



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

TempusUGR - Sistema de gestión personalizada de horarios académicos para la Universidad de Granada

Realizado por
Juan Miguel Acosta Ortega

ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



**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, 25 de mayo de 2025

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

Informa:

Que el presente trabajo, titulado *xxxxxx*, 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, 25 de mayo 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.	El problema	3
2.2.	La solución	3
2.3.	Restricciones	3
2.4.	Objetivos	4
2.4.1.	Objetivo Principal	4
2.4.2.	Objetivos Generales	4
2.4.3.	Objetivos Específicos	4
3	Estado del arte	6
3.1.	Contextualización	6
3.2.	Visualización y gestión de horarios académicos en la UGR	7
3.3.	Análisis comparativo de sistemas de planificación personalizada en educación superior	10
3.4.	Desarrollo de servicios web	13
3.4.1.	Arquitecturas de software	16
3.4.2.	Tecnologías de desarrollo	20
3.4.3.	Sistemas de almacenamiento de datos	23
3.4.4.	Autenticación y autorización	24
3.4.5.	Comunicación entre servicios	25
3.4.6.	Despliegue de sistemas	27
3.4.7.	Pruebas y calidad del software	28
4	Especificación de requisitos	30
4.1.	Personas	30
4.1.1.	Personas del sistema	30
4.2.	Escenarios	31
4.2.1.	Escenarios del sistema	32
4.3.	Historias de usuario	33
4.3.1.	Estructura de una historia de usuario	34
4.3.2.	Historias de usuario	35
4.4.	Requisitos funcionales	45
4.4.1.	Gestión de usuarios	45
4.4.2.	Gestión de horarios académicos	46
4.5.	Requisitos no funcionales	47
4.5.1.	Rendimiento	47
4.5.2.	Usabilidad	47
4.5.3.	Seguridad	47
4.5.4.	Mantenibilidad	48

4.5.5.	Portabilidad	48
4.5.6.	Disponibilidad	48
4.6.	Requisitos de información	48
4.6.1.	Servicio de usuarios	49
4.6.2.	Servicio de horarios	49
4.6.3.	Servicio de suscripciones académicas	50
4.7.	Validación de los requisitos	51
5	Planificación del proyecto	52
5.1.	Cronograma del proyecto	52
5.2.	Metodología de desarrollo	53
5.2.1.	Roles y Responsabilidades en este Proyecto	53
5.2.2.	Proceso Scrum Implementado	54
5.2.3.	Justificación de la Metodología	54
5.2.4.	Gestión de Tareas y Seguimiento del Progreso	55
5.3.	Gestión de riesgos	58
5.4.	Presupuesto del proyecto	59
5.4.1.	Presupuesto en formato tabla	59
5.4.2.	Desglose de la información	59
6	Diseño del sistema y tecnologías escogidas	62
6.1.	Arquitectura del sistema	62
6.1.1.	Arquitectura de microservicios	62
6.1.2.	Tecnologías y Frameworks	62
6.1.3.	Diseño de la base de datos	62
6.1.4.	Diseño de la API	62
6.2.	Diseño de la Interfaz de Usuario (UI) y la Experiencia del Usuario (UX)	62
7	Implementación	63
7.1.	Iteración 0	63
7.2.	Iteración 1	63
7.3.	Iteración 2	63
7.4.	Iteración 3	63
7.5.	Iteración 4	64
7.6.	Iteración 5	64
8	Despliegue del sistema	65
8.1.	Configuración del servidor	65
8.2.	Contenerización del sistema	66
8.2.1.	Pasos para la contenerización del backend	66
8.2.2.	Docker Compose	68
8.2.3.	Pasos para la contenerización del frontend	69
8.3.	Levantar el sistema	72
8.3.1.	Paso del HTTP a HTTPS en las llamadas al backend	72
8.3.2.	Pruebas de carga en un entorno real	73

9 Conclusiones y trabajos futuros	79
9.1. Evaluación del proyecto	79
9.2. Dificultades y resolución	79
9.3. Mejoras posibles y trabajos futuros	79
Bibliografía	80
Anexo A: Glosario	82

Índice de figuras

3.1. Comparación de horarios de diferentes grados: ETSIIT (arriba), ADE (centro) y Doble Grado en Nutrición Humana y Dietética (abajo). . .	8
3.2. Horario de la asignatura Bases Químicas de la Biología, impartida en el Grado en Biología.	9
3.3. Aplicación móvil de la Universidad de Almería (UAL App).	11
3.4. Aplicación My Study Life.	12
4.1. Persona 1: Alumno de la UGR	30
4.2. Persona 2: Profesor de la UGR	31
4.3. Persona 3: "Administrador" de la UGR	31
5.1. Gantt del proyecto.	52
5.2. Ejemplo de historia de usuario en Github Projects.	56
5.3. Tablero del 2º Sprint durante su desarrollo.	56
5.4. Segmento del "Roadmap" del proyecto.	57
5.5. Resumen de horas de desarrollo en CLockify.	57
6.1. Logo de TempusUGR	62
8.1. Gráfica de la prueba de carga con 100 usuarios concurrentes.	74
8.2. Gráfica de la prueba de carga con 500 usuarios concurrentes.	76
8.3. Gráfica de la prueba de carga con 1000 usuarios concurrentes.	77

Índice de tablas

4.1. Estructura de una historia de usuario	35
4.2. Historia de usuario HU-1	35
4.3. Historia de usuario HU-2	36
4.4. Historia de usuario HU-3	36
4.5. Historia de usuario HU-4	37
4.6. Historia de usuario HU-5	37
4.7. Historia de usuario HU-6	37
4.8. Historia de usuario HU-7	38
4.9. Historia de usuario HU-8	38
4.10. Historia de usuario HU-9	38
4.11. Historia de usuario HU-10	39
4.12. Historia de usuario HU-11	39
4.13. Historia de usuario HU-12	40
4.14. Historia de usuario HU-13	40
4.15. Historia de usuario HU-14	41
4.16. Historia de usuario HU-15	42
4.17. Historia de usuario HU-16	43
4.18. Historia de usuario HU-17	43
4.19. Historia de usuario HU-18	44
4.20. Historia de usuario HU-19	44
4.21. Historia de usuario HU-20	45
5.1. Resumen de gastos elegibles del proyecto.	59
8.1. Resultados de la prueba de carga con 100 usuarios concurrentes. . . .	74
8.2. Resultados de la prueba de carga con 500 usuarios concurrentes. . . .	76
8.3. Resultados de la prueba de carga con 1000 usuarios concurrentes. . .	77

1. Introducción

La arquitectura de microservicios ha emergido como un paradigma fundamental en el desarrollo de sistemas complejos y escalables. Su enfoque modular y distribuido ofrece ventajas claras en términos de flexibilidad, mantenibilidad y despliegue independiente. Sin embargo, la implementación exitosa de una arquitectura de microservicios conlleva una serie de desafíos técnicos que resultan atractivos desde una perspectiva de aprendizaje y desarrollo profesional.

