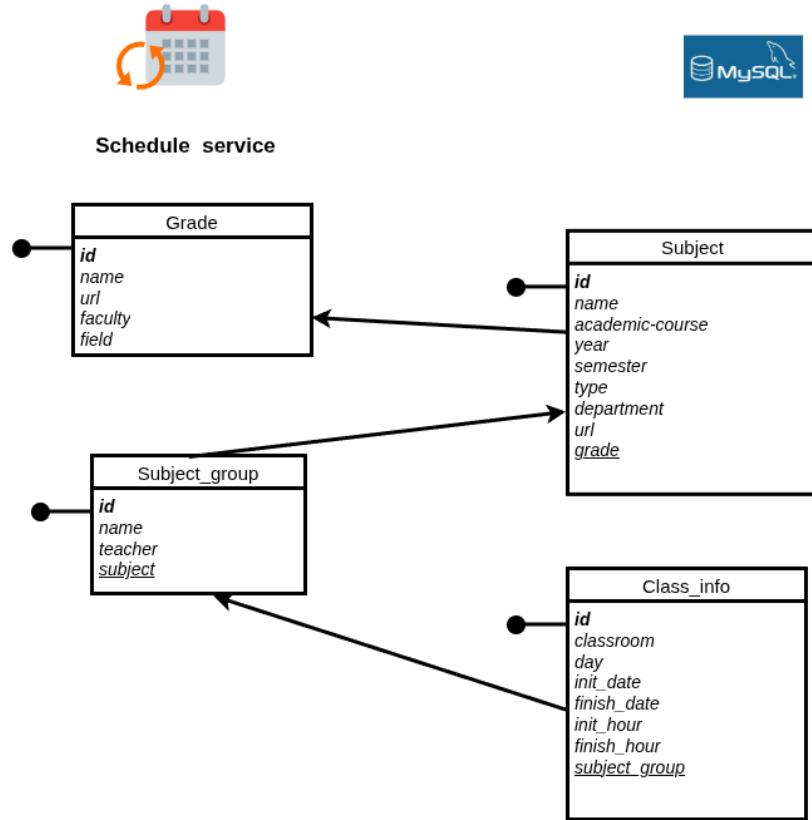


Figura 6.5: Modelo de datos del Servicio de Horarios y Calendario

A continuación, se describen las entidades y sus relaciones:

- **Entidad Grade (Grado):** Representa las diferentes titulaciones académicas de las que se obtienen los horarios.
 - **id:** Clave primaria única para cada titulación.
 - **name:** Nombre completo de la titulación (ej., "Grado en Ingeniería Informática").
 - **url:** URL o identificador del recurso externo de donde se obtienen los datos de la titulación.
 - **faculty:** Facultad a la que pertenece la titulación.
 - **field:** Área de conocimiento o campo de estudio de la titulación.
- **Entidad Subject (Asignatura):** Contiene la información de las asignaturas que forman parte de las titulaciones.
 - **id:** Clave primaria única para cada asignatura.
 - **name:** Nombre de la asignatura.
 - **academic-course:** Curso académico al que pertenece la asignatura.
 - **year:** Año del plan de estudios en el que se imparte la asignatura.

- **semester**: Semestre en el que se imparte la asignatura.
 - **type**: Tipo de asignatura (ej., “Obligatoria”, “Optativa”).
 - **department**: Departamento responsable de la asignatura.
 - **url**: URL o identificador del recurso externo de donde se obtienen los datos de la asignatura.
 - **grade**: Clave foránea que referencia a la entidad **Grade**, indicando a qué titulación pertenece la asignatura (relación uno a muchos: una titulación tiene muchas asignaturas).
- **Entidad Subject_group (Grupo de Asignatura)**: Representa los diferentes grupos en los que se divide una asignatura para su impartición.
- **id**: Clave primaria única para cada grupo de asignatura.
 - **name**: Nombre o identificador del grupo (ej., “Grupo 1”, “Grupo A”).
 - **teacher**: Profesor o profesores asignados a este grupo.
 - **subject**: Clave foránea que referencia a la entidad **Subject**, indicando a qué asignatura pertenece este grupo (relación uno a muchos: una asignatura tiene muchos grupos).
- **Entidad Class_info (Información de Clase)**: Almacena los detalles específicos de cada sesión de clase para un grupo de asignatura.
- **id**: Clave primaria única para cada entrada de clase.
 - **classroom**: Aula o lugar donde se imparte la clase.
 - **day**: Día de la semana en que se imparte la clase.
 - **init_date**: Fecha de inicio de la clase o del período de clases.
 - **finish_date**: Fecha de finalización de la clase o del período de clases.
 - **init_hour**: Hora de inicio de la clase.
 - **finish_hour**: Hora de finalización de la clase.
 - **subject_group**: Clave foránea que referencia a la entidad **Subject_group**, indicando a qué grupo de asignatura pertenece esta clase (relación uno a muchos: un grupo puede tener muchas clases).

Este esquema de base de datos 6.6 permite al **Servicio de Horarios y Calendario** organizar y gestionar de manera eficiente toda la información académica necesaria para la composición de los horarios de los usuarios.

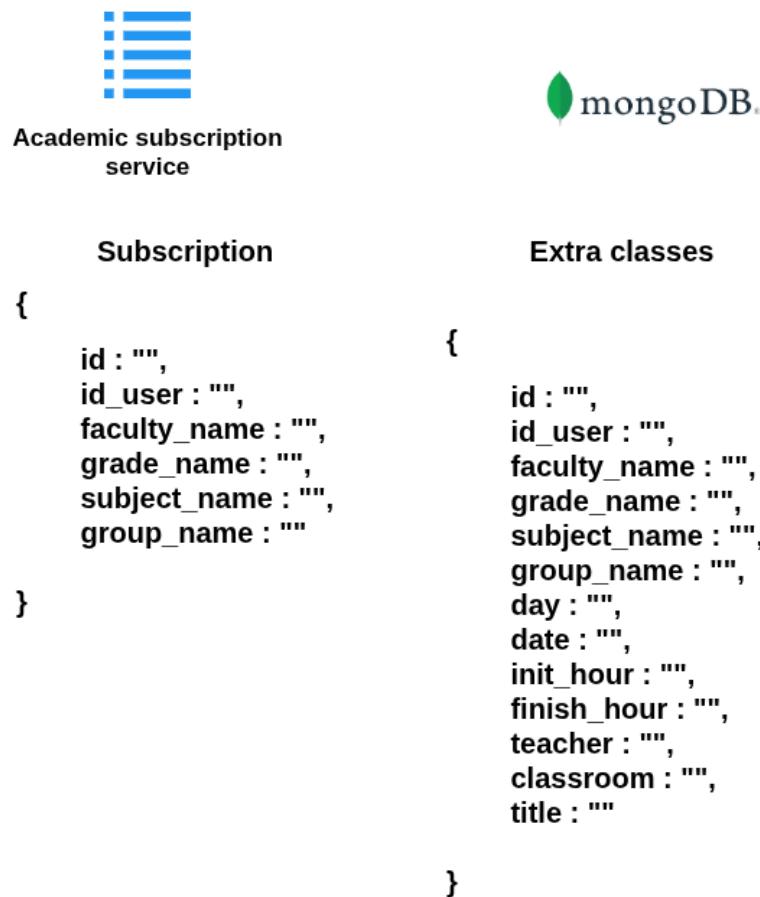


Figura 6.6: Modelo de datos del Servicio de Matriculaciones

A continuación, se detalla la estructura esperada de los documentos en cada colección:

- **Colección Subscription:** Esta colección almacena las matriculaciones de los usuarios en asignaturas y grupos específicos, reflejando su horario oficial.
 - **id:** Identificador único del documento de suscripción.
 - **id_user:** Identificador del usuario al que pertenece esta suscripción. Permite vincular la suscripción a un usuario específico.
 - **faculty_name:** Nombre de la facultad de la titulación asociada.
 - **grade_name:** Nombre de la titulación a la que se suscribe el usuario.
 - **subject_name:** Nombre de la asignatura a la que el usuario está matriculado.
 - **group_name:** Nombre del grupo de la asignatura al que el usuario está matriculado.

Esta estructura flexible permite registrar la información de la matriculación sin la rigidez de un esquema relacional fijo, adaptándose a posibles variaciones en los detalles académicos.

- **Colección Extra classes:** Esta colección está diseñada para almacenar clases o eventos adicionales que los usuarios puedan crear para su horario personalizado, no directamente vinculados al horario oficial.

- **id:** Identificador único del documento de clase extra.
- **id_user:** Identificador del usuario que creó esta clase adicional.
- **faculty_name:** Nombre de la facultad relacionada con la clase (opcional, para contexto).
- **grade_name:** Nombre de la titulación relacionada (opcional, para contexto).
- **subject_name:** Nombre de la asignatura (si aplica, para contexto).
- **group_name:** Nombre del grupo (si aplica, para contexto).
- **day:** Día de la semana en que se imparte la clase extra.
- **date:** Fecha específica de la clase extra.
- **init_hour:** Hora de inicio de la clase extra.
- **finish_hour:** Hora de finalización de la clase extra.
- **teacher:** Nombre del profesor o ponente de la clase extra.
- **classroom:** Aula o lugar donde se imparte la clase extra.
- **title:** Título o descripción breve de la clase extra.

La flexibilidad de MongoDB es particularmente útil aquí, ya que los campos pueden variar entre documentos de "extra classes" dependiendo de la naturaleza del evento.

La elección de MongoDB para estas colecciones permite una **rápida inserción y consulta de datos**, así como una **fácil adaptación a cambios en el esquema de datos** sin requerir migraciones complejas, lo que es ventajoso para la gestión dinámica de las suscripciones y eventos personalizados de los usuarios.

6.2.4. Diseño de la API

El diseño de la API del backend se basa en el principio de **REST (Representational State Transfer)**, que define un conjunto de convenciones para la creación de servicios web escalables y mantenibles. La API está estructurada en torno a recursos, cada uno representado por una URL única, y utiliza los métodos HTTP estándar para interactuar con estos recursos.

Para su estandarización se ha seguido la especificación de la **RFC 7231**[65], que define el HyperText Transfer Protocol (HTTP/1.1) y sus métodos, así como las convenciones para la creación de APIs RESTful. Esta especificación proporciona un marco claro para el diseño de APIs, asegurando que sean intuitivas, predecibles y fáciles de consumir por los clientes.

Además de esta se ha revisado la [RFC 1738](#)^[66], que define el formato de las URLs, asegurando que todas las rutas de la API sigan una estructura coherente y fácil de entender. Esto es crucial para la usabilidad de la API, ya que permite a los desarrolladores comprender rápidamente cómo interactuar con los diferentes recursos del sistema.

Para hacer pruebas y documentar la API de manera efectiva, se ha utilizado **Postman**, una herramienta ampliamente adoptada en la comunidad de desarrollo para probar y documentar APIs. Postman permite a los desarrolladores enviar solicitudes HTTP a la API, inspeccionar las respuestas y crear colecciones de pruebas que pueden ser compartidas y reutilizadas. Además, facilita la generación de documentación interactiva de la API, lo que mejora la experiencia del desarrollador al integrar o consumir los servicios.

Estructura de la API

En primer lugar, y gracias al uso de un **API Gateway**, se ha definido una estructura de URL base para la API, que es la siguiente:

¹ <http://<host>:<port>/calendarugr/v1>

Donde “host” es la ip del host donde se despliega la aplicación y “port” es el puerto en el que escucha el API Gateway. Esta estructura base permite una fácil identificación de la versión de la API y proporciona un punto de entrada común para todas las solicitudes. Además en secciones posteriores al implementar https, y al adquirir un dominio de la UGR, esta URL base se actualizará para reflejar el dominio oficial del servicio.

¹ <https://tempus.ugr.es/calendarugr/v1>

A partir de esta URL base, los diferentes microservicios exponen sus funcionalidades a través de rutas específicas. La API se ha estructurado lógicamente según los servicios del backend, facilitando la comprensión y el uso por parte de los clientes. A continuación, se detallan los principales grupos de endpoints.

Nota sobre Autorización: Todos los endpoints de la API requieren autenticación y autorización mediante JWT, con las siguientes excepciones que son de acceso público o gestionan la autenticación inicial:

- /auth/login
- /auth/refresh
- /user/register
- /user/activate
- /academic-subscription/ics (para la descarga pública de calendarios ICS)

- /academic-subscription/sync-url (para la obtención de URLs públicas de sincronización)
- /user/reset-pass-mail
- /user/reset-password

Endpoints del Servicio de Usuarios (User Service): Gestionan la administración de usuarios, roles y la recuperación de contraseñas.

- **Administración de Usuarios (Admin Endpoints):**

- **POST /user/admin/register**

Descripción: Crea un usuario sin la necesidad de pasar por confirmación de cuenta (para Administrador).

Cuerpo (JSON):

```

1 {
2   "nickname": "string",
3   "email": "string",
4   "password": "string",
5   "role": {
6     "name": "string"
7   },
8   "notification": true
9 }
```

- **PUT /user/admin/update/{id}**

Descripción: Actualiza la información de un usuario por su ID (para Administrador).

Cuerpo (JSON):

```

1 {
2   "nickname": "string",
3   "email": "string",
4   "password": "string",
5   "role": {
6     "name": "string"
7   },
8   "notification": true
9 }
```

- **DELETE /user/admin/delete/{id}**

Descripción: Elimina un usuario por su ID.

- **Gestión de Roles (Role related):**

- **POST /user/role/create**

Descripción: Crea un nuevo rol.

Cuerpo (JSON):

```
1 {
2   "name": "string"
3 }
```

- **GET /user/role/all**

Descripción: Obtiene todos los roles disponibles.

- **DELETE /user/role/delete**

Descripción: Borra un rol existente.

Cuerpo (JSON):

```
1 {
2   "name": "string"
3 }
```

- **Recuperación de Contraseña (Password recovery):**

- **POST /user/reset-pass-mail**

Descripción: Solicita el envío de un correo electrónico para el reseteo de contraseña.

Parámetros de consulta: mail (string)

- **Consulta y Gestión General de Usuarios:**

- **GET /user/all**

Descripción: Obtiene la lista completa de usuarios.

- **GET /user/nickname/{nickname}**

Descripción: Obtiene un usuario por su nickname.

- **GET /user/email/{email}**

Descripción: Obtiene un usuario por su dirección de correo electrónico.

- **GET /user/user-info**

Descripción: Obtiene la información del usuario asociado al token actual.

- **POST /user/register**

Descripción: Registra un nuevo usuario en el sistema.

Cuerpo (JSON):

```
1 {
2   "nickname": "string",
3   "email": "string",
4   "password": "string"
5 }
```

- **POST /user/activate**

Descripción: Activa una cuenta de usuario utilizando un token de activación.

Parámetros de consulta: token (string)

- **PUT /user/deactivate**

Descripción: Desactiva la cuenta del usuario actual.

Cuerpo (JSON):

```
1 {
2   "currentPassword": "string"
3 }
```

- **PUT /user/nickname**

Descripción: Actualiza el nickname del usuario actual.

Cuerpo (JSON):

```
1 {
2   "nickname": "string"
3 }
```

- **PUT /user/role**

Descripción: Cambia el rol de un usuario.

- **PUT /user/password**

Descripción: Actualiza la contraseña del usuario actual.

Cuerpo (JSON):

```
1 {
2   "currentPassword": "string",
3   "newPassword": "string"
4 }
```

- **PUT /user/activate-notifications**

Descripción: Activa las notificaciones para el usuario actual.

- **PUT /user/deactivate-notifications**

Descripción: Desactiva las notificaciones para el usuario actual.

- **POST /user/email-list**

Descripción: Obtiene una lista de correos electrónicos de usuarios basándose en una lista de IDs.

Cuerpo (JSON):

```
1 [
2   "id1",
3   "id2"
4 ]
```

- **Endpoints del Servicio de Autenticación (Auth Service):** Gestionan el proceso de inicio de sesión y la renovación de tokens.

- **POST /auth/login**

Descripción: Permite a un usuario iniciar sesión en el sistema.

Cuerpo (JSON):

```
1 {
2   "email": "string",
3   "password": "string"
4 }
```

- **POST /auth/refresh**

Descripción: Permite renovar un token de acceso utilizando un refresh token.

Cuerpo (JSON):

```
1 {
2   "refreshToken": "string"
3 }
```

- **Endpoints del Servicio de Correo (Mail Service):** Se utilizan para enviar correos electrónicos.

- **POST /email/send**

Descripción: Envía un correo electrónico.

Cuerpo (JSON):

```
1 {
2   "receiver": "string",
3   "subject": "string",
4   "message": "string"
5 }
```

- **Endpoints del Servicio de Horarios (Schedule Consumer Service):** Proporcionan acceso a la información de horarios académicos obtenida mediante web scraping.

- **GET /schedule-consumer/classes-from-group**

Descripción: Obtiene las clases de un grupo específico de una asignatura y titulación.

Parámetros de consulta: grade (string), subject (string), group (string)

- **GET /schedule-consumer/grades**

Descripción: Obtiene la lista de todas las titulaciones disponibles.

- **GET /schedule-consumer/subjects-groups**

Descripción: Obtiene las asignaturas y grupos asociados a una titulación específica.

Parámetros de consulta: grade (string)

- **GET /schedule-consumer/teacher-classes**

Descripción: Obtiene las clases impartidas por un profesor, buscando por nombre parcial.

Parámetros de consulta: partialTeacherName (string)

- **Endpoints del Servicio de Matriculaciones Académicas (Academic Subscription Service):** Permiten a los usuarios gestionar sus suscripciones a asignaturas y grupos, así como crear y gestionar eventos personalizados.

- **GET /academic-subscription/classes**

Descripción: Obtiene las clases a las que el usuario está suscrito.

- **GET /academic-subscription/entire-calendar**

Descripción: Obtiene el calendario completo del usuario (suscripciones y eventos extra).

