Texto

El contenido generado por IA puede ser incorrecto.Imagen que contiene Logotipo

El contenido generado por IA puede ser incorrecto.

Grado en Ingeniería del Software

Desarrollo de una aplicación web progresiva sobre la gestión de trabajos de fin de grado en universidades (Development of a progressive web application on the management of end-of-degree projects in universities)

Realizadopor

David Bueno Carmona

Tutorizado por

Eduardo Guzmán de los Riscos

Departamento

(NOMBRE DEL DEPARTAMENTO)

UNIVERSIDAD DE MÁLAGA

MÁLAGA, septiembre 2025

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Grado en Ingeniería del Software

**Desarrollo de una aplicación web progresiva sobre la gestión de trabajos de fin de grado en universidades**

**Development of a progressive web application on the management of end-of-degree projects in universities**

Realizado por

**David Bueno Carmona**

Tutorizado por

**Eduardo Guzmán de los Riscos**

Departamento

**[Nombre del departamento]**

UNIVERSIDAD DE MÁLAGA

MÁLAGA, JUNIO DE 2025

Fecha defensa: X de Septiembre de 2025

Resumen

[Texto Resumen].

Palabras clave:

Abstract

[Resumen en inglés]

Keywords:

Índice

[Resumen 1](#_Toc204855944)

[Abstract 1](#_Toc204855945)

[Índice 1](#_Toc204855946)

[Índice de Figuras 1](#_Toc204855947)

[Introducción 3](#_Toc204855948)

[1.1. Motivación 3](#_Toc204855949)

[1.2. Objetivos 4](#_Toc204855950)

[1.3. Antecedentes 5](#_Toc204855951)

[1.4. Metodología 6](#_Toc204855952)

[1.4.1. Sprints 7](#_Toc204855953)

[1.5. Organización del documento 14](#_Toc204855954)

[Tecnologías y Herramientas 15](#_Toc204855955)

[2.1. Base de Datos 15](#_Toc204855956)

[2.2. Servidor de Backend 16](#_Toc204855957)

[2.3. Aplicación de Frontend 16](#_Toc204855958)

[Especificación y analisis 17](#_Toc204855959)

[3.1. Gestión de riesgos 17](#_Toc204855960)

[3.2. Análisis de requisitos 20](#_Toc204855961)

[3.2.1. Historias de usuario - Requisitos funcionales 20](#_Toc204855962)

[3.2.2. Historias de usuario - Requisitos no funcionales 23](#_Toc204855963)

[3.3. Casos de uso 24](#_Toc204855964)

[Referencias 26](#_Toc204855965)

[Apéndice A. Manual de Instalación 27](#_Toc204855966)

[A.1. Requerimientos 27](#_Toc204855967)

[Apéndice B. Otros Recursos 28](#_Toc204855968)

Índice de Figuras

[Fig. 1. Título de la Primera Figura 25](#_Toc204855969)

Índice de tablas

[Tabla 1. Riesgos de la seguridad de la información 20](#_Toc204856150)

[Tabla 2. Riesgos Técnicos 20](#_Toc204856151)

Introducción

## Motivación

La gestión de los Trabajos de Fin de Grado en muchas universidades es un proceso que involucra múltiples actores y tareas administrativas que, cuando no están correctamente coordinadas, pueden derivar en ineficiencias, errores y frustración para estudiantes, profesores y personal de administración.

Tradicionalmente, la asignación de TFGs, la selección de tutores, la formación de grupos y la entrega de documentos se han gestionado a través de correos electrónicos, hojas de cálculo o formularios impresos. Esta falta de integración provoca pérdida de información, duplicidades, y sobrecarga administrativa. Además, los sistemas existentes en algunas universidades suelen ser cerrados, de difícil acceso para los estudiantes, o limitados en cuanto a personalización.

TFGInfo nace como una respuesta a esta problemática, proponiendo una solución web centralizada, accesible desde cualquier dispositivo, y adaptada a las necesidades específicas de los centros educativos. Su desarrollo también representa una oportunidad para aplicar conocimientos adquiridos en el grado en Ingeniería Informática, tales como diseño de bases de datos, desarrollo web full-stack, seguridad, y experiencia de usuario.