Este proyecto de fin de grado se presenta como una oportunidad perfecta para explorar de manera profunda los entresijos de esta arquitectura. Conceptos como la comunicación eficiente entre servicios (síncrona y asíncrona), la implementación de un API Gateway robusto para la gestión centralizada de peticiones, enrutamiento, seguridad ... , la aplicación de mecanismos de seguridad en un entorno distribuido, y las estrategias de balanceo de carga para garantizar la disponibilidad del sistema, representan áreas de gran interés práctico y teórico.

El desarrollo de un sistema de calendario personalizado para la Universidad de Granada, basado en una arquitectura de microservicios, permitirá no solo consolidar conocimientos teóricos, sino también enfrentarse a los retos inherentes a la construcción de un sistema distribuido real. La necesidad de diseñar APIs claras y bien definidas, gestionar la persistencia de datos en múltiples servicios, asegurar la consistencia y la tolerancia a fallos, y monitorizar el rendimiento del sistema en su conjunto, proporcionarán una experiencia de aprendizaje invaluable. En definitiva, este proyecto se concibe como un vehículo para comprender y aplicar de manera práctica los principios y las mejores prácticas asociadas a la arquitectura de microservicios, un campo con una alta demanda en la industria del desarrollo de software.

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 la motivación del trabajo y la estructura de la memoria.
- **Capítulo 2: Descripción del problema.** En este capítulo se describe el problema que se va a resolver en el trabajo.
- **Capítulo 3: Estado del arte.** En este capítulo se presenta un resumen del estado del arte tanto en el ámbito de sistemas de gestión de horarios como en el ámbito de sistemas de información basados en web.
- **Capítulo 4: Especificación de requisitos.** En este capítulo se presentan las personas, escenarios, y los requisitos del sistema en forma de historias de

usuario y requisitos funcionales, no funcionales, y de información.

- **Capítulo 5: Planificación.** En este capítulo se presenta la planificación temporal de trabajo, la metodología de desarrollo escogida y su implementación. Además se presenta un presupuesto del trabajo.
- **Capítulo 6: Diseño.** En este capítulo se presenta el diseño de la arquitectura de microservicios, de la API REST Implementada, de las bases de datos y del frontend desarrollado.
- **Capítulo 7: Implementación.** En este capítulo se presenta la implementación del sistema dividido en los sprints realizados.
- **Capítulo 8: Despliegue.** En este capítulo se presenta el despliegue del sistema en el servidor de la UGR con lo que todo ello conlleva (gestión del servidor, SSL, HTTPS, dominio, etc.).
- **Capítulo 9: Conclusiones.** En este capítulo se presentan las conclusiones y trabajos futuros planteados.
- **Anexo A: Glosario.** En este anexo se presenta un glosario con las definiciones de términos técnicos utilizados a lo largo del trabajo.

2. Descripción del problema

A continuación se presenta la descripción del problema que se va a resolver en este trabajo de fin de grado, así como los objetivos y restricciones del sistema a desarrollar.

2.1. El problema

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. La solución

Para introducir estas mejoras, 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.

2.3. Restricciones

El sistema debe cumplir con las siguientes restricciones:

- **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.

2.4. Objetivos

2.4.1. 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.

2.4.2. 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.

2.4.3. 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. Poniendo de ejemplo a la ETSIT, debemos acceder a la página oficial de la facultad [1] 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, 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 muy poco eficiente, ya que el estudiante tiene que buscar la información de manera manual, 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	ALEM	FP	FP	FP	FP	FP	FP	FP
10:30-11:30	CA	CA	FP	FP	FP	FP	FP	FP	FP
11:30-12:30	CA (A1)	FP (A2)	FP (A2)	FP (A2)	FP (A2)	FP (A2)	FP (A2)	FP (A2)	FP (A2)
12:30-13:30	ALEM (A1)	FP (A2)	FP (A2)	FP (A2)	FP (A2)	FP (A2)	FP (A2)	FP (A2)	FP (A2)
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 (E20)	MAT (D03) (E20)		FDAE	
9:30 a 10:30	MAT (E20)	MAT (D03) (E20)		FDAE	
10:30 a 11:30	IOF	IOF	IMK (D03) (D25)	MAT (D03)	
11:30 a 12:30	IOF	IOF	IMK (D03) (D25)	MAT (D03)	
12:30 a 13:30	EP (D03) (E20)	EP (D03) (E20)	FDAE	IMK (D03) (D25)	
13:30 a 14:30	EP (D03) (E20)	EP (D03) (E20)	FDAE	IMK (D03) (D25)	

Grado en Nutrición Humana y Dietética (2º)									
Asignatura	Créditos	Tipología	Horario	Septiembre	Octubre	Noviembre	Diciembre	Enero	Febrero
Bioquímica Metabólica (BQM)	1	FB	1,5	8:30 h 11:30 h 16:00 h	15 23 30	7 14 21 28	4 11 18 25	9 16 23 30	6 13 20 27
Fisiología Humana (FH)	1	FB	1,5	8:30 h 11:30 h 16:00 h				1,2 1,2 3,4 3,4	
Microbiología (M)	1	Ob	1,5	8:30 h 11:30 h 16:00 h	1 3 2 4				
Nutrición I (N1)	1	FB	1,5	8:30 h 11:30 h 16:00 h		1 3 2 4 5			
Tecnología Culinar (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					
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 [2] 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 [3] 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 periodos de actividad docente, exámenes finales, periodos de evaluación de convocatorias ... se ha de acceder a la web de la Secretaría General en la UGR [4] 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 periodos 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. 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

Hora \ Día	LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES
7:00					
8:00					
9:00	Grupo	Grupo	Grupo	Grupo	Grupo
10:00	Grupo: 7	Grupo: 5	Grupo: 7	Grupo: 5	Grupo: 7
11:00	Grupo: 7	Grupo: 5	Grupo: 7	Grupo: 5	Grupo: 7
12:00	Gru	Gru	Gru	Gru	Gru
13:00					
14:00					
15:00					
16:00	Grupo: 7	Grupo: 5	Grupo: 3	Grupo: 1	Grupo: 3
17:00					
18:00	Grupo: 6	Grupo: 6	Grupo: 7	Grupo: 7	Grupo: 7
19:00					
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.