- **GET /academic-subscription/subscriptions**

Descripción: Obtiene las suscripciones activas del usuario.

- **POST /academic-subscription/subscription**

Descripción: Suscribe al usuario a una asignatura y grupo específicos.

Cuerpo (JSON):

```

1 {
2   "faculty": "string",
3   "grade": "string",
4   "subject": "string",
5   "group": "string"
6 }
```

- **POST /academic-subscription/subscription-batching**

Descripción: Permite suscribirse a múltiples asignaturas y grupos en una sola solicitud.

Cuerpo (JSON):

```

1 [
2   {
3     "faculty": "string",
4     "grade": "string",
5     "subject": "string",
6     "group": "string"
7   }
8 ]
```

- **GET /academic-subscription/ics**

Descripción: Permite descargar el calendario del usuario en formato ICS.

- **GET /academic-subscription/sync-url**

Descripción: Obtiene una URL pública para sincronizar el calendario ICS del usuario con otras aplicaciones.

- **DELETE /academic-subscription/subscription-grade**

Descripción: Borra todas las suscripciones de un usuario para una titulación específica.

- **DELETE /academic-subscription/subscription**

Descripción: Borra una suscripción específica de un usuario.

- **GET /academic-subscription/group-event**

Descripción: Obtiene los eventos de grupo (clases extra) creados por el usuario.

- **POST /academic-subscription/group-event**

Descripción: Inserta una nueva clase extra o evento de grupo para el usuario.

Cuerpo (JSON):

```
1 {
2     "classroom": "string",
3     "day": "string",
4     "date": "YYYY-MM-DD",
5     "initHour": "HH:MM:SS",
6     "finishHour": "HH:MM:SS",
7     "groupName": "string",
8     "subjectName": "string",
9     "teacher": "string",
10    "gradeName": "string",
11    "facultyName": "string",
12    "title": "string"
13 }
```

- **DELETE /academic-subscription/group-event**

Descripción: Borra un evento de grupo específico por su ID.

- **GET /academic-subscription/faculty-group-event**

Descripción: Recoge todos los eventos creados por la facultad del usuario.

- **POST /academic-subscription/faculty-event**

Descripción: Crea un evento a nivel de facultad.

Cuerpo (JSON):

```
1 {
2     "day": "string",
3     "date": "YYYY-MM-DD",
4     "initHour": "HH:MM:SS",
5     "finishHour": "HH:MM:SS",
6     "facultyName": "string",
7     "title": "string"
8 }
```

- **DELETE /academic-subscription/faculty-event**

Descripción: Borra un evento de facultad por su ID.

6.3. Frontend: Diseño y tecnologías clave

El frontend de TempusUGR se ha desarrollado utilizando **Angular**, un framework de desarrollo web que permite crear aplicaciones de una sola página (SPA) de manera eficiente y escalable. Angular es conocido por su arquitectura basada en componentes, lo que facilita la reutilización de código y la separación de preocupaciones, permitiendo un desarrollo más organizado y mantenable.

Se alinea con las mejores prácticas de desarrollo web moderno, incluyendo el uso de **TypeScript** como lenguaje principal, lo que proporciona tipado estático y características avanzadas de programación orientada a objetos. Esto mejora la calidad del código y facilita la detección temprana de errores durante el desarrollo.

Además, Angular cuenta con un robusto sistema de inyección de dependencias, lo que permite una gestión eficiente de los servicios y componentes, mejorando la modularidad y la testabilidad de la aplicación. También incluye herramientas integradas para el manejo del enrutamiento, la gestión del estado y la comunicación con APIs RESTful, lo que simplifica el desarrollo de aplicaciones complejas.

6.3.1. Diseño de la Interfaz y Experiencia del Usuario (UI/UX)

En cuanto a la parte visual del proyecto, se tuvo en mente desde el principio tener una interfaz usable, intuitiva y accesible, de manera que se facilitara lo máximo posible el acceso a la información del horario personalizado.

Para ello, se optó por un diseño minimalista [6.7](#), con una paleta de colores clara y un uso moderado de imágenes. Además se utilizó la tipografía “Segoe UI” por su diseño moderno con letras redondeadas y diseño limpio que se ve bien en pantallas y papel.

- Color primario: #b82d2a 
- Color secundario: #e4afaf 
- Color de fondo: #f5f5f5 
- Color de texto: #333333 



Figura 6.7: Logo de TempusUGR

El diseño de la interfaz se ha centrado en la usabilidad, asegurando que los usuarios puedan navegar fácilmente por las diferentes secciones de la aplicación. Se han implementado menús claros y botones intuitivos para facilitar la interacción con el sistema. Además, se ha prestado especial atención a la accesibilidad, siguiendo las pautas WCAG (Web Content Accessibility Guidelines) para garantizar que la aplicación sea usable por personas con discapacidades.

Además se han seguido los diez principios de diseño de Jakob Nielsen^[67], que son:

1. **Visibilidad del estado del sistema:** La aplicación proporciona retroalimentación clara sobre las acciones del usuario, como confirmaciones de suscripciones o errores en la entrada de datos.
2. **Coincidencia entre el sistema y el mundo real:** Se utilizan términos y conceptos familiares para los usuarios, evitando jerga técnica innecesaria.
3. **Control y libertad del usuario:** Los usuarios pueden deshacer acciones fácilmente, como cancelar una suscripción o eliminar un evento.
4. **Consistencia y estándares:** Se mantiene una terminología y diseño coherentes en toda la aplicación.
5. **Prevención de errores:** Se implementan validaciones para evitar entradas incorrectas, como fechas inválidas o grupos inexistentes.
6. **Reconocimiento en lugar de recuerdo:** Los menús y opciones son visibles y accesibles, reduciendo la carga cognitiva del usuario.
7. **Flexibilidad y eficiencia de uso:** La aplicación permite a los usuarios realizar tareas comunes rápidamente mediante atajos y opciones avanzadas.
8. **Diseño estético y minimalista:** La interfaz es limpia y sin distracciones innecesarias, enfocándose en la funcionalidad principal.
9. **Ayuda a los usuarios a reconocer, diagnosticar y recuperarse de errores:** Se proporcionan mensajes de error claros y sugerencias para resolver problemas comunes.
10. **Ayuda y documentación:** Se incluye una sección de ayuda accesible que explica cómo utilizar las principales funcionalidades de la aplicación.

7. Implementación

Punto a revisar

- Juanmi

En este capítulo se describe la implementación del proyecto, así como detalles técnicos y decisiones tomadas durante el desarrollo. Se divide en varios sprints, cada uno con sus propias tareas y objetivos.

7.1. Sprint 0

En este sprint no se comienza el desarrollo del proyecto, sino que, como se menciona en la sección [4](#) y [5](#), se realiza una investigación sobre las tecnologías a utilizar, y se detallan los requerimientos del sistema, y la arquitectura a implementar.

Al usar Java (versión 21) como lenguaje de programación y Spring Boot (versión 3.4.4) como framework de desarrollo, se sigue un patrón similar para el desarrollo de los servicios, ya que todos comparten componentes y directorios similares.

7.1.1. Estructura general de los servicios

El código fuente de los servicios se organiza en paquetes, siguiendo una estructura común para todos los servicios. Esta estructura incluye:

- **config:** Contiene la configuración del servicio, como la configuración de seguridad, bases de datos, etc.
- **controller:** Contiene los controladores REST que manejan las peticiones HTTP.
- **dto:** Contiene los objetos de transferencia de datos (DTO) utilizados para la comunicación entre el cliente y el servidor.
- **model:** Contiene las entidades del dominio del servicio.
- **repository:** Contiene las interfaces de repositorio que extienden de JPA para la persistencia de datos.
- **service:** Contiene la lógica de negocio del servicio.
- **mapper:** Contiene los mapeadores para convertir entre entidades y DTOs.
- **Application.java:** Clase principal que arranca el servicio.

De esta forma, se consigue una estructura clara y coherente para el desarrollo de los servicios, facilitando la comprensión y el mantenimiento del código.

Además para la configuración de estos, se utiliza un archivo de propiedades (application.properties) que permite definir las propiedades específicas del servicio, como la conexión a la base de datos, el puerto en el que se ejecuta el servicio, etc.

7.2. Sprint 1

Este primer sprint se comienza con cierta incertidumbre al no haber tenido todavía la reunión con Alberto Guillén Perales, el director del CEPRUD, por lo que no se sabe a qué información se va a tener acceso, y por tanto cómo se van a implementar ciertas funcionalidades.

Sin embargo, ya que el proceso para poder hacer uso del sistema de autenticación de la UGR parece ser largo y requiere de una serie de permisos y pasos previos [68], se decide en este punto comenzar a implementar en el backend todas las funcionalidades relacionadas con la gestión de usuarios, y autenticación basadas en las credenciales de la UGR.

Para conseguir esto se implementan los servicios “User Service”, “Auth Service”, “Mail Service” y el “API Gateway”.

7.2.1. User Service

Este servicio se plantea como el encargado de gestionar los usuarios del sistema, permitiendo registrar, consultar, actualizar y eliminar usuarios. Además, se encarga de la gestión de roles y permisos, así como de la codificación de contraseñas.

Las contraseñas se almacenan de forma segura mediante codificación (por ejemplo, BCrypt), y no en texto plano.

Los usuarios pueden poseer los roles de ROLE_INACTIVE, ROLE_STUDENT, ROLE_TEACHER, o ROLE_ADMIN, y estos nos permiten controlar el acceso a diferentes funcionalidades del sistema.

Aunque este servicio no es el encargado de la autenticación, sirve de soporte para el servicio de autenticación, proporcionando la información de los usuarios y sus roles.

Integración con otros microservicios

- Es consultado por otros servicios para validar identidad o permisos de los usuarios.
- Envía mensajes mediante RabbitMQ para notificar el registro de usuarios y/o cambios en sus credenciales.

Diagrama de clases

Este es el primer servicio implementado, y se sigue una estructura similar en los demás servicios para mantener la coherencia en el proyecto. La arquitectura de los servicios en la **Arquitectura limpia o hexagonal**, que se basa en la separación de responsabilidades y la independencia de las capas, se refleja en el diagrama de clases del servicio User Service, que se muestra en la figura 7.1.

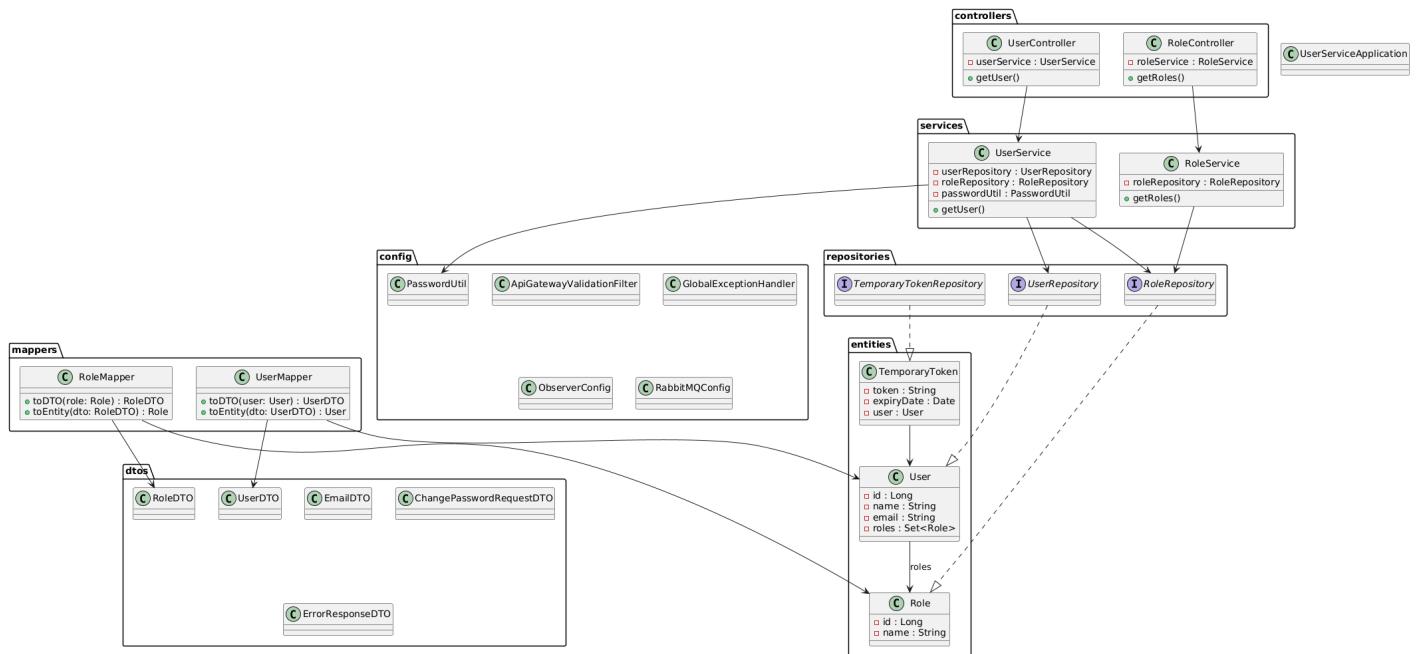


Figura 7.1: Diagrama de clases del servicio User Service realizado con PlantUML

Este diagrama representa las principales clases del servicio, incluyendo los controladores, servicios, repositorios y entidades. Cada clase tiene una responsabilidad clara y se comunica con otras clases a través de interfaces, lo que permite una fácil extensibilidad y mantenimiento del código.

Interacción entre componentes

1. Controladores (controllers)

- **UserController y RoleController:** Son clases que exponen endpoints HTTP. Actúan como punto de entrada de las solicitudes del cliente.
- **Interacción:** Invocan métodos en los servicios correspondientes (UserService, RoleService) para obtener datos o ejecutar lógica de negocio.

2. Servicios (services)

- **UserService:** Contiene la lógica de negocio relacionada con usuarios.

- Depende de UserRepository y RoleRepository para acceder a la base de datos.
- Usa PasswordUtil para tareas relacionadas con contraseñas (encriptación, validación, etc).
- **RoleService:** Contiene lógica de negocio asociada a roles.
 - Se comunica con RoleRepository.

3. Repositorios (repositories)

- **UserRepository, RoleRepository, TemporaryTokenRepository:** Son interfaces que definen operaciones CRUD (Create, Read, Update, Delete) sobre las entidades.
- **Interacción:** Son utilizados por los servicios para obtener o almacenar datos en la base de datos.

4. Entidades (entities)

- **User:** Representa un usuario del sistema. Tiene atributos como id, name, email, y una colección de roles.
- **Role:** Representa un rol o permiso dentro del sistema. Se relaciona con los usuarios.
- **TemporaryToken:** Usado para funcionalidades temporales como recuperación de contraseña. Contiene un token, una fecha de expiración y referencia al User.
- **Interacción:** Estas entidades son gestionadas por los repositorios y utilizadas en la lógica de negocio de los servicios.

5. Mapeadores (mappers)

- **UserMapper y RoleMapper:** Se encargan de convertir entidades (User, Role) a sus correspondientes objetos de transferencia de datos (UserDTO, RoleDTO) y viceversa.
- **Interacción:** Usados por los servicios para traducir datos entre capas internas y externas.

6. DTOs (dtos)

- **UserDTO, RoleDTO, EmailDTO, ChangePasswordRequestDTO, ErrorResponseDTO:** Representan estructuras de datos que se usan para enviar y recibir información a través de la API.

- **Interacción:** Son utilizados por los controladores y servicios para comunicar información de forma segura y controlada, evitando exponer entidades directamente.

7. Configuración (config)

- **PasswordUtil:** Clase de utilidad para el manejo de contraseñas.
- **ApiGatewayValidationFilter:** Filtro que valida solicitudes entrantes desde el API Gateway.
- **GlobalExceptionHandler:** Captura y maneja excepciones globalmente.
- **RabbitMQConfig, ObserverConfig:** Configuración para mensajería y observadores (por ejemplo, eventos asincrónicos).
- **Interacción:** Algunas de estas clases son inyectadas en servicios (PasswordUtil), otras se ejecutan automáticamente (GlobalExceptionHandler).

8. Flujo de Interacción General

1. El cliente realiza una solicitud a través del UserController o RoleController.
2. El controlador llama al servicio correspondiente (UserService o RoleService).
3. El servicio accede a los repositorios para obtener datos desde las entidades.
4. Se usan los mapeadores para convertir entidades en DTOs.
5. El DTO es devuelto al cliente a través del controlador.

Aunque en cada servicio se implementan diferentes funcionalidades, la estructura y la interacción entre los componentes siguen un patrón similar, lo que facilita la comprensión del código. Para ilustrar de manera más simple lo detallado anteriormente, se muestra la figura 7.2, que representa la comunicación general entre los componentes del backend.

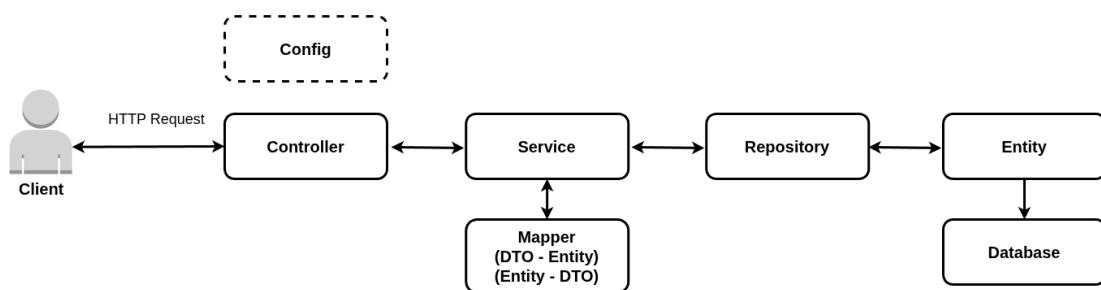


Figura 7.2: Comunicación general entre los componentes del backend

7.2.2. Mail Service

El servicio de correo electrónico se encarga de enviar correos electrónicos a los usuarios del sistema. Este servicio es utilizado por otros servicios para enviar notificaciones, como la confirmación de registro, recuperación de contraseña, etc.