Desde una perspectiva académica y profesional, el proyecto busca resolver una necesidad real en el ámbito universitario, al tiempo que constituye una valiosa experiencia de desarrollo de software completo, desde el análisis de requisitos hasta la implementación y pruebas. Además, TFGInfo pone especial énfasis en la modularidad y extensibilidad, de forma que pueda ser evolucionado en el futuro con nuevas funcionalidades, integraciones o adaptaciones a otras instituciones.

## Objetivos

**Diseñar e implementar una aplicación web completa para gestionar TFGs**

El objetivo principal del proyecto es desarrollar una solución tecnológica integral que abarque tanto el backend como el frontend. La aplicación debe permitir gestionar el ciclo de vida completo de un Trabajo de Fin de Grado, incluyendo la creación, asignación, seguimiento y finalización de este. Esta implementación incluye el modelado de la base de datos, el desarrollo de servicios mediante una API Rest, y la construcción de una interfaz de usuario moderna, intuitiva y responsive.

**Ofrecer una interfaz intuitiva tanto para administradores como para profesores y alumnos**

La experiencia de usuario es fundamental para asegurar la adopción de la herramienta por parte de los distintos perfiles que interactúan con el sistema. Por ello, se ha planteado un diseño centrado en el usuario (UX), donde cada tipo de usuario accede únicamente a las funcionalidades relevantes para su rol, minimizando la complejidad y evitando sobrecargas de información. Los alumnos pueden consultar líneas de TFG disponibles, los profesores pueden gestionar sus propuestas y los administradores pueden supervisar y validar todo el proceso.

**Integrar roles y permisos diferenciados**

El sistema debe contemplar una gestión de usuarios basada en roles que limite el acceso a la información y funcionalidades según el perfil del usuario. Los roles definidos son: **Administrador**, **Profesor** y Estudiante. Esta distinción es clave para mantener la seguridad, la integridad de los datos y garantizar que cada usuario interactúe con el sistema de forma adecuada a su función institucional.

**Garantizar la integridad de los datos**

Uno de los objetivos técnicos esenciales del proyecto es asegurar que la información almacenada sea coherente y no se produzcan inconsistencias en la base de datos. Esto se logra mediante el uso de claves foráneas y validaciones tanto en el backend como en el frontend.

**Facilitar el trabajo colaborativo mediante grupos de trabajo**

El sistema permite la creación y gestión de grupos de trabajo que pueden estar formados por uno o varios alumnos y profesores. Esta funcionalidad permite reflejar la realidad de muchos TFGs que se desarrollan en entornos colaborativos, interdepartamentales o interinstitucionales. Además, fomenta la transparencia y la comunicación entre todos los implicados en un proyecto de fin de grado.

## Antecedentes

La necesidad de gestionar Trabajos de Fin de Grado de forma organizada y eficiente ha impulsado el desarrollo de diferentes plataformas en el ámbito universitario. A continuación, se analizan algunas de las herramientas más conocidas y utilizadas, así como sus limitaciones en el contexto en el que se enmarca este proyecto.

**SIGMA**

Es una de las soluciones más extendidas en universidades españolas. Se trata de un sistema de gestión académica muy completo que abarca desde la matriculación hasta la gestión de titulaciones, expedientes y TFGs. Sin embargo, SIGMA presenta ciertas desventajas:

* Su implementación suele requerir acuerdos institucionales complejos y costosos.
* Tiene una curva de aprendizaje pronunciada, especialmente para usuarios no técnicos.
* No es fácilmente adaptable a necesidades específicas de departamentos o facultades concretas.
* Carece de flexibilidad para integrar nuevos flujos de trabajo sin intervención del proveedor.

**Universitas XXI**

Esta plataforma también es muy potente y utilizada en muchas instituciones de educación superior. Su foco está en la gestión administrativa de las universidades en su conjunto, incluyendo módulos para TFGs. A pesar de sus capacidades:

* Suele estar muy centralizado y controlado por las áreas de administración, lo que limita la autonomía de usuarios como profesores y estudiantes.
* La personalización e incorporación de nuevas funcionalidades no es sencilla.
* En muchos casos, su uso se limita a consultas pasivas, sin interacción fluida entre los distintos actores.