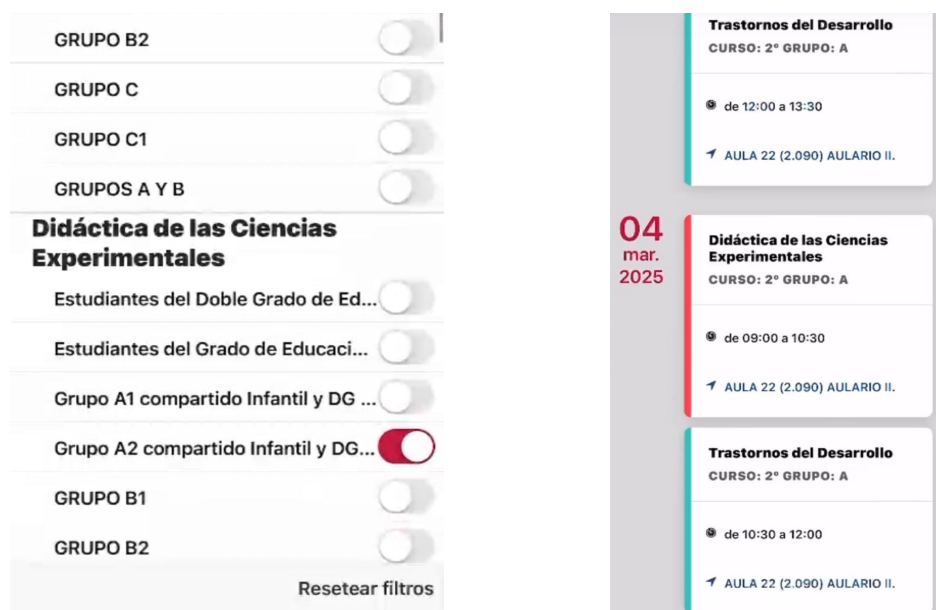


Figura 3.3: Aplicación móvil de la Universidad de Almería (UAL App).

Exponiendo un ejemplo, la Universidad de Almería (UAL) ha implementado en su aplicación móvil multiplataforma “UAL App”, 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' [5], una aplicación multiplataforma que permite a los estudiantes gestionar sus horarios académicos, tareas y exámenes de manera integrada. 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 es 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.

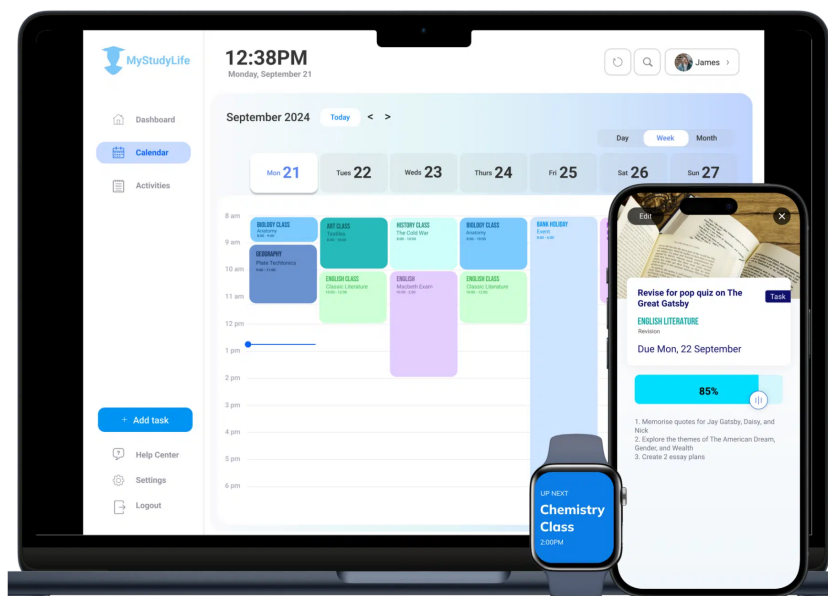


Figura 3.4: Aplicación My Study Life.

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 [6] o Microsoft Outlook [7]. 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 [8] 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 tres tecnologías las que se han estandarizado como las más utilizadas para la creación de APIs: REST, 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. **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.

3. **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)** 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 [9]:** 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 [10] 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)

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 desplegados 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)

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)

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 Basada en el Espacio (SBA - Space-Based Architecture) y Principios Nativos de la Nube

Descripción (SBA): Conocida también como arquitectura de tuplas espaciales, la SBA está diseñada para abordar problemas de alta escalabilidad y concurrencia, minimizando los cuellos de botella asociados a las bases de datos centralizadas. En este modelo, la lógica de negocio y los datos se particionan y se distribuyen en múltiples unidades de procesamiento (PUs) auto-suficientes. Cada PU generalmente contiene la aplicación, una caché de datos en memoria y, potencialmente, su propio motor de base de datos. La coordinación y comunicación se realizan a través de un “espacio de tuplas” compartido o una caché distribuida.

Principios Nativos de la Nube (Cloud-Native): Más que una arquitectura específica, es un enfoque para diseñar y construir aplicaciones que explotan plenamente las ventajas del modelo de computación en la nube. Esto incluye el uso de contenedores (por ejemplo, Docker), orquestación (por ejemplo, Kubernetes), diseño para la resiliencia (circuit breakers, reintentos), observabilidad y automatización extensiva (CI/CD).

Posicionamiento y Evolución: SBA ofrece una solución para escenarios de escalabilidad extrema. Los principios nativos de la nube son fundamentales en el estado del arte actual, proporcionando el ecosistema y las herramientas para desplegar, gestionar y escalar eficazmente arquitecturas distribuidas

modernas, incluyendo prominentemente los microservicios y sistemas basados en EDA.

Arquitectura de Microservicios

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** 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):** 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:** .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:** 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:** 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, el **JavaScript** 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)** 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** 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 trío fundamental del desarrollo frontend.

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

1. **React:** 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:** 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:** 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, PostgreSQL y Microsoft SQL Server. 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), bases de datos clave-valor (Redis), bases de datos en columna (Cassandra) y bases de datos orientadas a grafos (Neo4j). 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.

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, Google Cloud SQL y Azure Cosmos DB.

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 y Google Cloud Spanner.

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:** 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:** 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:** 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):** Este método combina múltiples factores de autenticación (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:** 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**
- **gRPC**
- **GraphQL**
- **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:** 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:** 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:** 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:** 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:** 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:** 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:** 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:** 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 rendimiento:** Estas pruebas evalúan la capacidad del sistema para manejar cargas de trabajo específicas y medir su rendimiento bajo diferentes condiciones. Las pruebas de rendimiento son esenciales para garantizar que el sistema sea escalable y responda adecuadamente a las demandas de los usuarios. Herramientas populares para pruebas de rendimiento incluyen JMeter, Gatling y LoadRunner.
- **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.