Su desarrollo combina tres tecnologías clave: **RabbitMQ**, **JavaMail** y **Thymeleaf**, permitiendo una integración fluida entre mensajería asíncrona y generación dinámica de contenido de correo.

El proceso comienza cuando otro componente del sistema necesita enviar un correo electrónico. En lugar de enviarlo directamente, publica un mensaje en una cola gestionada por RabbitMQ. Este servicio actúa como consumidor de esos mensajes mediante el uso de listeners, que reciben el contenido del mensaje en tiempo real. Esta estrategia permite desacoplar el proceso de envío de correo del resto de la lógica de negocio, mejorando la escalabilidad y la resiliencia del sistema ante posibles errores.

Una vez recibido el mensaje, el servicio extrae los datos necesarios, como la dirección de correo del destinatario, el asunto, y las variables que se inyectarán en el contenido. Para la composición del cuerpo del correo, se utiliza Thymeleaf, un motor de plantillas que permite construir correos HTML personalizados. Gracias a Thymeleaf, se pueden generar correos enriquecidos visualmente, dinámicos y adaptados al contexto de cada notificación, manteniendo una presentación clara y profesional.

Después de generar el contenido del correo, se utiliza la biblioteca JavaMail para su envío. JavaMail proporciona una API robusta y configurable que permite construir mensajes MIME, manejar los encabezados, adjuntos y establecer la conexión con el servidor SMTP. En este caso, se ha configurado el servicio para utilizar **Gmail como proveedor de correo electrónico**, lo cual requiere definir parámetros como el host SMTP de Gmail (`smtp.gmail.com`), el puerto correspondiente, y habilitar la autenticación y conexión segura mediante TLS. Esta configuración se especifica en los archivos de propiedades del servicio, lo que facilita su adaptación a distintos entornos de despliegue.

En conjunto, este microservicio representa una solución elegante y modular para el envío de correos [7.3](#) en arquitecturas basadas en microservicios. Aprovecha la comunicación asíncrona de RabbitMQ, la potencia expresiva de las plantillas Thymeleaf, y la fiabilidad de JavaMail para lograr un sistema de notificaciones por correo eficaz, personalizable y mantenible.



Figura 7.3: Correo de activación de cuenta

7.2.3. Auth Service

El servicio de autenticación es el encargado de gestionar la autenticación de los usuarios del sistema. Este servicio se encarga de validar las credenciales de los usuarios y generar tokens JWT (JSON Web Tokens) para autenticar las solicitudes a otros servicios.

Su lógica reside en la clase AuthService. La operación principal, `authenticate`, recibe las credenciales enviadas por el cliente y devuelve un par de tokens *JSON Web Token* (JWT): un *access token* de corta duración y un *refresh token* de duración más amplia.

El proceso comienza verificando que la dirección de correo pertenezca a la Universidad de Granada; de lo contrario, se aborta la autenticación. Despues, mediante un WebClient reactivo, el servicio consulta al microservicio de usuarios (`user-service`) para recuperar el UserDTO correspondiente. Si el usuario existe, está activo y la contraseña coincide con el *hash* almacenado (comprobación delegada a `PasswordUtil`), se generan ambos tokens.

Estructura del JWT

Un JWT es una cadena en tres partes "`header.payload.signature`", codificadas en Base64Url. En este servicio se firma con el algoritmo simétrico HS256, usando la clave secreta `JWT_SECRET`:

```

Key key = Keys.hmacShaKeyFor(SECRET_KEY.getBytes(StandardCharsets.UTF_8));
String jwt = Jwts.builder()
    .setHeaderParam("typ", "JWT")
    .setSubject(id)           // Identificador del usuario
    .claim("role", role)     // Rol de la aplicación
    .setIssuedAt(new Date(now))
    .setExpiration(new Date(now + ttl))
    .signWith(key, SignatureAlgorithm.HS256)
    .compact();

```

La cabecera (header) indica el tipo de token y el algoritmo de firma. El cuerpo (payload) almacena el identificador del usuario (sub), su rol y las marcas temporal de emisión y caducidad (iat, exp). La firma garantiza la integridad: si cualquier byte del token cambia, la verificación falla.

Duración y refresco

El *access token* caduca en 24 horas (86,400,000 ms), su vida corta limita el riesgo ante robo. El *refresh token* dura una semana y sólo sirve para obtener nuevos *access tokens*. La operación *refresh* valida la firma y la vigencia del *refresh token*; si supera la comprobación, crea un nuevo *access token* manteniendo el *refresh token* original mientras no haya expirado. El servidor no almacena sesiones, por lo que el mecanismo es *stateless* y escalable: basta con compartir la misma `JWT_SECRET` entre las instancias.

Flujo de autenticación

El diseño evita guardar estado en la base de datos y permite balancear las peticiones entre múltiples réplicas sin afinidad de sesión.

El flujo de autenticación y generación de tokens JWT se ilustra en la figura 7.4, donde se muestra cómo el servicio interactúa con el user-service para validar las credenciales y generar los tokens necesarios.

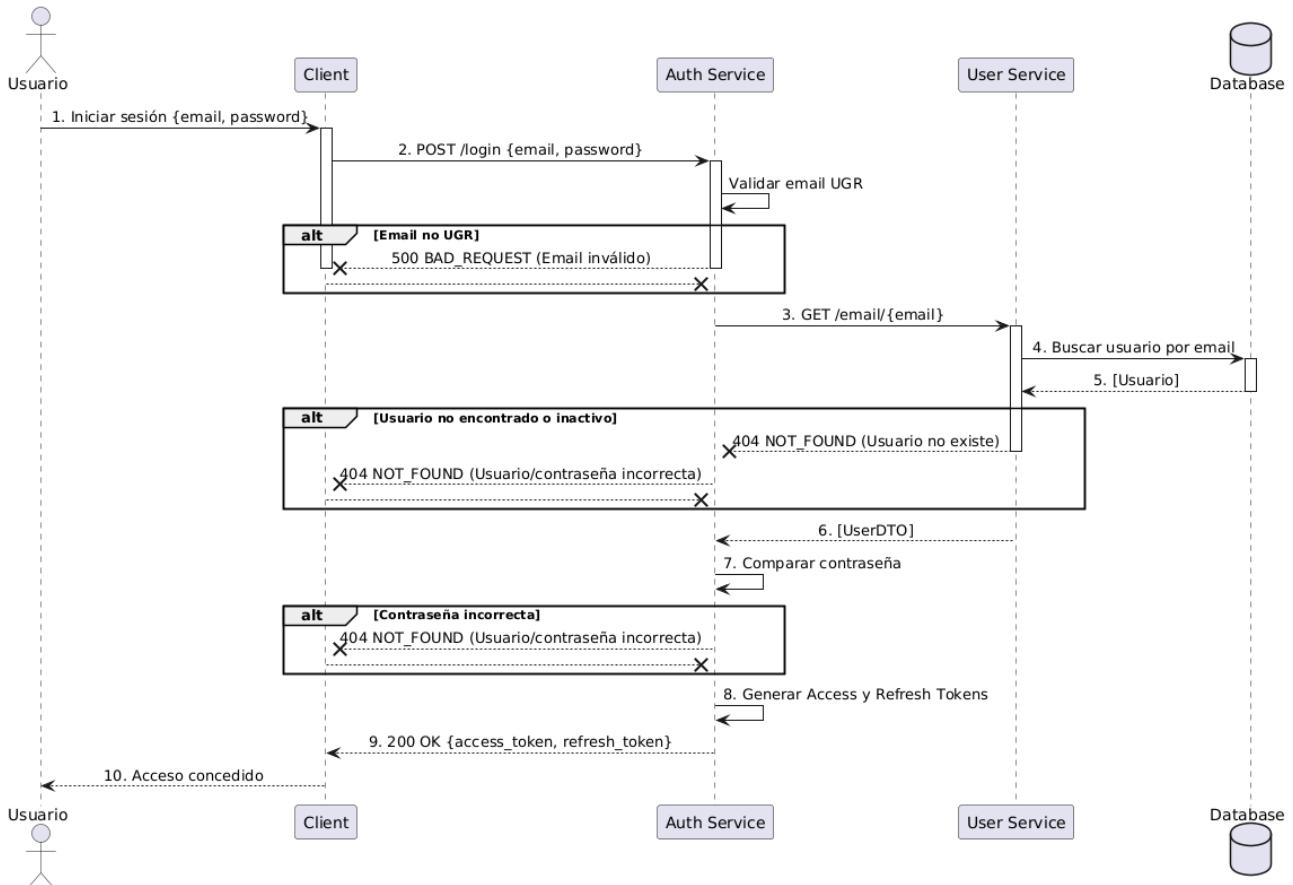


Figura 7.4: Flujo de autenticación y generación de tokens JWT

7.2.4. API Gateway

El api-gateway actúa como el punto de entrada centralizado para todas las peticiones del sistema basado en microservicios. Está construido utilizando **Spring Cloud Gateway**, una solución reactiva que permite enrutar solicitudes a los distintos servicios internos, como user-service o auth-service. Su función no se limita al enrutamiento, sino que también gestiona aspectos transversales como la seguridad, la autenticación mediante JWT, y la validación de rutas públicas y protegidas.

Spring Security en el Gateway

Spring Security es un framework de seguridad potente y altamente configurable, que se utiliza para proteger tanto aplicaciones monolíticas como distribuidas. En el contexto del api-gateway, se emplea para interceptar las peticiones entrantes y aplicar mecanismos de autenticación antes de permitir el acceso a los servicios internos. Aquí no se gestionan sesiones, sino que se trabaja con tokens JWT (JSON Web Tokens), que son validados en cada petición.

El archivo `SecurityConfig` define la configuración de seguridad. En este componente se especifican:

- Los filtros personalizados que deben ejecutarse antes del procesamiento de cada solicitud.
- Las rutas que deben ser públicas (por ejemplo, el login o el registro).
- Que el sistema funciona sin sesiones, usando una política stateless.

Filtros personalizados

Spring Security permite encadenar filtros que procesan las solicitudes antes de llegar a los controladores. En el gateway, se implementan dos filtros principales:

JwtAuthenticationFilter Este filtro se encarga de interceptar todas las solicitudes entrantes y validar que el JWT (token de acceso) esté presente y sea válido. Para ello, extrae el token del encabezado `Authorization` y realiza una verificación utilizando una clave secreta compartida. En caso de éxito, el filtro añade los atributos del usuario autenticado al encabezado de la petición, de modo que los servicios internos puedan conocer la identidad y rol del usuario. Si el token no es válido o está ausente, la solicitud se rechaza con un error HTTP 401 Unauthorized.

PathPrefixFilter Elimina el prefijo `/calendarugr/v1` de la ruta de cada petición entrante en el API Gateway. Si la URL comienza con ese prefijo, lo elimina y reescribe la ruta antes de pasar la petición al siguiente filtro o servicio interno. Así, los microservicios reciben rutas limpias y sin el prefijo de versión, facilitando el enrutamiento interno. Además, el filtro tiene la máxima prioridad para ejecutarse antes que otros filtros de seguridad.

Configuración de rutas seguras

En el archivo de configuración `SecurityConfig`, se define explícitamente qué rutas deben estar protegidas. Por ejemplo, aquellas que comienzan con `/user` pueden requerir un token JWT válido, mientras que rutas como `/auth/login` son públicas. Esta distinción permite implementar un control de acceso granular a través del propio gateway, centralizando así la lógica de seguridad.

Además, se utiliza una política de seguridad sin estado (`SecurityContextHolder.MODE_INHERITABLETHREADLOCAL`) y se desactivan los mecanismos tradicionales como CSRF y sesiones HTTP, ya que todo el control de identidad se realiza mediante tokens.

En resumen, el api-gateway cumple un rol esencial en la arquitectura del sistema, actuando no solo como un proxy inverso, sino también como un punto de control de acceso. Gracias a Spring Security y a los filtros como `JwtAuthenticationFilter`, se logra una protección robusta y centralizada,

adecuada para entornos distribuidos donde cada microservicio es independiente pero requiere información sobre la identidad del usuario que realiza la petición.

7.2.5. RabbitMQ

El servicio de mensajería RabbitMQ se integra en el sistema para facilitar la comunicación asíncrona entre los diferentes microservicios. Este enfoque permite desacoplar los servicios, mejorar la escalabilidad y manejar picos de carga sin afectar la disponibilidad del sistema.

Configuración de RabbitMQ

RabbitMQ se configura en el archivo `application.properties` de cada servicio que lo utiliza. Se especifican parámetros como el host del servidor RabbitMQ, el puerto, el nombre de usuario y la contraseña.

Funcionamiento de RabbitMQ

RabbitMQ es un sistema de mensajería basado en el modelo *message broker*, que permite la comunicación asíncrona entre distintos servicios. Se apoya en el protocolo AMQP (*Advanced Message Queuing Protocol*), un protocolo binario de capa de aplicación diseñado para asegurar la interoperabilidad entre sistemas, fiabilidad en la entrega de mensajes y soporte para confirmaciones, encolado y reintentos.

En una arquitectura de microservicios, RabbitMQ permite desacoplar componentes mediante el intercambio de mensajes a través de colas. Un servicio puede publicar un mensaje sin necesidad de que el receptor esté disponible en ese momento. Esta estrategia mejora la tolerancia a fallos y la escalabilidad del sistema.

Componentes principales

RabbitMQ se basa en cuatro componentes clave:

- **Productores (Producers):** servicios que envían mensajes. En este caso, el `user-service` actúa como productor.
- **Intercambios (Exchanges):** puntos intermedios que reciben mensajes y los redirigen a una o más colas basándose en reglas de enrutamiento. Existen varios tipos, como `direct`, `fanout`, `topic` y `headers`.
- **Colas (Queues):** estructuras donde los mensajes son almacenados hasta que un consumidor los procesa.
- **Consumidores (Consumers):** servicios que reciben y procesan los mensajes. En este caso, el `mail-service` es el consumidor.

Configuración en user-service

El servicio user-service actúa únicamente como productor. Su configuración en RabbitMQConfig define un DirectExchange llamado mail_exchange, y una clave de enrutamiento registering_routing_key. Esto implica que todos los mensajes enviados desde este servicio se publican a dicho exchange con esa clave.

```
DirectExchange exchange() {  
    return new DirectExchange("mail_exchange");  
}
```

Este fragmento configura el punto de entrada de los mensajes que serán enviados al sistema de mensajería.

Configuración en mail-service

El servicio mail-service funciona exclusivamente como consumidor de mensajes [7.5](#). Su configuración es más extensa, ya que debe definir tanto las colas como las asociaciones con el intercambio, además de políticas de reintento y manejo de errores.

Se definen dos colas principales:

- registering_queue, para correos de confirmación de registro.
- notification_queue, para notificaciones genéricas.

Ambas están enlazadas al mail_exchange mediante sus respectivas claves de enrutamiento. Esta asociación se realiza mediante Binding.

También se especifica un convertidor de mensajes para que RabbitMQ utilice JSON como formato de serialización, mediante Jackson2JsonMessageConverter, permitiendo la compatibilidad con objetos Java.

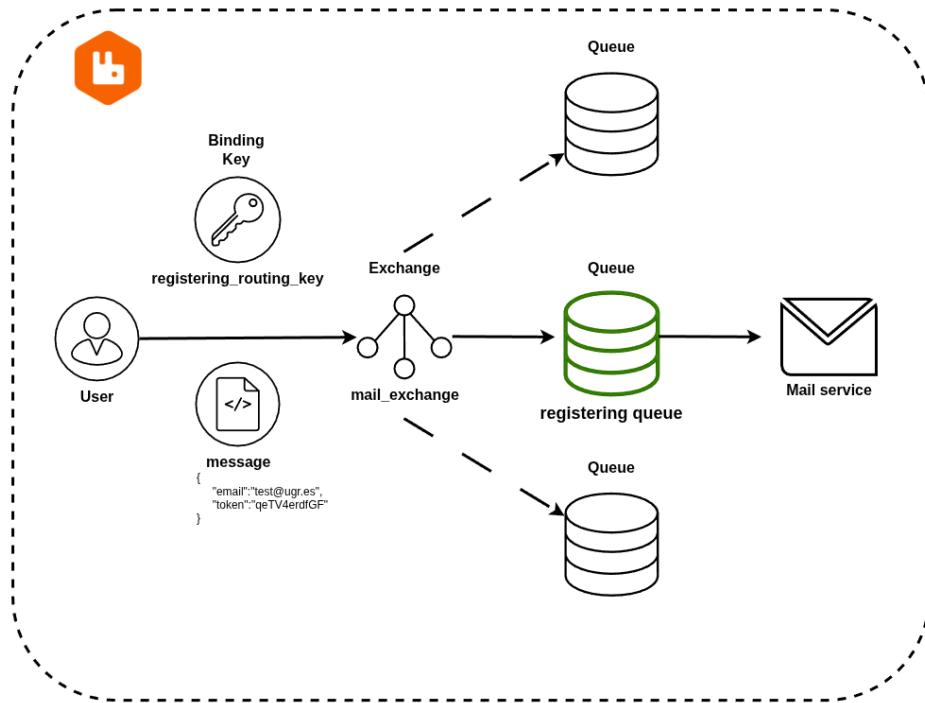


Figura 7.5: Configuración de RabbitMQ en el servicio mail-service

Reintentos automáticos y Dead Letter Queues

Una característica importante de la configuración es la gestión de reintentos automáticos y colas de mensajes muertos (DLQ).