**Soluciones caseras: hojas de cálculo, formularios y correo electrónico**

En muchos departamentos, especialmente aquellos sin acceso a plataformas institucionales robustas, la gestión de TFGs se realiza mediante hojas de cálculo compartidas (por ejemplo, en Google Sheets), formularios en línea (Google Forms, Microsoft Forms), o incluso por intercambio de correos electrónicos. Esta aproximación es ágil al principio, pero presenta grandes limitaciones:

* No hay control de acceso ni gestión de roles.
* Se generan errores frecuentes por edición simultánea o pérdida de datos.

- Es difícil escalar el sistema cuando aumenta el volumen de TFGs o el número de usuarios.

**¿Qué aporta TFGInfo frente a estas alternativas?**

TFGInfo nace con el objetivo de situarse en un punto intermedio entre las soluciones corporativas sobredimensionadas y las soluciones improvisadas. Su propuesta de valor incluye:

* **Ligereza y portabilidad**: al ser una aplicación web modular y abierta, puede desplegarse en servidores propios o en la nube sin dependencia de terceros.
* **Acceso multiplataforma**: funciona desde navegadores modernos, sin necesidad de instalaciones adicionales.
* **Gestión por roles**: incorpora una gestión de permisos desde el diseño, asegurando que cada usuario accede solo a las funcionalidades que le corresponden.
* **Personalización**: puede adaptarse fácilmente a las particularidades de cada facultad, titulación o plan de estudios.

En resumen, TFGInfo pretende cubrir el vacío existente entre las soluciones rígidas institucionales y las herramientas improvisadas, proporcionando una plataforma equilibrada, eficaz y accesible para la gestión académica del TFG.

## Metodología

Para realizar este proyecto se ha aplicado una metodología incremental e iterativa basada en los principios ágiles. La finalidad es dividir el trabajo en iteraciones o sprints ya que en el desarrollo de una aplicación existen etapas muy diferenciables. Se ha elegido esta metodología por las ventajas que aporta:

**Flexibilidad ante cambios**

Durante el desarrollo de un sistema de gestión como TFGInfo, es común que los requisitos evolucionen o se refinen a medida que se avanza. La metodología ágil permite adaptar el rumbo del proyecto sin penalizar el avance previo, facilitando la incorporación de nuevas funcionalidades, cambios en la lógica de negocio o mejoras en la interfaz.

**Descomposición en entregas pequeñas**

Esta metodología propone dividir el trabajo en unidades más manejables (iteraciones o sprints), lo que permite centrarse en objetivos concretos a corto plazo. Esto hace que el progreso sea visible, medible y fácil de gestionar. En el caso de TFGInfo, esto se tradujo en diseñar la estructura de la aplicación realizando el backend, luego funcionalidades básicas, y más tarde componentes visuales avanzados.

**Retroalimentación temprana**

Aunque se trata de un proyecto académico, la posibilidad de obtener feedback de profesores, compañeros o incluso usuarios reales (como estudiantes y docentes) durante el desarrollo puede enriquecer el resultado. Con esta metodología, este feedback puede incorporarse entre iteraciones sin necesidad de rehacer todo el trabajo.

**Reducción de riesgos**

Al ir construyendo y validando el sistema de forma incremental, se identifican errores y cuellos de botella más rápido. Esto reduce el riesgo de que un fallo grave se detecte demasiado tarde. Por ejemplo, si un endpoint presenta problemas de rendimiento o si una vista es confusa, puede corregirse en la siguiente iteración.

**Mejor planificación y control del tiempo**

En el contexto de un TFG, donde hay una fecha límite clara, Una metodología ágil permite planificar de forma realista las tareas según el tiempo disponible. Se priorizan las funcionalidades esenciales primero y se dejan los extras para fases finales, evitando que el proyecto quede incompleto por falta de tiempo.

**Entrega de valor desde el principio**

Incluso si el proyecto no se completa en su totalidad, al final de cada iteración se tiene una versión funcional del sistema. Esto garantiza que siempre haya una “versión demostrable” del proyecto, algo crucial para presentaciones, tutorías o revisiones intermedias.

**Orientación a objetivos claros y medibles**

Se favorece la definición de metas concretas por iteración. Esto ayuda a mantener el enfoque y a evitar desviaciones. En el caso de TFGInfo, se traduce en tareas bien definidas como “crear endpoint de búsqueda de TFGs” o “diseñar formulario de alta de alumno”.