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

4.1. 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...

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

4.1.1. Personas del sistema

PERSONA #1: MANUEL RODRÍGUEZ JIMÉNEZ



DEMOGRAFÍA

Edad: 21
Género: Hombre
Profesion: 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.

Bio: Estudiante de Ingeniería Informática en la UGR. Está en tercer año aunque se encuentra recuperando asignaturas de primero y de segundo.

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: Mujeres
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

4.2. 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.

4.2.1. 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 tableros 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.

4.3. 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. [11].

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.

4.3.1. 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

4.3.2. 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

4.4. Requisitos funcionales

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

4.4.1. 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.

4.4.2. 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.

4.5. Requisitos no funcionales

4.5.1. 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.

4.5.2. 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.

4.5.3. 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).

4.5.4. 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.

4.5.5. 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.

4.5.6. 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.

4.6. Requisitos de información

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

4.6.1. 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.

4.6.2. 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.

4.6.3. 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.

4.7. 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 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**[12], que permite la creación de diagramas de Gantt de manera sencilla y efectiva. A continuación, se presenta el diagrama de Gantt del proyecto, que muestra las diferentes fases y tareas a realizar en cada sprint.

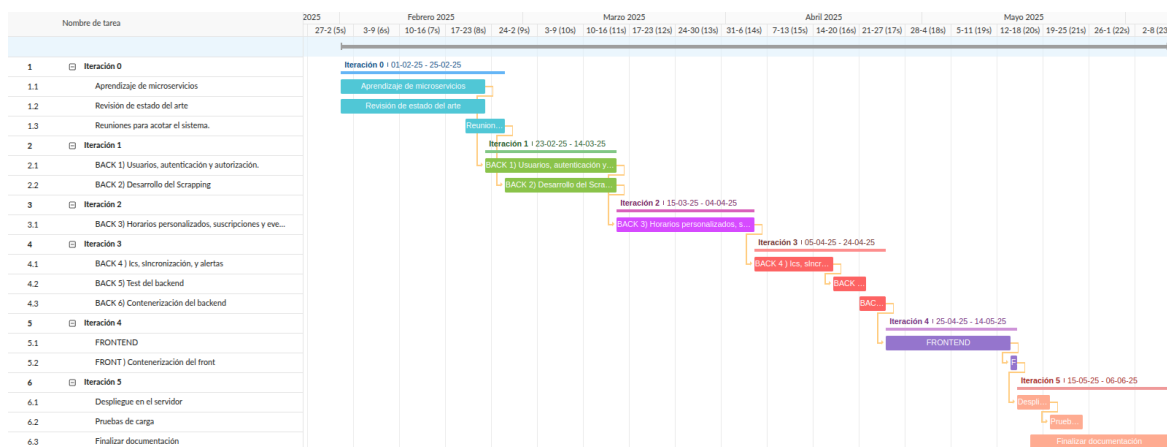


Figura 5.1: Gantt del proyecto.

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**[13]. 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 conllevaba una serie de criterios de aceptación que se han ido marcando a medida que se iban cumpliendo.



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 iteración.

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”).

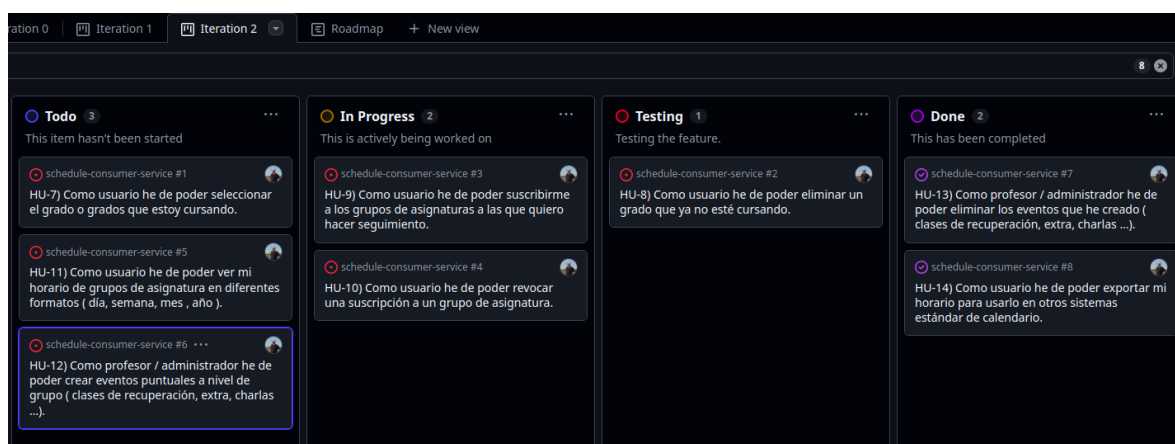


Figura 5.3: Tablero del 2º Sprint durante su desarrollo.

- **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.

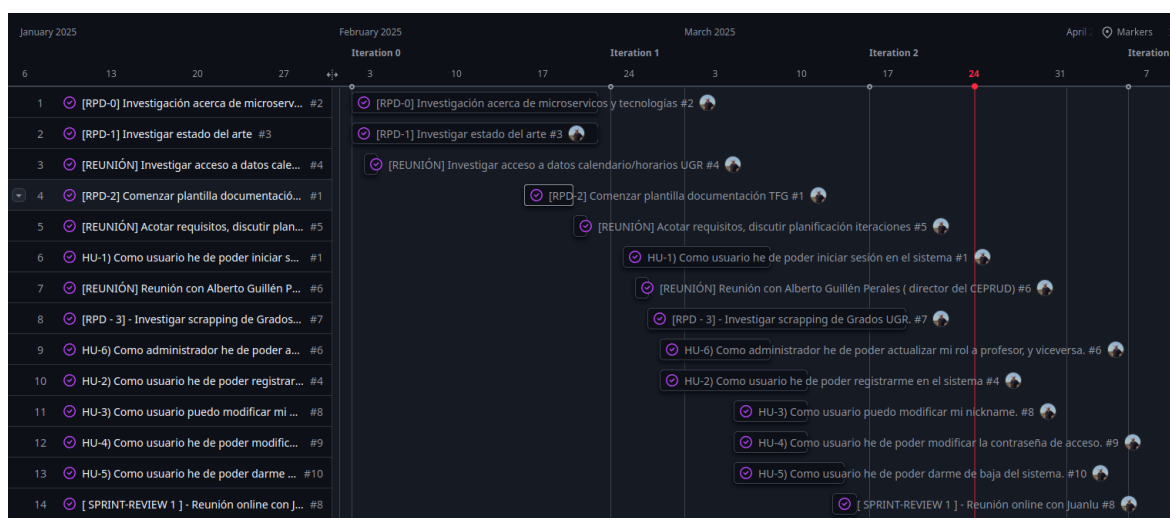


Figura 5.4: Segmento del "Roadmap" del proyecto.