Los reintentos se configuran a través de un `RetryInterceptor`:

```
.maxAttempts(3)
.backOffOptions(1000, 2.0, 50000)
```

Esto indica que, si el procesamiento de un mensaje falla, se reintentará hasta tres veces, con un intervalo inicial de 1 segundo y un multiplicador exponencial. Si después de los reintentos no se logra procesar el mensaje, este se considera fallido.

En lugar de eliminar estos mensajes fallidos, se redirigen a una **Dead Letter Queue**, en este caso `mail_dead_letter_queue`, asociada a su propio Dead Letter Exchange (`mail_dead_letter_exchange`). Esto permite su análisis posterior, ya sea manual o mediante herramientas automáticas.

Resumen funcional

En este sistema, el `user-service` publica eventos de registro de usuarios a RabbitMQ. El `mail-service` escucha la cola correspondiente y envía los correos pertinentes. En caso de fallos temporales, se aplican políticas de reintentos. Si el fallo persiste, el mensaje es redirigido a una DLQ, asegurando que ningún mensaje se pierde sin ser registrado.

Esta estrategia desacopla los servicios, mejora la resiliencia del sistema, y permite manejar errores de forma controlada sin interrumpir el flujo principal de la aplicación.

7.2.6. Primer hito del backend

El primer hito del backend se alcanza con la implementación de los servicios de autenticación y gestión de usuarios, junto con el servicio de correo electrónico. Estos servicios permiten registrar usuarios, autenticar sus credenciales y enviar correos electrónicos de confirmación. Además, se ha implementado el API Gateway para centralizar las peticiones y gestionar la seguridad mediante JWT.

El siguiente paso antes de comenzar el siguiente sprint, y tras la reunión con Alberto Guillén Perales, es investigar cómo se pueden obtener los horarios de las asignaturas de la UGR.

Para ello se investiga el sistema de scrapping, y se implementa un primer prototipo que permite obtener los horarios de las asignaturas de la UGR. Este prototipo se implementa en una primera aproximación del servicio schedule-consumer-service, que se encargará de consumir los horarios de las asignaturas y generar eventos a partir de ellos.

La tecnología seleccionada para realizar la tarea de scrapping es Jsoup, una biblioteca de Java que permite extraer y manipular datos de documentos HTML. Jsoup proporciona una API sencilla para navegar por el DOM, seleccionar elementos y extraer información, lo que facilita la tarea de scrapping.

Scrapping con Jsoup

El sistema realiza un proceso de extracción automatizado desde un portal académico para recopilar información sobre titulaciones, asignaturas, profesorado, grupos y horarios, utilizando técnicas de *web scraping* con la biblioteca Jsoup y almacenamiento en base de datos mediante repositorios JPA.

La primera fase del proceso consiste en acceder al portal principal de ramas de conocimiento, desde donde se extraen los enlaces hacia las distintas titulaciones. A partir de cada enlace, se accede a la página específica de la titulación y se recogen sus datos básicos. Si la titulación no está ya registrada en la base de datos, se crea una nueva entrada con su nombre, URL y rama correspondiente.

Posteriormente, se recorre cada titulación almacenada y se accede a su página detallada para obtener dos tipos de información. Por un lado, si no se ha registrado la facultad correspondiente, se extrae de una tabla presente en el sitio web. Por otro lado, se recopilan los enlaces de todas las asignaturas que pertenecen a dicha titulación. Estas asignaturas se crean en la base de datos sólo si no existen previamente, y se asocian con la titulación correspondiente.

La última fase del proceso accede a cada una de las asignaturas almacenadas

para extraer información detallada. En primer lugar, se recoge la información general de la asignatura como el curso académico, el año, el semestre, el tipo y el departamento responsable. Se aplican medidas de control como el recorte del texto del departamento si excede una longitud determinada.

Después, se identifica al profesorado vinculado a la asignatura y a sus respectivos grupos. Si el grupo ya existe, se actualiza su lista de profesores. Si no existe, se crea y se almacena junto con el nombre del profesor. Se contempla la posibilidad de que algunos grupos no aparezcan explícitamente en la web, en cuyo caso se generan automáticamente.

Finalmente, se extrae el horario de clases para cada grupo, identificando el día de la semana, aula, fechas de inicio y fin, y horas de inicio y fin. Esta información se transforma en objetos de clase que se asocian con el grupo correspondiente. Si no se detecta una clase idéntica en la base de datos, se crea una nueva entrada para ella.

Este enfoque [7.6](#) permite consolidar una base de datos académica completa, dinámica y precisa, ideal para ser explotada por nuestro sistema.

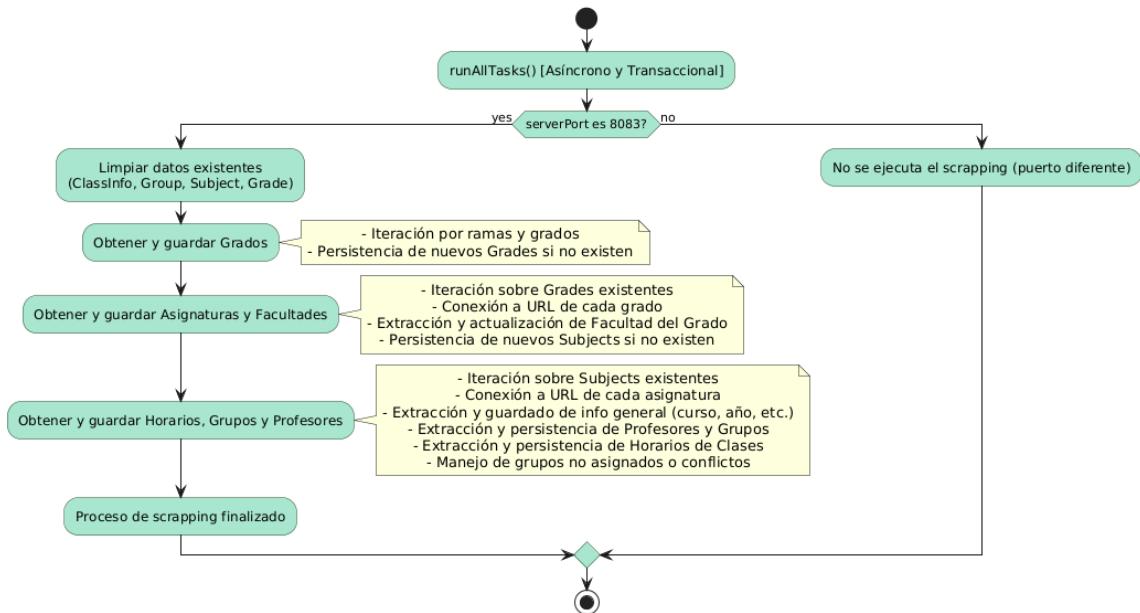


Figura 7.6: Proceso de scrapping de asignaturas y horarios

7.3. Sprint 2

En el comienzo de este sprint se continúa afinando y completando el script de scrapping, y revisando que se recoja información de todos los grados y asignaturas de la UGR. Esto se consigue realizando una labor de análisis exhaustivo de las diferentes páginas y secciones que conforman el portal de “grados UGR” mencionado con anterioridad.

Schedule Consumer Service

Paralelo a este desarrollo, se comienza el desarrollo del servicio `schedule-consumer-service`. Este es el encargado de, en primer lugar, hacer uso del scrapping para obtener los horarios de las asignaturas y almacenar la información en la base de datos, y en segundo lugar, de exponer endpoints relacionados con la consulta de estos datos. Uno de los objetivos de la realización se este servicio es poseer endpoints públicos para poder ser consumidos desde otros sistemas externos a nuestro frontend “TempusUGR”.

Una vez hecho el scrapping, y la información en base de datos (integrando el script de scrapping con el resto de componentes de Spring como lo son “Repositories”, “Controllers”, etc), se necesita una manera de actualización de la información, puesto que es una de las premisas del sistema, mantener un acceso personalizado y actualizado del calendario académico.

Para conseguir esto, y puesto que iterar y comprobar cambios sobre las más de 50.000 clases distintas impartidas en grupos de asignatura de la UGR es una tarea costosa y compleja, se decide implementar un “Cron job” que se encargue de realizar el scrapping de forma periódica, y así mantener la información actualizada. Este cron job se implementa en el servicio `schedule-consumer-service` y se ejecuta todos los días a las 00:00 horas, de forma que se actualizan los horarios de las asignaturas y se eliminan aquellos que ya no existen.

Spring Boot proporciona una forma sencilla de implementar cron jobs mediante la anotación `@Scheduled`. Esta anotación permite definir un método que se ejecutará periódicamente según una expresión cron. En este caso, se ha configurado para que el método de actualización de horarios se ejecute diariamente a medianoche. Además de la anotación `@Scheduled`, se utiliza la anotación `@EnableScheduling` en la clase de configuración principal del servicio para habilitar el soporte de programación de tareas, y la anotación `@Async` para permitir la ejecución asíncrona de los métodos, lo que mejora la eficiencia y la capacidad de respuesta del servicio. Para evitar un fallo del funcionamiento del servicio mientras se realiza una sustitución de los datos (Primero se borran los horarios antiguos, y luego se añaden los nuevos), se hace uso también de la anotación `@Transactional`, proveniente de “Spring Transaction”, en el método que realiza la actualización de horarios. Esto asegura que todas las operaciones de base de datos se realicen dentro de una transacción, lo que garantiza la consistencia de los datos y evita problemas de concurrencia. Así los usuarios pueden seguir consultando la información de horarios sin interrupciones, incluso durante el proceso de actualización.

Una vez implementado el servicio se consigue el servicio troncal del backend, que permite obtener los horarios de las asignaturas de la UGR, y exponerlos a través de una API REST. Este servicio se convierte en el núcleo del sistema, ya que proporciona la información necesaria para el objetivo principal del proyecto: ofrecer un calendario académico personalizado y actualizado para los estudiantes de la UGR.

Academic Subscription Service

Así se comienza con el desarrollo del servicio academic-subscription-service, que se encargará de gestionar las suscripciones a los grupos de asignaturas de cualquier grado de la UGR.

Las suscripciones se componen del identificador del usuario, la facultad, el grado, la asignatura y su grupo, puesto que de otra manera no se podría identificar de forma única un grupo de asignatura, ya que puede haber grupos con el mismo nombre en diferentes grados o facultades. Con esta información se hacen peticiones al servicio schedule-consumer-service para obtener los horarios de las asignaturas [7.7](#) y generar un calendario personalizado para el usuario con sus clases oficiales.

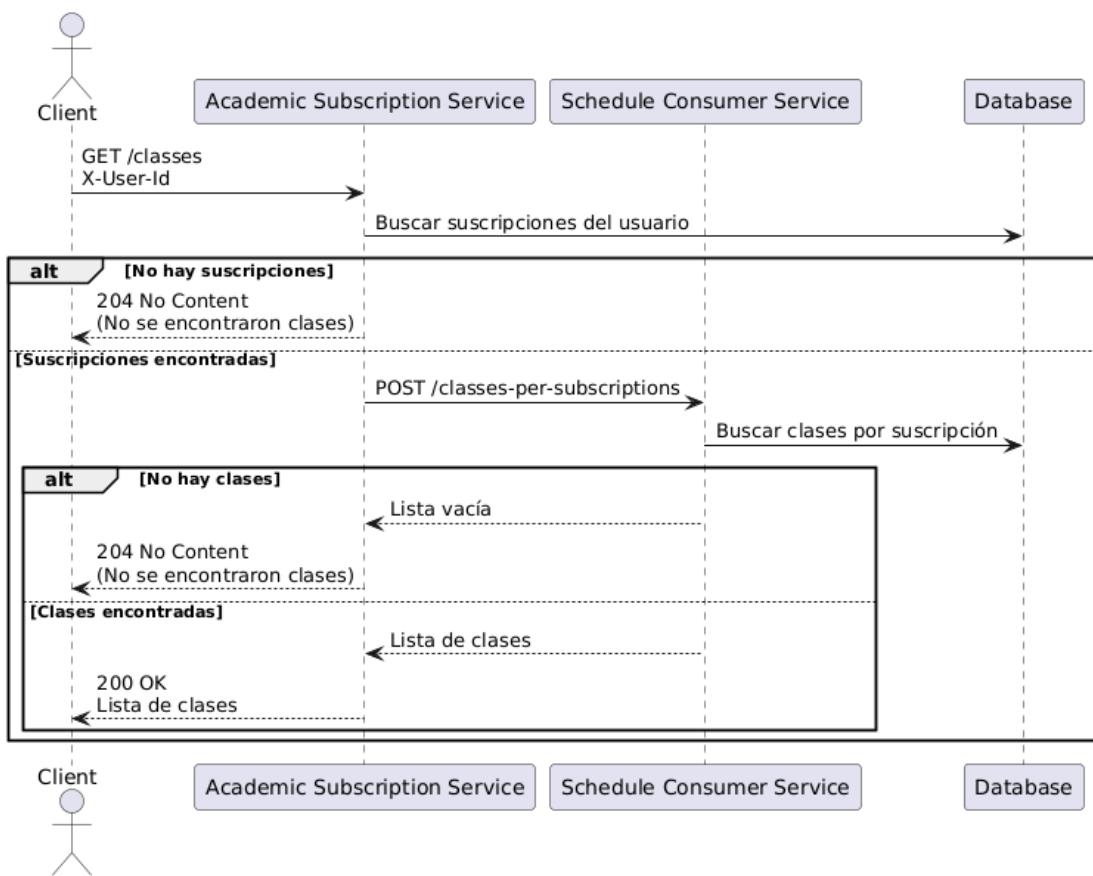


Figura 7.7: Servicio de suscripciones académicas

Una vez se consigue esta información en formato “JSON”, se implementa la lógica para generar un archivo .ics[\[69\]](#) que pueda ser importado en aplicaciones de calendario como Google Calendar, Outlook, etc. Para ello se utiliza la biblioteca ical4j, que permite crear y manipular archivos de calendario en formato iCalendar (.ics).

Para crear esta primera versión de “.ics” se hace uso de los eventos recurrentes (RRULE) de iCalendar, que permiten definir patrones de repetición para eventos. Esto

es especialmente útil para las clases que se repiten semanalmente, ya que permite definir una sola entrada en el calendario que se repetirá automáticamente en las fechas correspondientes.

Toda la información en este servicio se almacena en una base de datos no relacional, en este caso MongoDB, que permite una mayor flexibilidad y escalabilidad para almacenar los datos de las suscripciones y los horarios de las asignaturas.

Eureka Discovery Server

A esta altura del proyecto en la que ya se tienen desarrollados varios servicios, se decide implementar un servidor de descubrimiento de servicios utilizando **Eureka**, que es parte del ecosistema de Spring Cloud. Eureka permite a los servicios registrarse y descubrir otros servicios en el sistema, facilitando la comunicación entre ellos sin necesidad de conocer sus direcciones IP o puertos específicos.

Para conseguir esto se crea un nuevo proyecto de Spring Boot que actúa como el servidor Eureka. Este servidor se configura para permitir que otros servicios se registren y descubran entre sí. Cada servicio que se desea registrar en Eureka debe incluir la dependencia de `spring-cloud-starter-netflix-eureka-client` y configurarse adecuadamente en su archivo `application.properties`.

Esto nos permite los siguientes avances:

- **Balanceo de carga:** Al registrar los servicios en Eureka, se pueden utilizar balanceadores de carga de manera dinámica, sin necesidad de configurar manualmente las direcciones IP y puertos de cada servicio. Eureka proporciona una lista de instancias disponibles, lo que permite al cliente elegir una instancia para enviar la solicitud.
- **Alta disponibilidad:** Si un servicio falla, Eureka permite que otros servicios sigan funcionando sin interrupciones, ya que pueden redirigir las peticiones a otras instancias disponibles.
- **Configuración centralizada:** Permite gestionar la configuración de los servicios desde un único punto, facilitando el mantenimiento y la actualización del sistema.
- **Facilidad de desarrollo:** Al utilizar Eureka, los desarrolladores pueden centrarse en la lógica de negocio de sus servicios sin preocuparse por la gestión de direcciones IP y puertos, ya que Eureka se encarga de ello. Es decir, las llamadas HTTP pasan de tener un formato similar a este `http://localhost:8081/calendarugr/v1/user-service/users` a `http://user-service/users`, donde `user-service` es el nombre del servicio registrado en Eureka.

De esta manera podemos tener un control de los servicios que se están ejecutando en el sistema [7.8](#), y facilitar la comunicación entre ellos. Además, se puede escalar horizontalmente los servicios, añadiendo más instancias de un mismo servicio sin necesidad de modificar el código del cliente.

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
ACADEMIC-SUBSCRIPTION-SERVICE	n/a (1)	(1)	UP (1) - b1cd8d1f2d53:academic-subscription-service:8084
API-GATEWAY	n/a (1)	(1)	UP (1) - 6b7177048edc:api-gateway:8090
AUTH-SERVICE	n/a (1)	(1)	UP (1) - 225642bc4e7c:auth-service:9000
MAIL-SERVICE	n/a (1)	(1)	UP (1) - a8e8fec07625:mail-service:8082
SCHEDULE-CONSUMER-SERVICE	n/a (1)	(1)	UP (1) - 2c339aa98c44:schedule-consumer-service:8083
USER-SERVICE	n/a (1)	(1)	UP (1) - 00cd8e7d600d:user-service:8081

Figura 7.8: Servidor de descubrimiento de servicios con Eureka

7.4. Sprint 3

El comienzo de estas tres semanas estuvo centrado en la finalización del servicio academic-subscription-service, que se encarga de gestionar las suscripciones a los grupos de asignaturas de cualquier grado de la UGR.

Para conseguir esto, se implementa la gestión de eventos a nivel de grupo (eventos que son relevantes para aquellas personas suscritas a ese grupo de asignatura), y eventos a nivel de facultad (eventos que son relevantes para todas las personas suscritas a cualquier grupo de asignatura de esa facultad). Estos eventos se almacenan en la base de datos del servicio academic-subscription-service, en una tabla diferente a las suscripciones.