### Sprints

**Sprint 1: Análisis y planificación del sistema**

**Objetivo:**

Definir las bases del proyecto TFGInfo, analizando los requisitos funcionales y no funcionales, los posibles riesgos, y estableciendo una estructura sólida a nivel conceptual y de datos. Este sprint representa el punto de partida del proyecto, marcando la dirección técnica y organizativa del desarrollo.

**Tareas**:

**Identificación de requisitos:**

En esta fase se recopilaron y documentaron las necesidades que debía cubrir la aplicación. Se realizó un análisis de los diferentes perfiles de usuario (administrador, profesor, alumno), sus interacciones con el sistema, y los procesos clave que debía soportar la plataforma, como la gestión de TFGs, la creación de grupos, la asignación de líneas temáticas y la autenticación basada en roles.  
Se elaboraron casos de uso, historias de usuario y se identificaron requisitos funcionales (qué debe hacer el sistema) y no funcionales (rendimiento, seguridad, accesibilidad, etc.).

**Identificación de riesgos**

En esta tarea se evaluaron los factores que podrían comprometer el desarrollo o el éxito del proyecto. Algunos de los riesgos detectados fueron:

Complejidad de las relaciones entre entidades (TFGs, usuarios, grupos, departamentos).

Posibles errores al importar datos desde fuentes externas (CSV).

Dificultades en la sincronización entre frontend y backend.

**Diseño y modelado de la aplicación**

Se construyó un diagrama de clases que representa las entidades principales del sistema y sus relaciones. Este diseño sirvió como base tanto para la implementación de la base de datos como para la arquitectura del backend.  
Se optó por un enfoque modular, donde cada componente (usuarios, TFGs, grupos, departamentos) se trató como una unidad autónoma con su propia lógica y controlador. Además, se definieron los flujos de interacción entre componentes y los puntos clave de acceso (API REST).

**Diseño del modelo y de la base de datos**

Basándose en el modelado anterior, se diseñó una base de datos relacional utilizando MySQL. Se definieron las tablas necesarias, sus atributos, claves primarias y foráneas, así como las relaciones entre ellas.  
Especial atención se puso en las relaciones muchos a muchos, por ejemplo, entre profesores y TFGs, o entre alumnos y grupos de trabajo. Se diseñaron tablas intermedias para estas relaciones, y se aplicaron restricciones de integridad referencial para asegurar la coherencia de los datos.  
También se preparó un conjunto inicial de datos para realizar pruebas, incluyendo scripts para crear la base de datos.

**Entregables:**

**Documento de requisitos y casos de uso**

Incluye la descripción de cada funcionalidad del sistema, los actores implicados, los escenarios principales, y los requerimientos técnicos y operativos. Sirve como referencia para todos los sprints posteriores.

**Diagrama de clases y esquema de base de datos relacional**

El diagrama de clases permite visualizar la estructura del sistema desde un punto de vista lógico, mientras que el esquema relacional detalla cómo se implementan estas relaciones en la base de datos. Ambos documentos son fundamentales para los desarrolladores.

**Lista priorizada de funcionalidades (Product Backlog)**

Se definió un backlog inicial con todas las funcionalidades necesarias ordenadas por prioridad y complejidad. Esto permitió estructurar los siguientes sprints de forma eficiente, garantizando que se abordaran primero las funcionalidades más críticas para el funcionamiento básico del sistema.

**Sprint 2: Preparación de entorno y estructura base**

**Objetivo:**

Establecer la base técnica mínima viable del sistema, configurando tanto el entorno de desarrollo como los primeros componentes funcionales del backend. Este sprint sienta las bases para poder comenzar el desarrollo de funcionalidades reales, garantizando que la infraestructura esté correctamente integrada y probada.

**Tareas:**  
**Extracción de los datos e integración en la base de datos.**  
Esta tarea incluyó:

* Revisión y limpieza de los datos: detección de inconsistencias, caracteres extraños, errores tipográficos y formatos incorrectos.
* Creación de scripts SQL para importar los datos a la base de datos MySQL de forma ordenada.
* Normalización de los datos: separación en tablas relacionadas, asignación de claves foráneas y eliminación de redundancias.
* Verificación de la integridad referencial y relaciones entre entidades (por ejemplo, cada alumno debe pertenecer a una carrera válida).