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[14] es una herramienta de seguimiento del tiempo que permite registrar el tiempo dedicado a cada tarea y generar informes sobre el progreso del proyecto. 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.

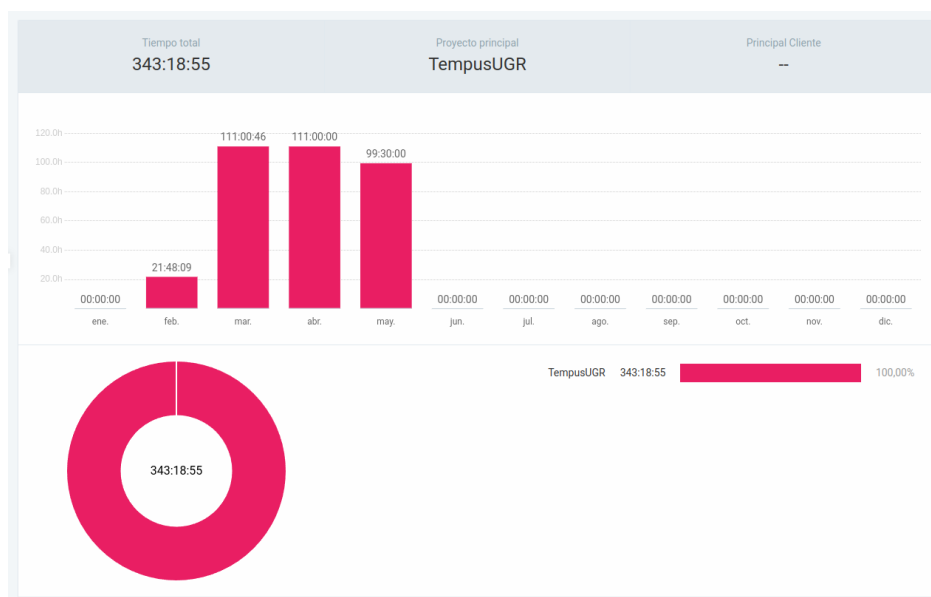


Figura 5.5: Resumen de horas de desarrollo en CLockify.

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 en la iteración 0, se han dedicado un total de 376.5 horas al desarrollo del proyecto.

5.3. 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 la 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 cambiaran 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.

5.4. Presupuesto del proyecto

****Revisar este apartado y detallar más si necesario** El presupuesto del proyecto se ha elaborado teniendo en cuenta los gastos elegibles y no elegibles, así como los costes asociados al desarrollo y despliegue del sistema. A continuación se presenta un resumen de los gastos elegibles, que son aquellos que se consideran necesarios para la realización del proyecto y que cumplen con los requisitos establecidos por la normativa de financiación.

5.4.1. Presupuesto en formato tabla

Gstos elegibles	Importe (€)
Gastos de personal	5.647,5
Gstos de suministros (desarrollo)	200
Gastos de suministros (despliegue)	140
Total	5.987,5

Tabla 5.1: Resumen de gastos elegibles del proyecto.

5.4.2. Desglose de la información

Gastos elegibles

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

Tarifa horaria aplicada: Aunque la plataforma Glassdoor [15] 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 horas × 15 €/hora = 5.647,5 €
- Coste mensual estimado (4 meses): 5.647,5 € ÷ 4 = 1.411,88 €/mes

Gastos de suministros 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: $30 \text{ €} \times 4 \text{ meses} = 120 \text{ €}$

■ **Electricidad:**

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

Coste total de suministros: 120 € (Internet) + 80 € (Electricidad) = **200 €**

Durante el despliegue Durante la fase de despliegue del proyecto, se ha considerado el uso de una máquina dedicada que actuará como servidor para alojar tanto el backend como el frontend de forma permanente.

Especificaciones del servidor:

- Sistema operativo: Ubuntu 22.04 LTS
- CPU: Intel Core i5 (4 núcleos)
- RAM: 16 GB
- Almacenamiento: SSD 512 GB
- Conectividad: Red de alta velocidad (Ethernet 1 Gbps)

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

Costes asociados al despliegue:

■ **Electricidad:** Funcionamiento continuo del servidor.

- Estimación mensual: 25 €/mes
- Periodo considerado: 4 meses
- Total: $25 \text{ €} \times 4 = 100 \text{ €}$

■ **Conectividad:** Uso intensivo de red para servir peticiones de clientes.

- Estimación mensual adicional: 10 €/mes
- Total: $10 \text{ €} \times 4 = 40 \text{ €}$

Coste total durante el despliegue: 100 € (electricidad) + 40 € (conectividad) =
140 €

6. Diseño del sistema y tecnologías escogidas

6.1. Arquitectura del sistema

6.1.1. Arquitectura de microservicios

6.1.2. Tecnologías y Frameworks

6.1.3. Diseño de la base de datos

6.1.4. Diseño de la API

6.2. Diseño de la Interfaz de Usuario (UI) y la Experiencia del Usuario (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, 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: #e4afae 
- Color de fondo: #f5f5f5 
- Color de texto: #333333 



Figura 6.1: Logo de TempusUGR

7. Implementación

7.1. Iteración 0

Curso de microservicios en el ecosistema de Spring Boot, reuniones con el director del TFG para definir el sistema. Reuniones con el secretario de la facultad, con el funcionariado de la ETSIT, y con el director del CEPRUD para conocer la información con la que cuento. Pruebas en Spring Boot.

7.2. Iteración 1

Se implementaron todas las historias de usuario relacionadas con la gestión de usuarios y roles, así como la autenticación y autorización. Para ello se implementaron los servicios user-service, auth-service, mail-service y el api-gateway. Además se hizo una primera aproximación del scrapping de los datos de horarios académicos de todos los grados de la ugr.

7.3. Iteración 2

Se afina el scrapping, se crea el servicio schedule-consumer-service, y se crea la lógica de suscripciones a grupos, y demás tareas relacionadas. Además se puede extraer el ".ics" de los horarios personalizados, para ello se implementa el academic-subscription-service.

Se crea además el servidor de descubrimiento de servicios con eureka, y así se investiga como mejorar el rendimiento con balanceo de carga y varias instancias.

7.4. Iteración 3

Fin del backend generando eventos a nivel de grupo y a nivel de facultad. Extracción del ics con clases oficiales, clases extra y eventos de facultad. Sincronización con Google calendar. Alertas cuando se crean eventos a nivel de grupo (clases extra). Se crean pruebas unitarias y se realizan pruebas de carga en el servidor de despliegue. Se dockeriza el sistema y se implementa en un servidor de producción.