Para poder crear estos eventos, el usuario debe ser tanto profesor como administrador, y para poder crear eventos a nivel de facultad, el usuario debe ser administrador. Estos eventos se crean a través de endpoints del servicio academic-subscription-service.

Una vez se crean los eventos, estos serán visibles para todos los usuarios suscritos a ese grupo de asignatura o facultad, y se enviarán notificaciones a través del servicio de correo electrónico (mail-service) a todos los usuarios suscritos con notificaciones activadas. Estas notificaciones se envían mediante RabbitMQ, como se ha explicado anteriormente, y se generan correos electrónicos personalizados utilizando plantillas Thymeleaf en el servicio de mail.

Una vez creados los eventos, e implementada la posibilidad de poder obtener la información de estos y clases oficiales, se adaptó la creación de los archivos .ics para incluir tanto las clases oficiales como los eventos de grupo y facultad. Estos al contrario que las clases oficiales, no son recurrentes, por lo que se crean como eventos únicos en el calendario. De esta manera, el usuario puede importar un archivo .ics que contenga tanto sus clases oficiales como los eventos de grupo y facultad a su calendario personal, ya sea Google Calendar, Outlook, etc.

Además de esto se implementa la sincronización con Google Calendar. Para conseguir esto se pueden seguir diferentes estrategias:

- **Google Calendar API:** Utilizar la API de Google Calendar para crear eventos

directamente en el calendario del usuario. Esto requiere que el usuario autorice la aplicación a acceder a su calendario.

- **Exportación de archivos ICS:** Generar un archivo .ics que el usuario pueda descargar e importar manualmente en su Google Calendar.
- **Url de sincronización:** Proporcionar una URL de suscripción a un calendario que Google Calendar pueda utilizar para sincronizar automáticamente los eventos y clases oficiales.

En este caso, y como el sistema ya requiere el correo institucional al usuario (con el de google no podemos descifrar si es estudiante o docente), se opta por la tercera opción, que es la más sencilla y permite al usuario sincronizar su calendario de forma automática. Para ello se genera una URL que el usuario puede añadir a su Google Calendar, y que se actualizará automáticamente con los eventos y clases oficiales del usuario. El acceso a esta URL es público, por lo que cualquier persona que tenga el enlace podrá acceder a los eventos y clases oficiales del usuario. Esto se decide hacer de esta manera para que Google Calendar pueda acceder a la información sin necesidad de autenticación, ya que el usuario ya ha autorizado la aplicación a acceder a su calendario al momento de crear la suscripción.

La fabricación de esta URL requiere la creación de un hash único para cada usuario que además sea “reversible” para identificar la pertenencia del mismo. Para conseguir esto se hizo uso de “AES”, un algoritmo de cifrado simétrico que permite cifrar y descifrar datos de forma segura. Se utiliza una clave secreta para cifrar el identificador del usuario, y se genera un hash único que se utiliza como parte de la URL de sincronización. De esta manera, se garantiza que la URL es única para cada usuario y que solo el usuario puede acceder a su información.

La URL queda con un formato similar a este:
<https://tempus.ugr.es/calendarugr/v1/academic-subscription/calendar/HASH>

Segundo hito del backend

Con la implementación del servicio academic-subscription-service y la sincronización con Google Calendar, se alcanza el segundo hito del backend. Este hito permite a los usuarios suscribirse a grupos de asignaturas, recibir notificaciones de eventos relevantes y sincronizar su calendario personal con las clases oficiales y eventos de su facultad.

Pruebas unitarias y de integración

Se implementan pruebas unitarias para todos los servicios del backend, utilizando **JUnit** (librería de pruebas para Java) y **Mockito** (framework de simulación para pruebas unitarias). Estas pruebas permiten verificar el correcto funcionamiento de los servicios, asegurando que las funcionalidades implementadas cumplen con los requisitos establecidos. Se realizan pruebas tanto a nivel de unidad como de integración, comprobando que los servicios interactúan

correctamente entre sí y con la base de datos.

Por ejemplo se implementan pruebas de integración en el servicio schedule-consumer-service para comprobar que todos los componentes del servicio funcionan correctamente, desde el scrapping de los horarios hasta la generación de los archivos .ics. Estas pruebas permiten detectar posibles errores en la lógica del servicio y asegurar que la información se almacena correctamente en la base de datos.

Algunos ejemplos de test en este servicio son:

- **testGetClassesFromGroup()**: Este test verifica el *endpoint* que devuelve las **clases de un grupo específico** (/classes-from-group). Simula una petición GET con parámetros para el grado, la asignatura y el grupo. Espera una respuesta exitosa (código 200 OK) con contenido JSON que es un *array*, y específicamente que el primer elemento tenga el día “viernes”. También incluye un System.out.println para depuración de la API Key y la respuesta.
- **testValidateExtraClass()**: Este test prueba el *endpoint* de **validación de clases extra** (/extraclass-validation). Envía una petición POST con un objeto ExtraClassDTO que contiene los detalles de una clase. Espera una respuesta exitosa (código 200 OK) con contenido JSON que es una cadena false, lo que indica que la clase extra enviada no es válida según la lógica del *backend*.
- **testGetGrades()**: Este test verifica el *endpoint* que devuelve la **lista de grados** (/grades). Simula una petición GET y espera una respuesta exitosa (código 200 OK) con contenido JSON que es un *array*, y que ese *array* contenga exactamente 6 elementos.
- **testGetSubjectsGroups()**: Este test prueba el *endpoint* que devuelve las **asignaturas y sus grupos asociados** para un grado específico (/subjects-groups). Envía una petición GET con el parámetro grade y espera una respuesta exitosa (código 200 OK) con un *array* JSON. Verifica que el primer elemento del *array* tenga la asignatura “Cálculo” y que su *array* de grupos asociado contenga 22 elementos.
- **testValidateSubscription()**: Este test verifica el *endpoint* de **validación de suscripciones** (/subscription-validation). Realiza dos pruebas POST:
 - La primera envía una SubscriptionDTO **válida** y espera una respuesta true.
 - La segunda envía una SubscriptionDTO **inválida** (con un grupo inexistente “Z”) y espera una respuesta false. Ambas esperan un código 200 OK.
- **testGetClassesFromGroupUnauthorized()**: Este test es una prueba de **seguridad o autorización**. Intenta acceder al mismo *endpoint* /classes-from-group que el primer test, pero utiliza una X-Api-Key **incorrecta**. Espera que la respuesta sea un código 403 Forbidden, indicando que la petición no está autorizada.

Ejemplos de test que comprueban la correcta interoperabilidad entre los servicios:

- **testConflictWithExistingClassMiddle()**: Verifica que el *backend* detecta un **conflicto de horario** cuando una clase extra propuesta se solapa con una clase existente en la base de datos (iniciando y terminando dentro del rango de la clase existente). Espera una respuesta de status 409 Conflict.
- **testBadRequestNoFaculty()**: Prueba la validación de entrada. Envía una petición para crear una clase extra donde el campo facultyName está **vacío**. Espera un status 400 Bad Request, indicando que el servidor ha rechazado la petición debido a datos inválidos o faltantes.
- **testConflictWithExistingClassInit()**: Comprueba que se detecta un **conflicto de horario** cuando la hora de inicio de la clase extra propuesta se solapa con una clase existente, aunque la hora de fin sea anterior al fin de la clase existente. Espera un status 409 Conflict.
- **testConflictWithExistingClassFinish()**: Asegura que se detecta un **conflicto de horario** cuando la hora de fin de la clase extra propuesta se solapa con una clase existente, aunque la hora de inicio sea posterior al inicio de la clase existente. Espera un status 409 Conflict.
- **testConflictWithExistingClassFull()**: Valida que se detecta un **conflicto de horario** cuando la clase extra propuesta abarca completamente el horario de una clase existente. Espera un status 409 Conflict.

En conjunto, estos test cubren la funcionalidad básica de varios *endpoints* de la API (Ejemplo de test pasados con éxito en la figura 7.9), incluyendo la recuperación de datos, la validación de información y la comprobación de la seguridad por clave API.

```

Test Runner for Java
✓ ⚙️ testGetClassesFromGroup() ${symbol-class} Sct
✓ ⚙️ testGetClassesFromGroup_unauthorized() ${symbol-class} Sct
✓ ⚙️ testGetGrades() ${symbol-class} ScheduleConsu
✓ ⚙️ testGetSubjectsGroups() ${symbol-class} Sched
✓ ⚙️ testValidateExtraClass() ${symbol-class} Sched
✓ ⚙️ testValidateSubscription() ${symbol-class} Sched

```

Figura 7.9: Pruebas unitarias e integración del servicio schedule-consumer-service

7.5. Sprint 4

En este punto ya tenemos un backend sólido y funcional, con servicios que permiten gestionar usuarios, autenticación, suscripciones a asignaturas y horarios de la UGR. El siguiente paso es desarrollar el frontend de la aplicación, que permitirá a los usuarios interactuar con el sistema de manera intuitiva y visual.

7.5.1. Frontend con Angular

El frontend se desarrolla con Angular (versión 19.0.2), lo que significa que se utiliza TypeScript como lenguaje principal, junto con HTML y CSS para la estructura y el estilo de la aplicación. Angular es un framework de desarrollo web que permite crear aplicaciones de una sola página (SPA) de manera eficiente y escalable.

Estructura del proyecto

El proyecto de Angular se organiza en “components”, “services” y “models”, siguiendo las mejores prácticas de Angular. Los componentes son las unidades básicas de la interfaz de usuario, los servicios se encargan de la lógica de negocio y la comunicación con el backend, y los modelos definen las estructuras de datos utilizadas en la aplicación.

Comunicación con el backend

La comunicación con el backend se realiza a través de servicios que utilizan HttpClient de Angular para hacer peticiones HTTP a los diferentes *endpoints* del API Gateway. Estos servicios manejan la autenticación mediante tokens JWT, que se envían en las cabeceras de las peticiones para identificar al usuario y su rol. Tanto el “access_token” como el “refresh_token” se almacenan en el almacenamiento local del navegador, lo que permite mantener la sesión del usuario activa entre recargas de página.

Además, también se ha hecho uso de “Observables” para manejar las respuestas del backend de manera asíncrona, lo que permite una experiencia de usuario más fluida y reactiva. Los servicios se encargan de transformar los datos recibidos del backend en objetos que pueden ser utilizados por los componentes de la interfaz de usuario.

Cada vez que se hace una petición al backend se hace uso de un “Interceptor” de Angular, este adjunta el token de acceso a la cabecera Authorization de casi todas las solicitudes salientes. Sin embargo, excluye ciertas rutas relacionadas con la autenticación inicial o el refresco de tokens (como /auth/login o /auth/refresh) para evitar bucles o requisitos innecesarios. Si una solicitud protegida se realiza sin un token de acceso, el interceptor redirige al usuario a la página de login y lanza un error. Su función más importante es el manejo de tokens de acceso expirados: si una API devuelve un error 401, el interceptor intenta refrescar el token usando el token de refresco. Si el refresco es exitoso, actualiza el token de acceso y reintenta la solicitud original. Si el token de refresco también ha expirado, elimina ambos tokens, redirige al usuario al login y reporta un error. Para otros tipos de errores HTTP, simplemente los propaga.

Rutas y navegación

El enrutamiento se gestiona mediante el módulo de enrutamiento de Angular, que permite definir rutas para cada componente de la aplicación. Esto facilita la navegación entre diferentes vistas y componentes sin necesidad de recargar la página completa, lo que mejora la experiencia del usuario. Además, se implementa un sistema de guardias de ruta para proteger las rutas que requieren autenticación, asegurando que solo los usuarios con un token JWT válido puedan acceder a ellas.

Interfaz de usuario

La interfaz de usuario se ha desarrollado únicamente con HTML, CSS y Tailwind CSS, sin utilizar librerías de componentes adicionales. Esto permite un mayor control sobre el diseño y la apariencia de la aplicación, adaptándola a las necesidades específicas del proyecto. Se han creado componentes reutilizables para las diferentes secciones de la aplicación, como formularios de inicio de sesión, registro, suscripciones y visualización de horarios. A continuación se muestran en las siguientes figuras ([7.10](#), [7.11](#), [7.12](#), [7.13](#), [7.14](#)) las páginas principales de la aplicación que incluyen el inicio de sesión, la página principal del calendario, la página de sincronización, la página de suscripciones y la página de eventos.



Figura 7.10: Página de inicio de sesión

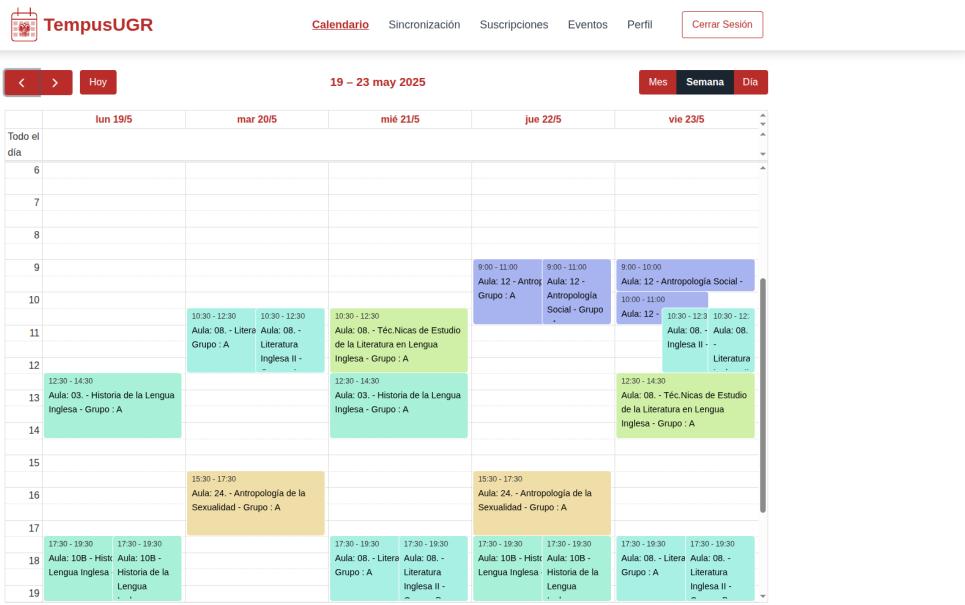


Figura 7.11: Página principal del calendario

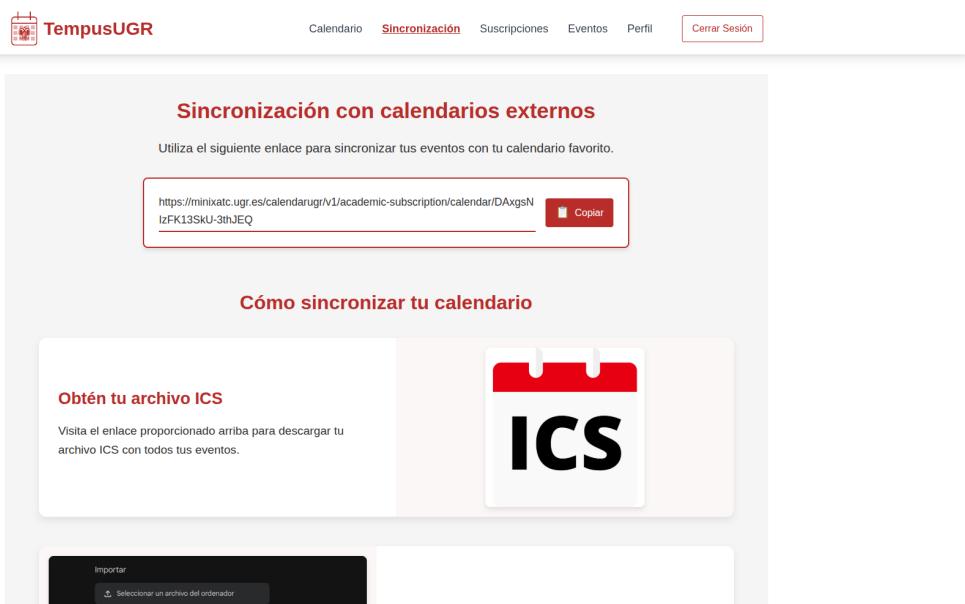


Figura 7.12: Página de sincronización con Google Calendar

Figura 7.13: Página de suscripciones a asignaturas

Figura 7.14: Página de eventos de grupo y facultad

Suscripción a las asignaturas de un profesor

Al principio de este sprint se ajustaron tanto el “Backlog” como el “Sprint Backlog” para incluir la posibilidad de que un profesor pueda suscribirse a las asignaturas que imparte automáticamente buscando su nombre. Esta es una funcionalidad que se detectó como necesaria durante el desarrollo del frontend, ya que tenemos la información necesaria para poder implementarla, y es una funcionalidad que se espera que sea utilizada por los profesores de la UGR. Al tener todos los servicios necesarios implementados, y funcionalidades similares

(obtener clases según grado, obtener todos los grados ...), no costó mucho tiempo y cambios significativos en el código. Se implementó un nuevo *endpoint* en el servicio *schedule-consumer-service* que permite obtener las asignaturas de un profesor a partir de su nombre, y se añadió una nueva funcionalidad en el servicio *academic-subscription-service* que permite suscribirse a varias asignaturas a la vez. Por otra parte en el apartado “suscripciones” se añadió un “dropdown” que sólo podrán visualizar los profesores y administradores, que les permitirá buscar por su nombre y suscribirse a todas sus asignaturas directamente.

7.6. Sprint 5

Las últimas tres semanas del desarrollo se destinaron a depurar y arreglar errores tanto en el frontend como en el backend. Además antes de comenzar este sprint también se hicieron modificaciones en el “Backlog” y el “Sprint Backlog” para añadir la funcionalidad de “reseteo de contraseña”. Esta se detectó al hacer pruebas exhaustivas del sistema. Al pretender desplegar el sistema y que esté preparado para su uso, esta es una funcionalidad que se considera esencial para cualquier sistema de gestión de usuarios, y que no se había implementado hasta ahora.