**Creación de un back-end .NET**

Se configuró un proyecto **ASP.NET Core** como base del backend del sistema. Esta tarea incluyó:

* Creación del proyecto desde Visual Studio y configuración del entorno de desarrollo.
* Instalación de dependencias necesarias.
* Definición del DbContext y clases de entidades que reflejan el modelo de datos diseñado en el sprint anterior.
* Configuración del archivo appsettings.json con los parámetros de conexión a la base de datos MySQL.
* Estructuración del proyecto en carpetas claras: Models, Controllers, Services, DTOs, etc.

**Entregables:**

**Proyecto ASP.NET Core configurado**

Se entrega un proyecto backend funcional con la arquitectura base ya creada. Está listo para que se añadan controladores, lógica de negocio y autenticación en los siguientes sprints.

**Conexión y prueba de base de datos MySQL**

El backend ya está conectado a la base de datos local (o remota) y es capaz de lanzar migraciones automáticas, ejecutar consultas básicas y manipular datos con Entity Framework.

**Datos de prueba cargados correctamente desde CSV**

Gracias a los scripts desarrollados, la base de datos contiene información realista que permite probar la lógica del sistema sin necesidad de ingresar datos manualmente.

**Pruebas básicas de acceso a datos (API mínima)**

Se desarrollaron endpoints iniciales de prueba para validar que las operaciones básicas (GET, POST, PUT, DELETE) funcionan correctamente. Estas pruebas demostraron que la API puede leer entidades desde la base de datos y devolverlas en formato JSON, lo cual confirma que la infraestructura está correctamente conectada y operativa.

**Sprint 3: Desarrollo del backend funcional**

**Objetivo:**

Implementar los controladores principales de la API REST, desarrollando la lógica de negocio asociada a cada uno de los módulos clave del sistema. Este sprint marca el paso de una infraestructura técnica básica a un sistema con funcionalidades específicas para la gestión de TFGs, usuarios y recursos académicos.

**Tareas:**  
**Desarrollo de controladores y endpoints**

Se implementaron los controladores principales, cada uno encargado de gestionar un conjunto de recursos y operaciones críticas del sistema. Se usó el patrón RESTful, asegurando rutas claras, consistentes y seguras, utilizando métodos HTTP adecuados (GET, POST, PUT, DELETE). Entre ellos se encuentran:

**Controlador de Usuarios:**

Este controlador es el encargado de gestionar la parte de autenticación en la aplicación. Tiene funcionalidades como, realizar el inicio de sesión, realizar el registro de un usuario nuevo, hacer las comprobaciones sobre el token de autenticación, conceder roles a los usuarios y comprobar que acciones están permitidas a los usuarios según su rol.

**Controlador de Reservas:**

Este controlador se centra en las acciones relacionadas con las reservas de un TFG, como la solicitud de un TFG por parte del alumno, la aceptación o denegación del TFG por parte del profesor, el envío de correos notificando las actualizaciones en el TFG, el cambio de estado que tiene un proyecto, entre otras funcionalidades.

**Controlador de Líneas de TFG:**

Gestiona las operaciones relacionadas con los Trabajos de Fin de Grado como podrían ser: La creación y modificación de líneas de TFGs, Asignación de tutores a las líneas, asignación de carreras que pueden optar a hacer el TFG en cuestión, etc.

**Otros controladores:**

Existe un controlador por cada entidad de la base de datos con la lógica de la que se encarga cada uno. La lógica básica que tienen todos son un CRUD, y las entidades que tienen controladores son: Alumnos, Profesores, Departamentos, Titulaciones, Centros, Líneas de TFG, Instancias de TFG y Canales de trabajo.

**Entregables:**

**API REST funcional con rutas bien definidas**

La estructura de rutas sigue convenciones REST claras, como:

* api/usuarios/{id}
* api/reservas/solicitar
* api/departamentos

Esto facilita el uso de la API por parte del frontend y su documentación.

**Lógica de negocio modular y testeada**

Cada controlador está vinculado a un servicio de negocio independiente (por ejemplo, UserService, TFGService), lo que mejora la organización del código y facilita su testeo.

**Validaciones y manejo de errores implementados**

Todos los controladores integran validaciones a nivel de entrada (campos requeridos, formatos, relaciones válidas) y gestión de errores con respuestas claras en formato JSON. Se implementaron respuestas estandarizadas (200 OK, 400 Bad Request, 404 Not Found, 500 Internal Server Error) que facilitan la depuración y la experiencia de desarrollo del frontend.