7.5. Iteración 4

Comienzo del frontend, se implementan las mismas historias de usuario que en el backend. Refinamiento del backend, añadidos para complementar el frontend.

7.6. Iteración 5

Se acaba el frontend. Se termina la documentación del proyecto. Se comienza la presentación del TFG.

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.

****Revisar este párrafo y detallar más si necesario** El servidor en el que se despliega el sistema se encuentra en el edificio auxiliar de la ETSIIT. Fue solicitado por el director del TFG, Juan Luis Jiménez Laredo, y facilitado por el CSIRC (Servicio de Ciberseguridad y Respuesta a Incidentes Informáticos) de la UGR. El servidor cuenta con las siguientes características:

Además se solicitó una IP estática para el servidor, que se ha configurado para que apunte al dominio `tempus.ugr.es`. Esto se ha conseguido gracias a la colaboración del CSIRC, que ha configurado el DNS para que apunte al servidor y al dominio.

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
           8761:8761 eureka-service
```

```
2
```

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-
           entrypoint-initdb.d/init.sql \
6             --name mysql \
7             mysql:8.0.41
```

```
8
```

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         GRANT ALL PRIVILEGES ON DB_SCHEDULE_CONSUMER_SERVICE.* TO '
           calendarugr'@' %';
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 *:443>
2          ServerName tempus.ugr.es
3          ServerAlias xxx.xx.xxx.xxx
4
5          # Directorio raiz donde se encuentra tu aplicacion
Angular
6          DocumentRoot /var/www/html
7
8          # Habilitar SSL
9          SSLEngine on
10         SSLCertificateFile /ejemplo/de/ruta/certificado.pem
11         SSLCertificateKeyFile /ejemplo/de/ruta/clave-privada
.pem
12
13         # Configuracion de cifrados seguros
14         SSLCipherSuite "HIGH:MEDIUM:!MD5:!RC4:!3DES"
15         SSLHonorCipherOrder on
16
17         # Protocolos seguros
18         SSLProtocol -all +TLSv1.2 +TLSv1.3
19
20         SSLProxyEngine On
21         SSLProxyProtocol -all +TLSv1.2
22         SSLProxyCheckPeerCN off
23         SSLProxyCheckPeerName off
24
25         # PROXY PARA EL API GATEWAY
26         ProxyPass /calendarugr/v1 http://api-gateway:8090/
calendarugr/v1
27         ProxyPassReverse /calendarugr/v1 http://api-gateway:
8090/calendarugr/v1
28
29         <Directory /var/www/html>
30             Options Indexes FollowSymLinks
31             AllowOverride All
32             Require all granted
33         </Directory>
34
35         ErrorLog ${APACHE_LOG_DIR}/mi-app-error.log
36         CustomLog ${APACHE_LOG_DIR}/mi-app-access.log
combined
37     </VirtualHost>
38

```

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      # Configuración 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     # Composición 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 [16], 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 simularon 100 usuarios concurrentes realizando solicitudes al sistema durante 2 minutos. Los resultados obtenidos fueron los siguientes:

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.

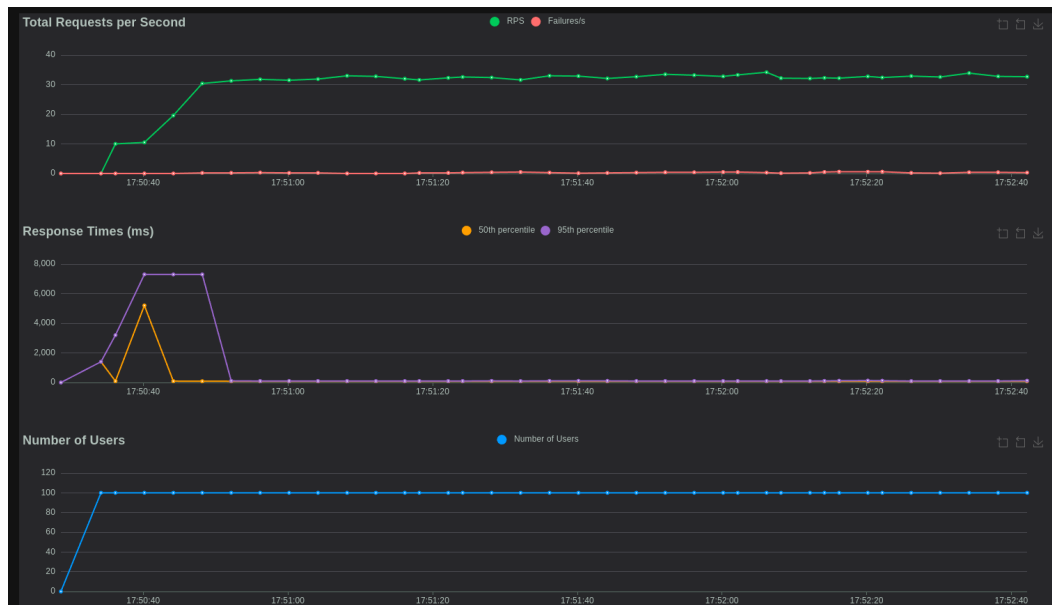


Figura 8.1: Gráfica de la prueba de carga con 100 usuarios concurrentes.

Se ha observado en la gráfica de “Response Times (ms)” un pico inicial de latencia significativa. Aproximadamente entre las 17:30:00 y 17:40:00, el percentil 95 alcanzó hasta 7300 ms (7 segundos) y el percentil 50 (mediana) llegó a unos 5200 ms (5 segundos). Posteriormente, los tiempos de respuesta se estabilizaron rápidamente, lo que es consistente con las estadísticas globales de la prueba, que muestran una mediana de 84 ms y un percentil 95 de 120 ms para el total de la ejecución.

Este comportamiento es típico de un “arranque en frío” o “cold start” del sistema, que puede involucrar varios factores:

- **Carga inicial de código:** Al inicio de la prueba, el servidor de aplicaciones necesita cargar el código en memoria, inicializar módulos, frameworks y establecer las primeras conexiones a la base de datos. Este proceso inicial consume tiempo.
- **Calentamiento de cachés:** Las aplicaciones web y las bases de datos (PostgreSQL) utilizan cachés. En las primeras solicitudes, estas cachés están vacías, y los datos deben ser recuperados de fuentes más lentas (disco o base

de datos), lo que aumenta la latencia. Conforme más solicitudes llegan, las cachés se llenan, mejorando los tiempos de respuesta.

- **Pool de conexiones:** La creación y llenado del pool de conexiones a la base de datos es una operación relativamente costosa que ocurre al inicio.