Al contrario que en el sprint anterior, esta funcionalidad sí requiere de varios pasos para su implementación puesto que requiere, en el caso del backend, un endpoint para solicitar el reseteo, lógica para poder mandar un correo al usuario, y otro endpoint para modificar su contraseña, mientras que en el frontend se requerían un modal para solicitar un correo de contacto, y una página para establecer la nueva contraseña.

En todos los sprint, además de desarrollo, se ha destinado parte del tiempo para ir completando la documentación (diagramas, esquemas, redacción, etc), pero en este sprint se pretende completar la memoria.

Además al final, y tras este sprint, se despliega el sistema en un servidor de producción, y se realizan pruebas de carga y estrés para comprobar que el sistema es capaz de soportar un número elevado de usuarios concurrentes. Este apartado se desarrolla con más detalle en el apartado de “Despliegue”⁸.

8. Despliegue del sistema

Una vez desarrollado el sistema, es necesario desplegarlo en un servidor para que los usuarios de la UGR puedan hacer uso de él. Para ello, se ha optado por contenerizar el sistema utilizando Docker, lo que permite una fácil gestión y escalabilidad de los microservicios que componen la aplicación.

El servidor en el que se despliega el sistema se encuentra en el edificio auxiliar de la ETSIIT y fue solicitado por el Departamento de Ingeniería de Computadores, Automática y Robótica. El servidor cuenta con las características técnicas definidas en el capítulo 5, en la sección de “Presupuesto del proyecto”.

Además se solicitó que el servidor tuviera acceso a la red de la UGR, para que los usuarios pudieran acceder al sistema desde cualquier punto de la universidad, y que se pudiera acceder al servidor de forma remota para poder gestionar el sistema y realizar tareas de mantenimiento. Al servidor se le ha asignado una IP privada de la red de la universidad. Por lo tanto, para acceder al servidor desde fuera de la red de la UGR, es necesario conectarse a la VPN de la universidad.

8.1. Configuración del servidor

Como paso previo a la contenerización del sistema y despliegue de la aplicación, se ha procedido a la configuración del servidor.

Esta configuración fue realizada tanto por el director como por el autor del TFG:

- Creación de usuarios y grupos necesarios para el despliegue de la aplicación.
- Instalación de Docker y Docker Compose. Configuración de Docker para que se ejecute como servicio al iniciar el sistema.
- Configurar ssh para permitir el acceso remoto al servidor, y scp para la transferencia de archivos.
- Instalar git para poder clonar los repositorios necesarios.

No hubo más pasos previos a la contenerización del sistema, y en gran parte esta es la ventaja de contenerizar el sistema, ya que permite una fácil gestión y despliegue de la aplicación sin necesidad de realizar configuraciones complejas en el servidor.

Sin hacer uso de esta tecnología se habrían tenido que realizar, entre otros, los siguientes pasos:

- Configuración de un servidor web (Apache) para servir la aplicación.

- Configuración de un servidor de base de datos (MySQL y MongoDB) para almacenar los datos de la aplicación.
- Instalación y configuración de Java y Maven para compilar y ejecutar los microservicios.
- Instalación y configuración de Node.js y Angular CLI para compilar el frontend de la aplicación.
- Configuración de un servidor de mensajería (RabbitMQ) para la comunicación entre microservicios.
- etc

Para acceder al servidor de manera remota se ha utilizado la VPN de la UGR, que permite conectarse al servidor de forma segura y acceder a los recursos de la red de la universidad.

Y para el paso de archivos entre el servidor y el equipo local se ha utilizado el protocolo SCP (Secure Copy Protocol), que permite transferir archivos de forma segura a través de SSH.

8.2. Contenerización del sistema

El primer paso realizado en este sentido ha sido la contenerización del backend del sistema, que está compuesto por varios microservicios.

8.2.1. Pasos para la contenerización del backend

En esta primera fase se han contenerizado, construido y levantado los servicios de la siguiente manera:

1. Crear una red en docker:

```
1 docker network create calendarugr
2
```

2. Generar los .jar de los microservicios, sin pasar los tests para una construcción sin conflictos para los servicios que ya están contenerizados:

```
1 ./mvnw clean package -DskipTests
2
```

3. Crear las imágenes de los microservicios (Ej imagen de Eureka service):

```
1 FROM amazoncorretto:21-alpine-jdk
2 WORKDIR /app
3 EXPOSE 8761
4 COPY ./target/eureka-service-0.0.1-SNAPSHOT.jar eureka-
service.jar
```

```
5
6     ENTRYPOINT ["java", "-jar", "eureka-service.jar"]
7
```

4. Construir la imagen de docker:

```
1     docker build -t eureka-service .
2
```

5. Para levantar los contenedores uno a uno (Ej levantando el contenedor de Eureka):

```
1     docker run -d --name eureka-service --network calendarugr -p
2         8761:8761 eureka-service
3
```

6. Bajar las imágenes oficiales de mysql:8.0.41 y mongo:6.0.4, además de las imágenes de RabbitMQ:

```
1     docker pull mysql:8.0.41
2     docker pull mongo:6.0.4
3     docker pull rabbitmq:3-management
4
```

7. Para levantar contenedores con variables de entorno (Ej levantando el contenedor de Mysql):

8.

```
1     docker run -p 3307:3306 --network calendarugr \
2         -e MYSQL_ROOT_PASSWORD=... \
3         -e MYSQL_USER=... \
4         -e MYSQL_PASSWORD=... \
5         -v /home/juanmi/mysql-scripts/init.sql:/docker-
6             entrypoint-initdb.d/init.sql \
7                 --name mysql \
8                     mysql:8.0.41
9
```

9. El init.sql es un script que se ejecuta al iniciar el contenedor de Mysql, y se utiliza para crear la base de datos y las tablas necesarias para el funcionamiento del sistema. El script se encuentra en la carpeta mysql-scripts del proyecto.

```
1     CREATE DATABASE IF NOT EXISTS DB_USER_SERVICE;
2     CREATE DATABASE IF NOT EXISTS DB_SCHEDULE_CONSUMER_SERVICE;
3
4     GRANT ALL PRIVILEGES ON DB_USER_SERVICE.* TO 'calendarugr'@'%
5         ';
6     GRANT ALL PRIVILEGES ON DB_SCHEDULE_CONSUMER_SERVICE.* TO ,
7         'calendarugr'@'%' ;
8
```

6 FLUSH PRIVILEGES;

7

10. Para levantar el contenedor de Mongo:

```
1 docker run -d --name mongodb \
2   -p 27018:27017 \
3   --network calendarugr \
4   -e MONGO_INITDB_ROOT_USERNAME=... \
5   -e MONGO_INITDB_ROOT_PASSWORD=... \
6   mongo:6.0.4
```

7

De esta manera se levantan todos los servicios uno a uno y se pueden probar de forma individual y en conjunto. Sin embargo, para facilitar el despliegue y la gestión de los microservicios, se ha optado por utilizar Docker Compose.

8.2.2. Docker Compose

Docker Compose es una herramienta que permite definir y ejecutar aplicaciones Docker multi-contenedor. Con Docker Compose, se puede definir la configuración de todos los microservicios en un único archivo `docker-compose.yml`, lo que facilita su gestión y despliegue.

Este enfoque nos permite centralizar la configuración de todos los microservicios en un único archivo, lo que facilita su gestión y despliegue, de manera que:

- Cada microservicio se define como un servicio en el archivo `docker-compose.yml`.
- Se especifican las imágenes de cada microservicio, los puertos que se exponen, las redes a las que pertenecen y las variables de entorno necesarias.
- Se definen las dependencias entre los servicios, lo que permite que Docker Compose gestione el orden de inicio de los contenedores.
- Se pueden definir volúmenes para persistir los datos de los servicios, como en el caso de MySQL y MongoDB.

De esta manera lo único que haría falta en el servidor para levantar todo el backend sería un directorio contenedor del archivo `docker-compose.yml`. Al tener este archivo referencia a las imágenes oficiales de MySQL, Mongo y RabbitMQ, además de las imágenes de los servicios subidos a Docker Hub, no es necesario tener las imágenes construidas en el servidor, ya que Docker Compose se encargará de descargarlas automáticamente al levantar los servicios.

Para levantar todos los servicios definidos en el archivo `docker-compose.yml`, se puede ejecutar el siguiente comando:

```
1 docker-compose up -d ( -d para que se levanten en segundo plano)
```

Además para facilitar aún se han creado automatizaciones para la construcción de las imágenes y el despliegue de los microservicios, de manera que se pueden ejecutar los siguientes comandos:

```
1 ./build_services.sh  
2 ./upload_to_hub.sh
```

Estos scripts se encargan de construir las imágenes de los microservicios y subirlas al repositorio de Docker Hub, lo que permite que se puedan desplegar en cualquier servidor con Docker instalado.

8.2.3. Pasos para la contenerización del frontend

El frontend del sistema está desarrollado en Angular y se ha decidido contenerizarlo utilizando Apache como servidor web. A continuación se detallan los pasos realizados para la dockerización del frontend:

1. Construir el proyecto Angular para producción:

```
1 ng build --configuration production  
2
```

Este comando generará una carpeta dist con los archivos necesarios para desplegar la aplicación. Estos serán trasladados a un directorio del servidor, por ejemplo, built_tempus, y deberá estar disponible en el mismo directorio que el docker-compose.yml, los certificados, y un directorio “apache” con el archivo de configuración de Apache y el Dockerfile.

Además, dentro del directorio built_tempus se debe crear un archivo .htaccess con el objetivo de redirigir todas las peticiones al archivo index.html del frontend, para que Angular pueda manejar el enrutamiento de la aplicación. El contenido del archivo .htaccess es el siguiente:

```
1 RewriteEngine On  
2 RewriteBase /  
3 RewriteRule ^index\.html$ - [L]  
4 RewriteCond %{REQUEST_FILENAME} !-f  
5 RewriteCond %{REQUEST_FILENAME} !-d  
6 RewriteRule . /index.html [L]  
7
```

2. Solicitar los certificados SSL necesarios para el dominio tempus.ugr.es. Estos certificados son necesarios para habilitar HTTPS en el servidor web, y habilitarlo tanto para el frontend como para el backend.
3. Copiar los certificados SSL en una carpeta del servidor, por ejemplo, en /home/user/certificates. Estos certificados son necesarios para habilitar HTTPS en el servidor web.

4. Crear un archivo de configuración para Apache (apache-ssl.conf) en el que se especifique la configuración del servidor web. Aquí se especifican el uso de SSL, la redirección de HTTP a HTTPS y la configuración del Reverse Proxy para el backend.
-

```
1      <VirtualHost *:4431.2 +TLSv1.3
20
21         SSLProxyEngine On
22         SSLProxyProtocol -all +TLSv1.2
23         SSLProxyCheckPeerCN off
24         SSLProxyCheckPeerName off
25
26         # PROXY PARA EL API GATEWAY
27         ProxyPass /calendarugr/v1 http://api-gateway:8090/
calendarugr/v1
28         ProxyPassReverse /calendarugr/v1 http://api-gateway:
8090/calendarugr/v1
29
30         <Directory /var/www/html>
31             Options Indexes FollowSymLinks
32             AllowOverride All
33             Require all granted
34         </Directory>
35
36             ErrorLog ${APACHE_LOG_DIR}/mi-app-error.log
37             CustomLog ${APACHE_LOG_DIR}/mi-app-access.log
combined
38         </VirtualHost>
```

Este archivo de configuración define un VirtualHost para el dominio tempus.ugr.es en el puerto 443 (HTTPS).

Gracias a esta configuración, Apache actuará como un proxy inverso para el backend, redirigiendo las peticiones al API Gateway que se ejecuta en el contenedor de Docker.

5. Creación del Dockerfile para el frontend, que se encargará de construir la imagen del contenedor que servirá la aplicación Angular. Este archivo irá en el mismo directorio que el archivo apache-ssl.conf.

```
1 # Base image
2 FROM ubuntu:latest
3
4 ENV DEBIAN_FRONTEND=noninteractive
5
6 # Install apache2
7 # Install dependencies
8 RUN apt-get update && apt-get install -y \
9     php \
10    apache2 \
11    libapache2-mod-php \
12    curl \
13    && rm -rf /var/lib/apt/lists/*
14 RUN apt-get update && apt-get install -y zip unzip git
15 RUN apt-get install -y iputils-ping && apt install -y
16 iproute2
17
18 # Activate apache2 modules and enable SSL and proxy
19 RUN a2enmod rewrite ssl proxy proxy_http && mkdir /etc/
20 apache2/ssl
21 # copy ssl files
22 COPY ./apache-ssl.conf /etc/apache2/sites-available/
23 apache-ssl.conf
24
25 # activate the site
26 RUN a2ensite apache-ssl.conf
27
28 EXPOSE 443
```

6. Diseñar el archivo docker-compose.yml para el frontend, que incluirá la configuración del contenedor de Apache y la redirección de las peticiones al API Gateway. En este archivo hacemos que el contenedor del frontend use la misma red que el resto de microservicios, y que se levante el contenedor de Apache con la configuración del archivo apache-ssl.conf.

Además se especifica el volumen donde se encuentran los certificados SSL, el directorio built_tempus que contiene los archivos del frontend y el archivo de configuración de Apache.

El directorio contenedor de lo necesario para desplegar el frontend debería contener algo parecido a lo siguiente:

```
1   - apache-docker/
2     - apache/
3       - apache-ssl.conf      # Configuracion de Apache con SSL
4       - DockerfileApache    # Dockerfile para construir el
contenedor
5       - certificados/
6         - ClavePrivada.pem   # Clave privada SSL
7         - Certificado.pem   # Certificado SSL
8       - docker-compose.yml   # Composicion de servicios Docker
9       - built_tempuis        # Carpeta donde se colocan los
archivos de la app Angular
```

8.3. Levantar el sistema

Una vez que se han configurado y contenerizado todos los microservicios, se puede levantar el sistema completo utilizando Docker Compose. Para ello se debe levantar primero el backend, que es el que crea también la red de Docker necesaria para el frontend, y luego el frontend.

Con esto se consigue que el sistema esté completamente desplegado y accesible a través del dominio tempus.ugr.es.

8.3.1. Paso del HTTP a HTTPS en las llamadas al backend

Si sólo se levantara el backend exponiendo el puerto 8090 para las peticiones al api gateway, las peticiones al backend se realizarían a través de HTTP. Sin embargo, para que el sistema funcione correctamente y se pueda acceder a él a través del dominio tempus.ugr.es, es necesario que las peticiones al backend se realicen a través de HTTPS. Para ello, se ha configurado Apache como un proxy inverso que redirige las peticiones al backend a través de HTTPS. De esta manera, las peticiones al backend se realizan a través del dominio tempus.ugr.es y el puerto 443, que es el puerto por defecto para HTTPS.

Ejemplo de endpoint del backend al que se accede a través de HTTP:

```
1 http://172.25.190.139:8090/calendarugr/v1/schedule-consumer/classes-from-
group
```

Ejemplo de endpoint del backend al que se accede a través de HTTPS:

```
1 https://tempus.ugr.es/calendarugr/v1/schedule-consumer/classes-from-group
```

8.3.2. Pruebas de carga en un entorno real

Una vez todo lo necesario levantado para un funcionamiento normal del sistema, se han realizado pruebas de carga en un entorno real para comprobar el rendimiento y la escalabilidad del sistema. Estas pruebas se han realizado utilizando Locust [70], que es una herramienta de código abierto para realizar pruebas de carga y rendimiento en aplicaciones web.

Las pruebas de carga se han realizado para distintos usuarios (con una media de diez suscripciones a grupos de asignatura) solicitando la información de su calendario entero (acción más común del sistema), al endpoint <https://tempus.ugr.es/calendarugr/v1/academic-subscription/entire-calendar>. Además las solicitudes no se han hecho directamente al backend mediante llamadas http, sino que se han hecho a través del dominio tempus.ugr.es, que es el que se utiliza para acceder al sistema desde el navegador. Esto permite simular un uso real del sistema, ya que los usuarios acceden al sistema a través del dominio y no directamente al backend.

Sabiendo que en la UGR hay aproximadamente 60.000 estudiantes, y que el número de profesores :

- Pruebas de carga con 100 usuarios concurrentes, simulando un uso normal del sistema.
- Pruebas de carga con 500 usuarios concurrentes, simulando un uso intensivo del sistema.
- Pruebas de carga con 1000 usuarios concurrentes, simulando un uso extremo del sistema.

Los parámetros que mide la herramienta son los siguientes:

1. **Req.:** Número total de solicitudes realizadas.
2. **Fails:** Número de solicitudes que han fallado.
3. **Med(ms):** Tiempo medio de respuesta en milisegundos.
4. **95 %ile (ms):** Tiempo de respuesta en el percentil 95 (es decir, el 95 % de las solicitudes se han respondido en este tiempo o menos).
5. **99 %ile (ms):** Tiempo de respuesta en el percentil 99 (es decir, el 99 % de las solicitudes se han respondido en este tiempo o menos).
6. **Avg(ms):** Tiempo medio de respuesta en milisegundos.
7. **Min(ms):** Tiempo mínimo de respuesta en milisegundos.
8. **Max(ms):** Tiempo máximo de respuesta en milisegundos.
9. **Avg size(bytes):** Tamaño medio de la respuesta en bytes.
10. **RPS:** Solicitudes por segundo.
11. **Fails/s:** Fallos por segundo.

Pruebas de carga con 100 usuarios concurrentes

En esta prueba se evaluó el comportamiento del sistema bajo una carga de 100 usuarios concurrentes durante 2 minutos. Los resultados cuantitativos se resumen en la Tabla 8.1.