**Sprint 4: Desarrollo del frontend e integración inicial**

**Objetivo:**

Construir la interfaz de usuario utilizando el framework Angular, creando componentes reutilizables y dinámicos, formularios interactivos y vistas con capacidades de búsqueda. Este sprint también incluyó la primera integración real entre el frontend y la API desarrollada en sprints anteriores.

**Tareas:**

**Creación del front-end en Angular**

Se inicializó el proyecto Angular utilizando Angular CLI, definiendo una estructura clara y modular de carpetas:

* components para los elementos reutilizables (formularios, listas, botones)
* services para las conexiones HTTP con el backend
* models para las interfaces de datos (DTOs)
* pages para las vistas principales del sistema (dashboard de reservas, gestión de TFGs, usuarios, etc.)

También se configuraron las rutas principales mediante el módulo de enrutamiento de Angular (RouterModule) y se integraron servicios HTTP usando HttpClientModule.

**Desarrollo del cliente (formulario y vistas)**

Se construyeron formularios dinámicos y validados para que los usuarios puedan crear y editar recursos como TFGs, usuarios, departamentos y reservas.

Características implementadas:

* Validaciones en tiempo real (campos obligatorios, formatos, longitud mínima/máxima).
* Formularios reactivos con FormBuilder.
* Mensajes de error claros y estilizados para mejorar la experiencia del usuario.
* Asociación dinámica de campos dependientes (por ejemplo, seleccionar una carrera actualiza los TFGs disponibles).
* Formularios adaptados a cada rol: el alumno ve formularios distintos a los de un administrador.

**Desarrollo de vistas de búsqueda y filtros**

Se crearon vistas que permiten listar los elementos principales del sistema (TFGs, usuarios, reservas, departamentos), acompañadas de filtros dinámicos.  
Funcionalidades incluidas:

* Filtrado por campos como estado, nombre, carrera, departamento.
* Conexión directa con la API mediante servicios (UserService, TfgService, etc.), asegurando que los datos mostrados estén sincronizados con la base de datos en tiempo real.

**Entregables:**

**Componentes funcionales de Angular conectados al backend**

Se entregaron los primeros componentes funcionales conectados a los endpoints REST. Estos componentes permiten a los usuarios visualizar y gestionar información almacenada en la base de datos. Además, se implementó el sistema de carga de datos asincrónica (observables con RxJS).

**Formularios con validaciones**

Todos los formularios críticos fueron validados tanto en frontend como en backend. Esto garantiza que los datos introducidos cumplen con los requisitos del modelo de datos, y mejora la usabilidad al informar al usuario de forma clara sobre cualquier error.

**Páginas de búsqueda con filtros activos**

Las páginas de listado de TFGs, usuarios o reservas incluyen herramientas completas para buscar, filtrar y ordenar los resultados, lo que facilita una navegación eficiente y rápida por el sistema. Estas vistas están diseñadas para ser reutilizadas y adaptables, según el tipo de entidad que se esté gestionando.

**Sprint 5: Estilo, adaptación, pruebas y documentación**

**Objetivo:**

Finalizar la capa visual de la aplicación asegurando una experiencia de usuario cuidada, accesible y funcional en distintos dispositivos. Este sprint también se centró en verificar la estabilidad del sistema a través de pruebas exhaustivas y en generar la documentación necesaria para su uso, instalación y entrega final.

**Tareas:**  
**Desarrollo del cliente: estilos de la aplicación**

En esta tarea se trabajó en la presentación estética y la coherencia visual del sistema.  
Acciones realizadas:

* Definición de una paleta de colores corporativa consistente con el entorno académico.
* Aplicación de estilos CSS y SCSS personalizados para botones, formularios, menús y tablas.
* Incorporación de íconos y tipografías que mejoran la usabilidad.
* Mejora de la jerarquía visual de las vistas para facilitar la lectura y la navegación.
* Uso de Angular Material y clases personalizadas para mantener uniformidad.

El objetivo fue lograr una interfaz profesional, clara y visualmente agradable, sin sobrecargar al usuario con elementos innecesarios.

**Diseño adaptativo (responsive)**

Se adaptó toda la interfaz para que funcione correctamente en distintos dispositivos: ordenadores, tablets y móviles.  
Técnicas utilizadas:

* Uso de Flex Layout, Grid y Media Queries para adaptar el contenido a distintos tamaños de pantalla.
* Comprobación del comportamiento de los componentes en resoluciones pequeñas, reorganizando elementos cuando era necesario.
* Optimización de menús, formularios y botones para pantallas táctiles.
* Simplificación de vistas en modo móvil para mejorar el rendimiento y la legibilidad.