- **Menor concurrencia inicial:** A pesar de un rápido incremento a 100 usuarios en la gráfica, las primeras solicitudes pueden impactar un sistema que aún no ha alcanzado su estado “caliente”.

En resumen, estos picos iniciales son un comportamiento esperado de “calentamiento” del sistema. Una vez que las cachés se llenan y las conexiones se establecen, el sistema opera de manera más eficiente, resultando en tiempos de respuesta significativamente menores.

En cuanto a los fallos registrados, una tasa de 35 fallos sobre un total de 4172 solicitudes es ****extremadamente baja****, representando una tasa de éxito del 99.20 %. Esto se considera, en la mayoría de los casos, despreciable y no indicativo de un problema sistémico grave. Las posibles causas para estos fallos aislados incluyen:

- **Fallos de red transitorios:** Problemas momentáneos en la conectividad entre el generador de carga y el servidor, o entre el servidor y la base de datos.
- **Timeouts:** Es posible que algunas solicitudes excedieran el tiempo máximo de espera configurado, especialmente durante el período de “cold start” donde las latencias eran muy altas.
- **Problemas puntuales del servidor/base de datos:** Eventos raros y breves como micro-reinicios, reinicios de conexión o mantenimiento puntual que afectaron solo a esas solicitudes.
- **Errores del generador de carga:** En ocasiones, el propio software de pruebas puede reportar fallos incorrectamente debido a problemas internos.

Dado el bajo número de fallos, no se justifica una investigación profunda sobre su origen, ya que es probable que sean anomalías aleatorias o efectos del proceso de inicialización del sistema. La estadística “Current Failures/s: 0.5” indica la tasa de fallos en un momento específico, no contradiciendo el total de fallos de la prueba.

Además el fallo registrado es en todos los casos el mismo, “RemoteDisconnected(‘Remote end closed connection without response’)”, lo que indica que el servidor cerró la conexión antes de enviar una respuesta. Esto puede deberse a un timeout o a un cierre inesperado de la conexión por parte del servidor, pero no necesariamente indica un problema grave en el sistema.

En resumen, los resultados de la prueba de carga con 100 usuarios concurrentes muestran que el sistema es capaz de manejar una carga moderada de usuarios sin problemas significativos, con tiempos de respuesta aceptables y una tasa de éxito muy alta.

Pruebas de carga con 500 usuarios concurrentes

En esta prueba se simularon 500 usuarios concurrentes realizando solicitudes al sistema durante 2 minutos. Los resultados obtenidos fueron los siguientes:

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.

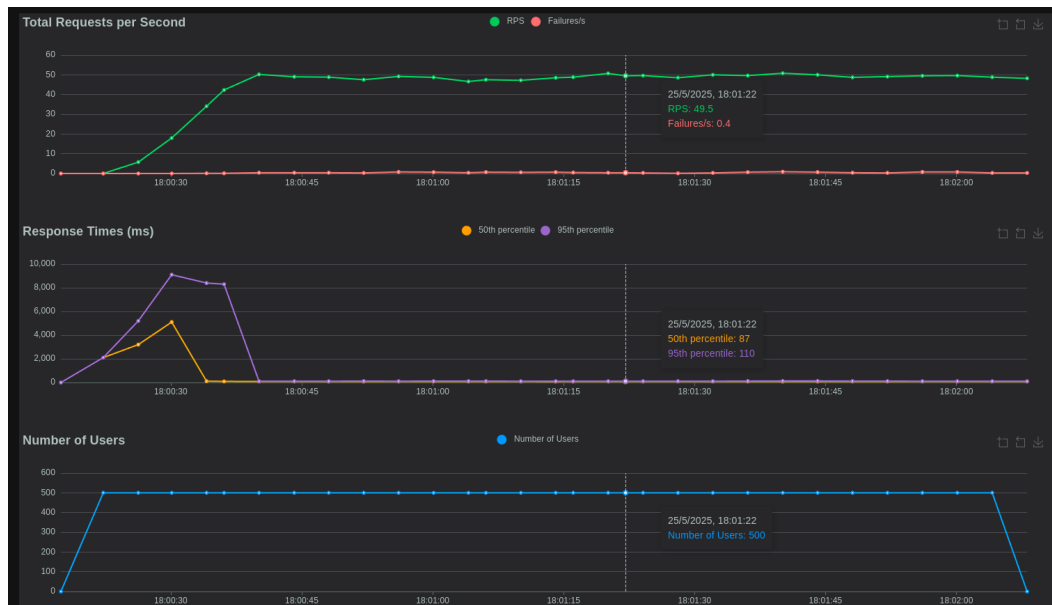


Figura 8.2: Gráfica de la prueba de carga con 500 usuarios concurrentes.

En esta prueba se observan comportamientos similares a los de la prueba con 100 usuarios concurrentes, pero con una latencia más alta en general. Esto es esperado, ya que al aumentar el número de usuarios concurrentes, el sistema debe manejar más solicitudes al mismo tiempo, lo que puede aumentar la latencia.

En la gráfica de “Response Times (ms)” se observa un pico de latencia significativa al inicio de la prueba, similar al de la prueba con 100 usuarios concurrentes. En este caso, el percentil 95 alcanzó hasta 9100 ms (9 segundos) y el percentil 50 (mediana) llegó a unos 5100 ms (5 segundos). Posteriormente, los tiempos de respuesta se estabilizaron, lo que es consistente con las estadísticas globales de la prueba, que muestran una mediana de 88 ms y un percentil 95 de 280 ms para el total de la ejecución.

Además en cuanto a los fallos registrados, una tasa de 48 fallos sobre un total de 4918 solicitudes es también muy baja, representando una tasa de éxito del 99.02 %. Esto indica que el sistema es capaz de manejar una carga moderada de usuarios sin problemas significativos, con tiempos de respuesta aceptables y una tasa de éxito muy alta.

En resumen, los resultados de la prueba de carga con 500 usuarios concurrentes muestran que el sistema es capaz de manejar una carga moderada de usuarios sin problemas significativos, con tiempos de respuesta aceptables y una tasa de éxito muy alta. Aunque se observan picos de latencia al inicio de la prueba, estos son esperados y no indican un problema grave en el sistema.

Pruebas de carga con 1000 usuarios concurrentes

En esta prueba se simularon 1000 usuarios concurrentes realizando solicitudes al sistema durante 2 minutos. Los resultados obtenidos fueron los siguientes:

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.

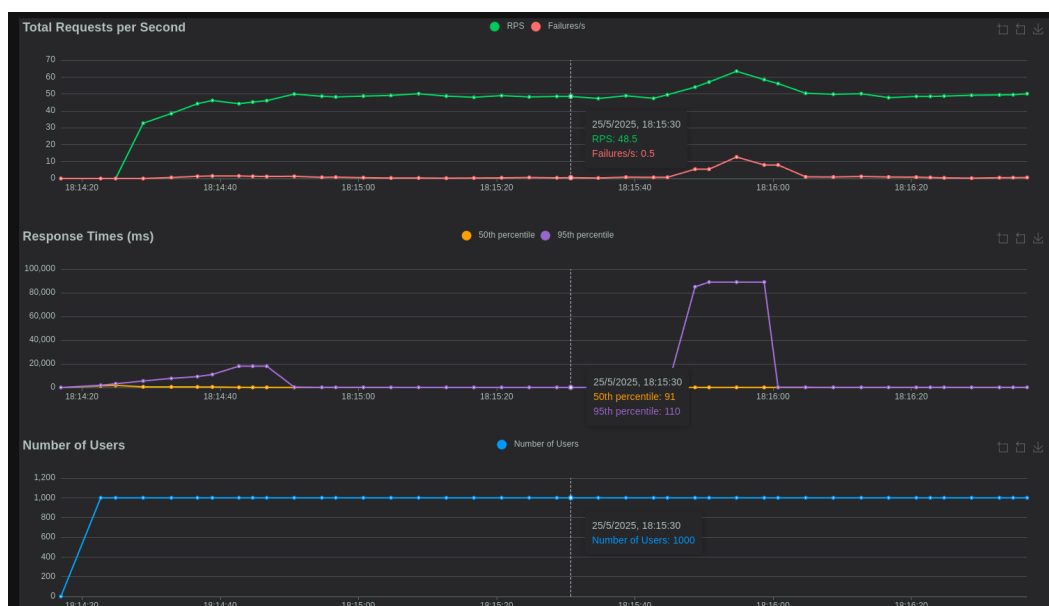


Figura 8.3: Gráfica de la prueba de carga con 1000 usuarios concurrentes.