Req.	Fails	Med(ms)	95 %(ms)	99 %(ms)	Avg(ms)	Min(ms)	Max(ms)	Avg size(Bytes)	RPS	Fails/s
4172	35	84	120	6300	213.04	1	7329	9636.47	32.2	0.1

Tabla 8.1: Resultados de la prueba de carga con 100 usuarios concurrentes.



Figura 8.1: Gráfica de la prueba de carga con 100 usuarios concurrentes.

Del análisis de la Gráfica en la figura 8.1 y los datos de la Tabla 8.1, se extraen las siguientes observaciones:

Rendimiento General y Tiempos de Respuesta: El sistema gestionó un total de 4172 solicitudes, alcanzando un throughput promedio de 32.2 RPS. La mediana del tiempo de respuesta (Med(ms)) se situó en unos excelentes 84 ms, y el percentil 95 (95 %(ms)) en 120 ms. Estos valores indican que, una vez superada la fase inicial, la gran mayoría de los usuarios experimentaron tiempos de respuesta muy satisfactorios. El tiempo promedio de respuesta (Avg(ms)) fue de 213.04 ms. La diferencia entre la media y la mediana sugiere la presencia de algunas latencias más elevadas, corroborado por el percentil 99 (99 %(ms)) de 6300 ms (6.3 segundos) y un tiempo máximo (Max(ms)) de 7329 ms (7.3 segundos). Aunque estos valores de cola son considerablemente más altos que el P95, afectan a un porcentaje muy reducido de peticiones bajo esta carga.

Comportamiento Inicial (Cold Start): La gráfica de “Response Times (ms)” (Gráfica en la figura 8.1) muestra un pico de latencia al inicio de la prueba

(aproximadamente entre 17:30:00 y 17:40:00), donde el P95 alcanzó hasta 7300 ms y la mediana (P50) unos 5200 ms. Este comportamiento es característico de un “arranque en frío” del sistema, atribuible a factores como la carga inicial de código en el servidor de aplicaciones, el calentamiento de cachés (aplicación y base de datos PostgreSQL), la inicialización del pool de conexiones a la base de datos y el impacto de las primeras solicitudes sobre un sistema que aún no ha alcanzado su estado óptimo de operación. Tras esta fase inicial, los tiempos de respuesta se estabilizaron rápidamente a los niveles mencionados anteriormente.

Tasa de Fallos: Se registraron 35 fallos (Fails) sobre 4172 solicitudes, lo que equivale a una tasa de éxito del 99.16 % (tasa de fallo del 0.84 %). Esta tasa se considera muy baja. Los fallos fueron consistentemente del tipo RemoteDisconnected('Remote end closed connection without response'), sugiriendo que el servidor cerró la conexión prematuramente, posiblemente debido a timeouts durante el pico de latencia del “cold start” o por eventos transitorios. Dada la baja incidencia, estos fallos no señalan un problema sistémico grave bajo esta carga. El promedio de fallos por segundo fue de 0.1 Fails/s.

Conclusión Parcial (100 Usuarios): Bajo una carga de 100 usuarios concurrentes, y descontando el efecto inicial de “cold start”, el sistema demostró ser estable y eficiente, ofreciendo tiempos de respuesta excelentes para la mayoría de las solicitudes y manteniendo una tasa de fallos mínima. Las latencias de cola (P99 y Máximo) son las primeras señales de peticiones que tardan más, pero su impacto es limitado en este nivel de carga.

Pruebas de carga con 500 usuarios concurrentes

Incrementando la carga a 500 usuarios concurrentes durante 2 minutos, se obtuvieron los resultados presentados en la Tabla 8.2.

Req.	Fails	Med(ms)	95 %(ms)	99 %(ms)	Avg(ms)	Min(ms)	Max(ms)	Avg size(Bytes)	RPS	Fails/s
4918	48	88	280	15000	887.77	1	105514	9623.15	48.2	0.3

Tabla 8.2: Resultados de la prueba de carga con 500 usuarios concurrentes.

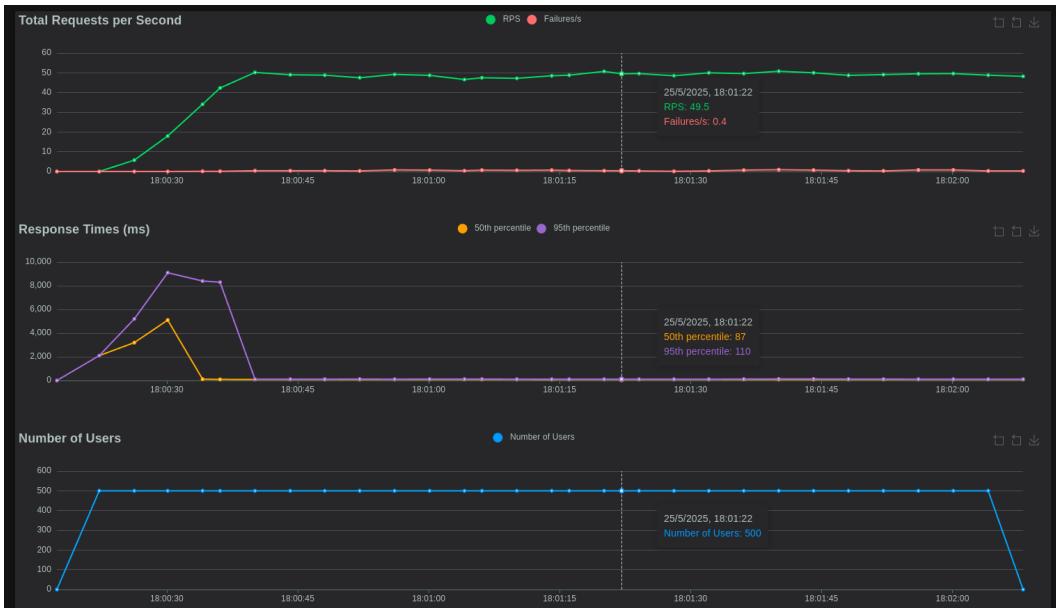


Figura 8.2: Gráfica de la prueba de carga con 500 usuarios concurrentes.

Rendimiento General y Tiempos de Respuesta: Con 500 usuarios, el sistema procesó 4918 solicitudes con un throughput promedio de 48.2 RPS. Es notable que, aunque el número de usuarios se quintuplicó respecto a la prueba anterior, el RPS solo aumentó aproximadamente un 50 % (de 32.2 a 48.2 RPS), lo que sugiere una disminución en la escalabilidad lineal del rendimiento. La mediana del tiempo de respuesta (Med(ms)) se mantuvo excelente en 88 ms, similar a la prueba de 100 usuarios. Sin embargo, el P95(ms) aumentó a 280 ms, aún aceptable, pero indicando una mayor proporción de solicitudes más lentas. El impacto en las latencias de cola es mucho más pronunciado: el P99(ms) se disparó a 15000 ms (15 segundos) y el Max(ms) a 105514 ms (105.5 segundos). Estos valores son alarmantes y evidencian que un 1-5% de las solicitudes experimentan degradaciones severas. El Avg(ms) de 887.77 ms está fuertemente influenciado por estas latencias extremas.

Comportamiento Inicial (Cold Start): La Gráfica en la figura 8.2 también muestra un pico de latencia inicial (P95 hasta 9100 ms, P50 hasta 5100 ms), atribuible al “cold start”, similar al observado con 100 usuarios. Tras esta fase, los tiempos para la mayoría de las solicitudes (mediana) se estabilizaron.

Tasa de Fallos: Se registraron 48 fallos (Fails) sobre 4918 solicitudes, resultando en una tasa de éxito del 99.02 % (tasa de fallo del 0.98 %). Esta tasa sigue siendo baja, aunque ligeramente superior a la prueba con 100 usuarios. El promedio de fallos por segundo fue de 0.3 Fails/s.

Conclusión Parcial (500 Usuarios): Con 500 usuarios concurrentes, el sistema mantiene una buena mediana de tiempo de respuesta, pero muestra signos claros

de estrés en las latencias de cola (P99 y Máximo). La escalabilidad del throughput no es lineal. Aunque la tasa de fallos es baja, la degradación severa para un pequeño porcentaje de solicitudes es un indicador de que el sistema se acerca a sus límites de capacidad o presenta cuellos de botella específicos que se manifiestan con mayor carga.

Pruebas de carga con 1000 usuarios concurrentes

La prueba final se realizó con 1000 usuarios concurrentes durante 2 minutos. Los resultados se detallan en la Tabla 8.3.

Req.	Fails	Med(ms)	95 %(ms)	99 %(ms)	Avg(ms)	Min(ms)	Max(ms)	Avg size(Bytes)	RPS	Fails/s
6672	215	92	2500	89000	2829.37	5	135196	9404.85	50.1	0.6

Tabla 8.3: Resultados de la prueba de carga con 1000 usuarios concurrentes.

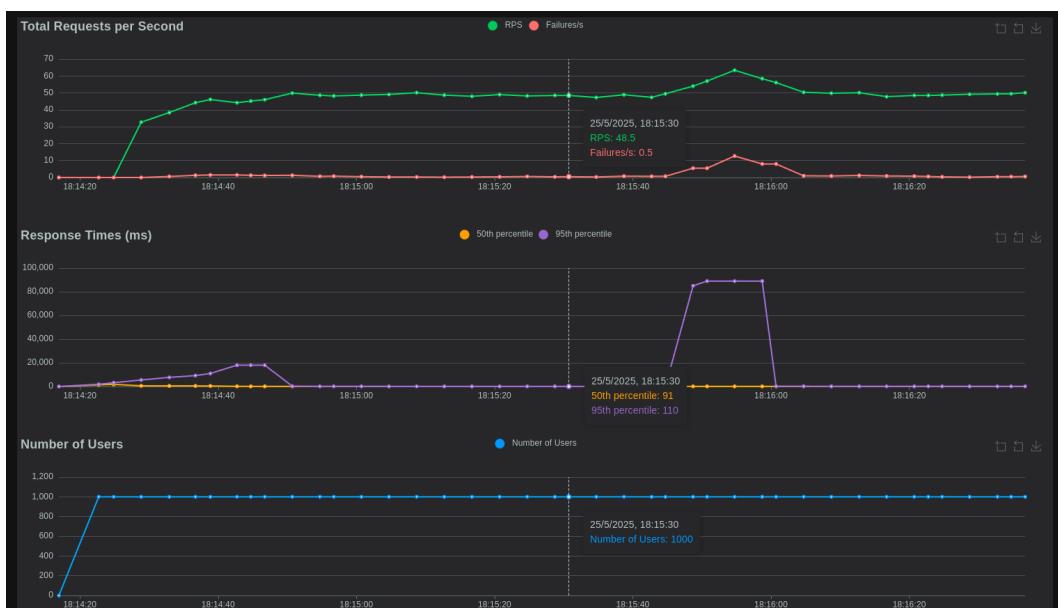


Figura 8.3: Gráfica de la prueba de carga con 1000 usuarios concurrentes.

Rendimiento General y Tiempos de Respuesta: Bajo esta carga intensiva, el sistema procesó 6672 solicitudes, con un throughput promedio de 50.1 RPS. Es crítico observar que duplicar los usuarios de 500 a 1000 apenas incrementó el RPS (de 48.2 a 50.1 RPS), lo que indica que el sistema ha alcanzado un techo de rendimiento o está operando muy cerca de su capacidad máxima. La mediana (Med(ms)) se mantiene sorprendentemente baja en 92 ms. No obstante, este dato es engañoso si se considera aisladamente. El P95(ms) se degradó drásticamente a 2500 ms (2.5 segundos), un valor que ya impacta negativamente la experiencia del usuario. Las latencias de cola son extremas: el P99(ms) alcanzó los 89000 ms (89 segundos) y el Max(ms) los 135196 ms (135.2 segundos). Estos tiempos son indicativos de un sistema sobrecargado. El Avg(ms) de 2829.37 ms refleja el fuerte impacto de estas latencias extremas.

Comportamiento Inicial y Durante la Prueba: Se observa el pico de “cold start” (P95 hasta 18000 ms), aunque la mediana durante este pico inicial se mantuvo más contenida (140 ms) en comparación con los P95. Un comportamiento anómalo destacado en la Gráfica de la figura 8.3 ocurrió alrededor de los 90 segundos de prueba, donde el P99 se disparó hasta los 89000 ms. Esto sugiere un evento de saturación o un problema severo que afectó a un pequeño porcentaje de solicitudes de manera aguda durante la ejecución de la prueba, y no solo al inicio.

Tasa de Fallos: Se registraron 215 fallos (Fails) sobre 6672 solicitudes, elevando la tasa de fallo al 3.22 % (tasa de éxito del 96.78 %). Aunque no es masiva, esta tasa es significativamente superior a las pruebas anteriores y, combinada con las latencias extremas, indica problemas de estabilidad. El promedio de fallos por segundo fue de 0.6 Fails/s.

Conclusión Parcial (1000 Usuarios): Con 1000 usuarios concurrentes, el sistema está claramente sobrecargado. A pesar de una mediana de respuesta aparentemente buena, el P95 es deficiente y los P99 y Máximo son inaceptables, indicando que una proporción no despreciable de usuarios experimentaría tiempos de espera muy prolongados. El throughput se ha estancado, y la tasa de fallos, aunque no catastrófica, es una señal de alerta. El sistema no es capaz de manejar esta carga de forma eficiente ni estable. Las posibles causas de esta saturación incluyen:

- **Saturación de recursos del servidor:** Límites de CPU, memoria, I/O, o descriptores de fichero en el servicio de aplicación.
- **Cuellos de botella en la base de datos:** Agotamiento del pool de conexiones, contención de bloqueos (locks), consultas lentas bajo concurrencia, o límites de recursos (CPU, RAM, IOPS) en PostgreSQL.
- **Agotamiento de recursos de red:** Posible saturación del ancho de banda o límites en el número de conexiones concurrentes a nivel de sistema operativo o balanceador de carga.
- **Límites de autoescalado no alcanzados o ineficaces:** Si el sistema cuenta con autoescalado, este podría no estar respondiendo con la suficiente rapidez o podría haber alcanzado sus límites configurados.

Conclusiones de las Pruebas de Carga

Las pruebas de carga progresiva con 100, 500 y 1000 usuarios concurrentes han proporcionado información valiosa sobre el comportamiento y los límites del sistema:

- **Carga Baja (100 Usuarios):** Tras la fase de calentamiento inicial (“cold start”), el sistema opera de forma óptima, con tiempos de respuesta excelentes para la mayoría de las solicitudes (Mediana 84 ms, P95 120 ms) y una tasa de fallos muy baja (0.84 %). El throughput promedio fue de 32.2 RPS.

- **Carga Moderada (500 Usuarios):** El sistema mantiene una buena respuesta para la mayoría de los usuarios (Mediana 88 ms), pero comienzan a emerger problemas significativos en las latencias de cola (P99 de 15 segundos, Máximo de 105.5 segundos). El throughput aumentó a 48.2 RPS, pero la escalabilidad no es lineal, sugiriendo la aparición de cuellos de botella. La tasa de fallos (0.98 %) se mantiene baja.
- **Carga Alta (1000 Usuarios):** El sistema muestra signos evidentes de saturación. Aunque la mediana de respuesta (92 ms) puede parecer aceptable, el P95 (2.5 segundos) es deficiente y el P99 (89 segundos) y Máximo (135.2 segundos) son críticos. El throughput apenas aumenta a 50.1 RPS, indicando un techo de rendimiento. La tasa de fallos se incrementa al 3.22 %. El sistema no maneja esta carga de manera efectiva.

En resumen, el sistema es robusto y eficiente bajo cargas bajas. A medida que la concurrencia aumenta a 500 usuarios, se observan las primeras señales de estrés, principalmente en las solicitudes más lentas. Con 1000 usuarios, el sistema está sobrecargado, manifestando una degradación severa en los tiempos de respuesta para un porcentaje significativo de las solicitudes y un estancamiento del throughput.

Para mejorar la capacidad del sistema de manejar cargas más altas y optimizar su rendimiento general, se proponen las siguientes líneas de actuación:

- **Análisis Profundo de Cuellos de Botella:** Utilizar herramientas de profiling y monitorización avanzada (APM) en el servidor de aplicaciones y en la base de datos para identificar los cuellos de botella exactos que causan la degradación del P99 y el estancamiento del RPS bajo cargas de 500 y 1000 usuarios. Investigar específicamente las consultas lentas, el uso de CPU/Memoria/IO y la contención de locks.
- **Optimización de la Base de Datos (PostgreSQL):**
 - Revisar y optimizar las consultas más frecuentes o lentas identificadas en el análisis.
 - Asegurar la correcta definición y uso de índices.
 - Ajustar la configuración de PostgreSQL (ej. `shared_buffers`, `work_mem`, `max_connections`, configuración del pool de conexiones) para la carga esperada.
- **Optimización del Servidor de Aplicaciones:**
 - Ajustar la configuración del servidor (ej. tamaño del heap de la JVM si es Java, número de workers/threads) para optimizar el uso de recursos.
 - Revisar el código de la aplicación en busca de ineficiencias, especialmente en las rutas críticas o aquellas que muestran mayor latencia.
- **Estrategias de Caché:** Implementar o mejorar las estrategias de caché a diferentes niveles (ej. caché de datos de aplicación con Redis o Memcached,

caché de consultas de base de datos) para reducir la carga sobre los componentes más lentos, especialmente la base de datos.

- **Escalabilidad Horizontal y Balanceo de Carga:** Si no está implementado, considerar una arquitectura de escalado horizontal con un balanceador de carga para distribuir las solicitudes entre múltiples instancias de la aplicación. Si ya existe, revisar la configuración del balanceador y la eficiencia del autoescalado.
- **Revisión de Timeouts y Configuración de Red:** Ajustar los timeouts a nivel de aplicación, servidor web y balanceador de carga para evitar cierres prematuros de conexión, especialmente bajo carga, sin enmascarar problemas de rendimiento subyacentes.