En este caso se observa un comportamiento similar al de las pruebas anteriores, pero con una latencia aún más alta en general. Esto es esperado, ya que al aumentar el número de usuarios concurrentes, el sistema debe manejar más solicitudes al mismo tiempo, lo que puede aumentar la latencia.

En la gráfica de “Response Times (ms)” se observa un pico de latencia significativa al inicio de la prueba, similar al de las pruebas anteriores. En este caso, el percentil 95 alcanzó hasta 18000 ms (18 segundos) y el percentil 50 (mediana) sin embargo no es tan alto como el de las otras pruebas, llegando a un pico de 140ms al principio. Posteriormente, los tiempos de respuesta se estabilizaron, lo que es consistente con las estadísticas globales de la prueba, que muestran una mediana

de 92 ms y un percentil 95 de 2500 ms para el total de la ejecución.

Además, en cuanto a los fallos registrados, una tasa de 215 fallos sobre un total de 6672 solicitudes es también baja, representando una tasa de éxito del 96.78 %. Aunque esta tasa es inferior a las pruebas anteriores, sigue siendo aceptable y no indica un problema grave en el sistema.

En esta prueba además se ha observado un comportamiento anómalo alrededor de los 90 segundos de la prueba, donde el percentil 99 alcanzó hasta 89000 ms (89 segundos). Esto indica que hubo un problema puntual en el sistema que afectó a algunas solicitudes, pero no a todas. Este tipo de comportamiento puede deberse a un problema de rendimiento en el servidor o en la base de datos, o a un problema de red.

La “recuperación” no se observa en el sentido de que los tiempos de respuesta o fallos vuelvan a niveles óptimos durante esta fase. Más bien, el sistema entra en un estado de alta latencia y fallos continuos bajo esta carga sostenida. La estabilización en el gráfico de tiempos de respuesta a partir de los 80 segundos no es una recuperación positiva, sino una indicación de que el sistema se ha atascado en un estado de rendimiento muy bajo.

Las causas de esta saturación con 1000 usuarios concurrentes pueden ser:

- **Saturación de recursos del servidor:** Límites de CPU, memoria o procesos en el servicio de aplicación.
- **Cuellos de botella en la base de datos:** Agotamiento del pool de conexiones, contención de bloqueos, o límites de recursos (CPU, RAM, IOPS) en PostgreSQL.
- **Timeouts:** Los tiempos de respuesta excesivos probablemente provocan que las solicitudes excedan los límites de tiempo establecidos, registrándose como fallos.
- **Límites de autoescalado:** Si el autoescalado no es suficientemente rápido o los límites del plan de servicio son insuficientes para 1000 usuarios.

9. Conclusiones y trabajos futuros

9.1. Evaluación del proyecto

9.2. Dificultades y resolución

9.3. Mejoras posibles y trabajos futuros

Bibliografía

- [1] 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.
- [2] Universidad de Granada. Página principal de grados ugr, 2025. URL <https://grados.ugr.es/>. Accedido el 23 de octubre de 2024.
- [3] 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.
- [4] 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.
- [5] My Study Life. Página principal de my study life, 2025. URL <https://mystudylife.com/>. Accedido el 12 de noviembre de 2024.
- [6] Google. Página principal de google calendar, 2025. URL <https://calendar.google.com/>. Accedido el 12 de noviembre de 2024.
- [7] Microsoft. Página principal de microsoft outlook, 2025. URL <https://outlook.live.com/>. Accedido el 12 de noviembre de 2024.
- [8] Moodle. Página principal de moodle, 2025. URL <https://moodle.org/>. Accedido el 12 de noviembre de 2024.
- [9] 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.
- [10] Neal Ford, Mark Richards, Pramod Sadalage, and Zhamak Dehghani. *Software Architecture: The Hard Parts*. O'Reilly Media, 2021. ISBN 978-1098109424.
- [11] Atlassian. Historias de usuario, 2024. URL <https://www.atlassian.com/es/agile/project-management/user-stories>. Accedido el 17 de diciembre de 2024.
- [12] GanttPRO. Ganttpro: Software de diagramas de gantt online, 2025. URL <https://app.ganttpro.com/>. Accedido el 17 de mayo de 2025.
- [13] CalendarUGR. Repositorio de calendarugr en github, 2025. URL <https://github.com/CalendarUGR>. Accedido el 24 de mayo de 2025.
- [14] Clockify. Página principal de clockify, 2025. URL <https://clockify.me/>. Accedido el 12 de noviembre de 2024.

- [15] 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 mayo de 2025.
- [16] Locust. Locust - scalable user load testing tool, 2025. URL <https://locust.io/>. Accedido el 25 de mayo de 2025.

Anexo A: 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 compartan 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.