La implementación de estas mejoras debería ser seguida por nuevas pruebas de carga para validar su efectividad y asegurar que el sistema puede escalar de manera eficiente y estable a mayores niveles de concurrencia.

8.3.3. Solución a la sincronización con Google Calendar

Nuestro servidor cuenta con una IP privada dentro de la red de la UGR, por lo que no es accesible desde el exterior de la red sin VPN. Esto supone un problema para la sincronización con Google Calendar, ya que este servicio requiere que el servidor sea accesible desde Internet para poder sincronizar el calendario de los usuarios con Google Calendar.

Para solucionar este problema, Francisco Manuel Illeras García, profesor titular de la ETSIIT perteneciente al departamento de Ingeniería de Computadores, Automática y Robótica, ha configurado un Reverse Proxy en su servidor con IP pública que redirige las peticiones asociadas a obtener la url de sincronización con Google Calendar a nuestro servidor. De esta manera, cuando un usuario solicita la sincronización con Google Calendar, la petición se redirige al servidor de Francisco Manuel Illeras García, que a su vez redirige la petición a nuestro servidor. Esto permite que los usuarios puedan sincronizar sus calendarios con Google Calendar sin necesidad de que nuestro servidor sea accesible desde fuera de la red de la UGR.

9. Conclusiones y trabajos futuros

9.1. Evaluación del proyecto

Meter enlace al proyecto en GitHub

– Juanmi

9.2. Dificultades y resolución

9.3. Mejoras posibles y trabajos futuros

Integración con sistemas institucionales de la UGR (API de los horarios, autenticación de la UGR, matriculaciones...), Despliegue CI/CD, Proyecto vivo y colaborativo en Github, Monitorización y estadísticas, Optimización del sistema para mejor rendimiento

– Juanmi

Bibliografía

- [1] Universidad de Granada. Promoción internacional de la universidad de granada, 2025. URL <https://internacional.ugr.es/informacion/presentacion/promocion-ugr>. Accedido el 15 de marzo de 2025.
- [2] Universidad de Granada. Página principal de la escuela técnica superior de ingeniería informática e ingeniería en tecnologías de telecomunicación, 2025. URL <https://etsiit.ugr.es/>. Accedido el 23 de octubre de 2024.
- [3] Universidad de Granada. Página principal de grados ugr, 2025. URL <https://grados.ugr.es/>. Accedido el 23 de octubre de 2024.
- [4] Universidad de Granada. Página principal del departamento de ciencias de la computación e inteligencia artificial, 2025. URL <https://decsai.ugr.es/>. Accedido el 23 de octubre de 2024.
- [5] Universidad de Granada. Página principal de la secretaría general de la universidad de granada, 2025. URL <https://secretariageneral.ugr.es>. Accedido el 23 de octubre de 2024.
- [6] My Study Life. Página principal de my study life, 2025. URL <https://mystudylife.com/>. Accedido el 12 de noviembre de 2024.
- [7] Google. Página principal de google calendar, 2025. URL <https://calendar.google.com/>. Accedido el 12 de noviembre de 2024.
- [8] Microsoft. Página principal de microsoft outlook, 2025. URL <https://outlook.live.com/>. Accedido el 12 de noviembre de 2024.
- [9] Moodle. Página principal de moodle, 2025. URL <https://moodle.org/>. Accedido el 12 de noviembre de 2024.
- [10] MDN Web Docs. Métodos de petición http. Documentación en línea, Mozilla, March 2025. Disponible en: <https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Methods>. Accedido el 14 de mayo de 2025.
- [11] Neal Ford, Mark Richards, Pramod Sadalage, and Zhamak Dehghani. *Software Architecture: The Hard Parts*. O'Reilly Media, 2021. ISBN 978-1098109424.
- [12] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson, 2017. Capítulo 22: Layered Architecture.
- [13] IBM. ¿qué es la arquitectura orientada a servicios (soa)?, 2025. URL <https://www.ibm.com/es-es/topics/soa>. Accedido el 15 de enero de 2025.
- [14] Amazon Web Services. Arquitectura dirigida por eventos (event-driven architecture) en aws, 2025. URL <https://aws.amazon.com/es/event-driven-architecture/>. Accedido el 15 de enero de 2025.

- [15] Amazon Web Services. Understanding serverless architectures, 2025. URL <https://docs.aws.amazon.com/whitepapers/latest/optimizing-enterprise-economics-with-serverless/understanding-serverless-architectures.html>. Accedido el 15 de enero de 2025.
- [16] Microsoft Learn. Estilo de arquitectura de microservicios, 2025. URL <https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices>. Accedido el 16 de enero 2025.
- [17] Inc. Pivotal Software. Spring cloud reference documentation, 2025. URL <https://spring.io/projects/spring-cloud>. Accedido el 15 de marzo de 2025.
- [18] Microsoft .net core documentation, 2025. URL <https://learn.microsoft.com/en-us/dotnet/core/>. Accedido el 15 de marzo de 2025.
- [19] Express.js Foundation. Express - node.js web application framework, 2025. URL <https://expressjs.com/>. Accedido el 15 de marzo de 2025.
- [20] Adrian Holovaty and Jacob Kaplan-Moss. *The Definitive Guide to Django: Web Development Done Right*. Apress, 2009. ISBN 978-1430219361.
- [21] Armin Ronacher. Flask documentation, 2025. URL <https://flask.palletsprojects.com/>. Accedido el 15 de marzo de 2025.
- [22] Mozilla Foundation. Javascript (mdn web docs), 2025. URL <https://developer.mozilla.org/es/docs/Web/JavaScript>. Accedido el 15 de marzo de 2025.
- [23] Mozilla Foundation. Html: Hypertext markup language (mdn web docs), 2025. URL <https://developer.mozilla.org/es/docs/Web/HTML>. Accedido el 15 de marzo de 2025.
- [24] Mozilla Foundation. Css: Cascading style sheets (mdn web docs), 2025. URL <https://developer.mozilla.org/es/docs/Web/CSS>. Accedido el 15 de marzo de 2025.
- [25] Inc. Meta Platforms. React – a javascript library for building user interfaces, 2025. URL <https://react.dev/>. Accedido el 15 de marzo de 2025.
- [26] Google LLC. Angular - the modern web developer's platform, 2025. URL <https://angular.io/>. Accedido el 15 de marzo de 2025.
- [27] Evan You et al. Vue.js - the progressive javascript framework, 2025. URL <https://vuejs.org/>. Accedido el 15 de marzo de 2025.
- [28] Oracle Corporation. Mysql :: The world's most popular open source database, 2025. URL <https://www.mysql.com/>. Accedido el 15 de marzo de 2025.
- [29] The PostgreSQL Global Development Group. Postgresql: The world's most advanced open source relational database, 2025. URL <https://www.postgresql.org/>. Accedido el 15 de marzo de 2025.

- [30] Microsoft. Microsoft sql server, 2025. URL <https://www.microsoft.com/en-us/sql-server/>. Accedido el 15 de marzo de 2025.
- [31] MongoDB Inc. Mongodb: The developer data platform, 2025. URL <https://www.mongodb.com/>. Accedido el 15 de marzo de 2025.
- [32] Redis Ltd. Redis: In-memory data store, cache, and message broker, 2025. URL <https://redis.io/>. Accedido el 15 de marzo de 2025.
- [33] The Apache Software Foundation. Apache cassandra, 2025. URL <https://cassandra.apache.org/>. Accedido el 15 de marzo de 2025.
- [34] Inc. Neo4j. Neo4j: The world's leading graph database, 2025. URL <https://neo4j.com/>. Accedido el 15 de marzo de 2025.
- [35] Memcached Team. Memcached, 2025. URL <https://memcached.org/>. Accedido el 15 de marzo de 2025.
- [36] Inc. Amazon Web Services. Amazon rds (relational database service), 2025. URL <https://aws.amazon.com/rds/>. Accedido el 15 de marzo de 2025.
- [37] Google Cloud. Google cloud sql, 2025. URL <https://cloud.google.com/sql>. Accedido el 15 de marzo de 2025.
- [38] Microsoft. Azure cosmos db, 2025. URL <https://azure.microsoft.com/en-us/products/cosmos-db/>. Accedido el 15 de marzo de 2025.
- [39] Inc. Amazon Web Services. Amazon aurora, 2025. URL <https://aws.amazon.com/rds/aurora/>. Accedido el 15 de marzo de 2025.
- [40] Google Cloud. Google cloud spanner, 2025. URL <https://cloud.google.com/spanner>. Accedido el 15 de marzo de 2025.
- [41] Oracle. Form-based authentication, 2025. URL <https://docs.oracle.com/cd/E19798-01/821-1841/6n mq2cpki/index.html>. Accedido el 15 de marzo de 2025.
- [42] JWT.io. Json web token (jwt) introduction, 2025. URL <https://jwt.io/introduction>. Accedido el 15 de marzo de 2025.
- [43] National Institute of Standards and Technology (NIST). Multi-factor authentication (mfa), 2025. URL <https://pages.nist.gov/800-63-3/sp800-63b.html>. Accedido el 15 de marzo de 2025.
- [44] Dick Hardt. Oauth 2.0 authorization framework, 2012. URL <https://datatracker.ietf.org/doc/html/rfc6749>. RFC 6749. Accedido el 15 de marzo de 2025.
- [45] IETF. Ldap: Lightweight directory access protocol, 2025. URL <https://datatracker.ietf.org/doc/html/rfc4511>. RFC 4511. Accedido el 15 de marzo de 2025.

- [46] Roy T. Fielding. Rest: Representational state transfer, 2000. URL https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Accedido el 15 de marzo de 2025.
- [47] W3C. Soap version 1.2 part 1: Messaging framework (second edition), 2007. URL <https://www.w3.org/TR/soap12-part1/>. Accedido el 15 de marzo de 2025.
- [48] Google. grpc: A high performance, open-source universal rpc framework, 2025. URL <https://grpc.io/>. Accedido el 15 de marzo de 2025.
- [49] Inc. Meta Platforms. Graphql: A query language for your api, 2025. URL <https://graphql.org/>. Accedido el 15 de marzo de 2025.
- [50] Inc. VMWare. Rabbitmq: Messaging that just works, 2025. URL <https://www.rabbitmq.com/>. Accedido el 15 de marzo de 2025.
- [51] The Apache Software Foundation. Apache kafka: A distributed streaming platform, 2025. URL <https://kafka.apache.org/>. Accedido el 15 de marzo de 2025.
- [52] Inc. Amazon Web Services. Amazon simple queue service (sqS), 2025. URL <https://aws.amazon.com/sqs/>. Accedido el 15 de marzo de 2025.
- [53] Docker Inc. Docker: Empowering app development for developers, 2025. URL <https://www.docker.com/>. Accedido el 15 de marzo de 2025.
- [54] The Kubernetes Authors. Kubernetes: Production-grade container orchestration, 2025. URL <https://kubernetes.io/>. Accedido el 15 de marzo de 2025.
- [55] HashiCorp. Terraform by hashicorp, 2025. URL <https://www.terraform.io/>. Accedido el 15 de marzo de 2025.
- [56] Inc. Red Hat. Ansible automation platform, 2025. URL <https://www.ansible.com/>. Accedido el 15 de marzo de 2025.
- [57] Jenkins Project. Jenkins: Build great things at any scale, 2025. URL <https://www.jenkins.io/>. Accedido el 15 de marzo de 2025.
- [58] Atlassian. Historias de usuario, 2024. URL <https://www.atlassian.com/es/agile/project-management/user-stories>. Accedido el 17 de diciembre de 2024.
- [59] GanttPRO. Ganttpro: Software de diagramas de gantt online, 2025. URL <https://app.ganttpro.com/>. Accedido el 17 de marzo de 2025.
- [60] TempusUGR. Repositorio de tempusugr en github, 2025. URL <https://github.com/TempusUGR>. Accedido el 24 de abril de 2025.
- [61] TempusUGR. Proyecto tempusugr - gestión y seguimiento en github projects, 2025. URL <https://github.com/orgs/TempusUGR/projects/3/views/4>. Accedido el 29 de marzo de 2025.

- [62] Clockify. Página principal de clockify, 2025. URL <https://clockify.me/>. Accedido el 12 de noviembre de 2024.
- [63] Glassdoor. Sueldos de junior full stack developer, 2025. URL https://www.glassdoor.es/Sueldos/junior-full-stack-developer-sueldo-SRCH_K00,27.htm. Consultado el 18 de marzo de 2025.
- [64] Niklas Heer. speed-comparison: A repo which compares the speed of different programming languages, 2024. URL <https://github.com/niklas-heer/speed-comparison>. Accedido el 20 de marzo de 2025.
- [65] Roy Fielding and Julian Reschke. Hypertext transfer protocol (http/1.1): Semantics and content, June 2014. URL <https://datatracker.ietf.org/doc/html/rfc7231>.
- [66] Tim Berners-Lee, Larry Masinter, and Mark McCahill. Uniform resource locators (url), December 1994. URL <https://datatracker.ietf.org/doc/html/rfc1738>.
- [67] Daniel Pérez. Los 10 heurísticos de usabilidad de nielsen: cómo mejorar la experiencia de usuario, 2022. URL <https://medium.com/pildorasux/10-heuristicos-nielsen-abc9c6ad04c0>. Accedido el 3 de junio de 2025.
- [68] Universidad de Granada Centro de Servicios de Informática y Redes de Comunicaciones (CSIRC). Autenticación ugr - servicios web, 2025. URL <https://csirc.ugr.es/personal/servicios-web/autenticacion-ugr>. Accedido el 15 de marzo de 2025.
- [69] iCalendar.org. icalendar - rfc 5545, 2025. URL <https://icalendar.org/>. Accedido el 15 de marzo de 2025.
- [70] Locust. Locust - scalable user load testing tool, 2025. URL <https://locust.io/>. Accedido el 25 de mayo de 2025.

Anexo: Glosario

A continuación se presenta un glosario con las definiciones de términos técnicos utilizados a lo largo del trabajo:

SCRUM : es un marco de trabajo ágil para el desarrollo de software. Se basa en la iteración y la colaboración entre los miembros del equipo de desarrollo.

Backlog : es una lista priorizada de tareas y requisitos que deben completarse en un proyecto. El backlog se utiliza para planificar el trabajo en cada sprint.

LMS : es un sistema de gestión de aprendizaje. Se utiliza para administrar, documentar, rastrear, informar y entregar cursos de formación. Un ejemplo de LMS es Moodle, que es un sistema de gestión de aprendizaje de código abierto.

Docker : es una plataforma de software que permite crear, desplegar y ejecutar aplicaciones en contenedores. Los contenedores son entornos ligeros y portátiles que permiten ejecutar aplicaciones de manera aislada del sistema operativo subyacente.

Microservicios : es un estilo arquitectónico que estructura una aplicación como un conjunto de servicios pequeños y autónomos. Cada servicio se ejecuta en su propio proceso y se comunica con otros servicios a través de APIs.

API : es un conjunto de definiciones y protocolos que permiten la comunicación entre diferentes sistemas. Las APIs permiten que diferentes aplicaciones se comuniquen entre sí y comparten datos.

Backend : es la parte de una aplicación que se encarga de la lógica de negocio y el acceso a los datos. El backend se ejecuta en un servidor y se comunica con el frontend a través de APIs.

Frontend : es la parte de una aplicación que se encarga de la interfaz de usuario y la interacción con el usuario. El frontend se ejecuta en el navegador del usuario y se comunica con el backend a través de APIs.

SSL : es un protocolo de seguridad que se utiliza para establecer una conexión segura entre un servidor y un cliente. SSL cifra los datos que se envían entre el servidor y el cliente, lo que protege la información sensible de ser interceptada por terceros.

HTTPS : es una versión segura de HTTP. HTTPS utiliza SSL para cifrar los datos que se envían entre el servidor y el cliente, lo que protege la información sensible de ser interceptada por terceros.

UI : es la interfaz de usuario. Se refiere a la parte de una aplicación con la que el usuario interactúa. La UI incluye elementos como botones, menús y formularios.

UX : es la experiencia del usuario. Se refiere a la forma en que un usuario interactúa con una aplicación y cómo se siente al hacerlo. La UX incluye aspectos como la usabilidad, la accesibilidad y la satisfacción del usuario.

JWT : es un estándar abierto que define un formato compacto y autónomo para transmitir información de forma segura entre partes como un objeto JSON. Esta información puede ser verificada y confiable porque está firmada digitalmente.

SSO : es un proceso de autenticación que permite a un usuario acceder a múltiples aplicaciones con una sola sesión de inicio de sesión. SSO simplifica la gestión de credenciales y mejora la experiencia del usuario al reducir la necesidad de recordar múltiples contraseñas.

REST : es un estilo arquitectónico para diseñar servicios web. REST se basa en el uso de HTTP y utiliza los métodos HTTP (GET, POST, PUT, DELETE) para realizar operaciones sobre recursos.

GraphQL : es un lenguaje de consulta para APIs y un entorno de ejecución para ejecutar esas consultas con los datos existentes. GraphQL permite a los clientes solicitar solo los datos que necesitan, lo que reduce la cantidad de datos transferidos entre el cliente y el servidor.

gRPC : es un marco de trabajo de código abierto que permite la comunicación entre aplicaciones distribuidas. gRPC utiliza HTTP/2 para la comunicación y Protocol Buffers como formato de serialización de datos.

Reverse Proxy : es un servidor que actúa como intermediario entre los clientes y uno o más servidores de backend. El reverse proxy recibe las solicitudes de los clientes y las reenvía a los servidores de backend, lo que permite distribuir la carga y mejorar la seguridad.

Cron Job : es una tarea programada que se ejecuta automáticamente en un servidor en intervalos regulares. Los cron jobs se utilizan para realizar tareas de mantenimiento, como copias de seguridad, limpieza de registros y actualizaciones de datos.