Gracias a estas mejoras, TFGInfo puede utilizarse desde casa, en la universidad o desde dispositivos móviles, lo que mejora la accesibilidad para todos los perfiles de usuario.

**Redactar la documentación final del proyecto**:

Manual de usuario: describe cómo usar la aplicación según el rol (navegación, creación de recursos, gestión de TFGs, etc.).

Manual de instalación: explica paso a paso cómo clonar el repositorio, configurar la base de datos, lanzar el backend y arrancar el frontend localmente. Incluye requisitos técnicos, comandos y posibles errores comunes.

**Entregables:**

**Interfaz visualmente cuidada y responsive**

El sistema cuenta con una interfaz moderna, agradable y perfectamente funcional en distintos tamaños de pantalla. Los elementos visuales están organizados y estilizados de forma coherente, cumpliendo estándares de usabilidad.

**Sistema probado desde múltiples perfiles de usuario**

Todas las funcionalidades han sido validadas desde el punto de vista de alumnos, profesores y administradores. Se confirmó que los permisos están correctamente aplicados y que cada usuario accede solo a las funciones que le corresponden.

**Manual de usuario e instalación redactado**

Se entregan ambos manuales con capturas, explicaciones paso a paso y lenguaje claro, facilitando tanto el uso del sistema como su despliegue en otros entornos o instituciones.

**Proyecto listo para presentación y entrega**

Al final del sprint, el proyecto se encuentra en un estado completamente funcional, probado, documentado y visualmente completo. Está preparado para su exposición ante el tribunal académico, así como para ser usado en entornos reales o ampliado en trabajos futuros.

## Organización del documento

TODO: Cuando el documento este acabado

Tecnologías y Herramientas

## Base de Datos

Para el desarrollo de TFGInfo se ha optado por una base de datos relacional, concretamente MySQL, una de las tecnologías SQL más consolidadas, robustas y ampliamente utilizadas en el ámbito académico y profesional.

El sistema gestiona una serie de entidades con relaciones complejas y jerárquicas como estudiantes, profesores, TFGs, departamentos, carreras, universidades y grupos, lo que hace que un modelo relacional estructurado sea especialmente adecuado.

En momentos previos de la elección de base de datos se han estudiado en un primer lugar que tipo de base de datos se iba a utilizar. Sobre este asunto se han tenido en consideración las principales características de los dos tipos de bases de datos principales, SQL y NoSQL. Para una aplicación que se basa mayoritariamente en las relaciones de sus entidades y el control de estas se ha optado por una base de datos SQL para garantizar la integridad de la información y de sus relaciones.

Entre las posibles bases de datos SQL se han contemplado para realizar este proyecto están MySql y PostgreSQL. Se ha optado por MySQL por la facilidad de instalación y la amplia documentación que existe al respecto.

La aplicación que se ha usado para la gestión de la base de datos es **MySQL Workbench**.

## Servidor de Backend

Para el desarrollo de una API para la aplicación TFGinfo, se han contemplado varios lenguajes de programación con sus respectivos Frameworks para la creación y gestión de APIs. Entre ellos Spring Framwork en Java, FastApi en Python y .NET para C#.

Entre estas opciones se ha optado por .NET para C# ya que aporta las siguientes características:

Una arquitectura robusta y modular que permite gestionar fácilmente funcionalidades como usuarios, TFGs, departamentos, reservas, etc., cada uno con su propia lógica bien encapsulada.

Alto rendimiento y eficacia, .NET ofrece un rendimiento superior gracias a su motor optimizado, compilación anticipada y bajo consumo de memoria. Esto se traduce en una API ágil, capaz de responder a muchas solicitudes concurrentes sin degradar el rendimiento, algo ideal para una plataforma multiusuario como TFGInfo.

Seguridad integrada, .NET ofrece una fácil gestión de la autenticación y autorización que resulta muy importante en una aplicación basada en roles (alumno, profesor y administrador)

Entity Framework es un mapeador objeto-relación que permite trabajar con entidades de forma sencilla segura y rápida. Proporciona facilidades para hacer consultas complejas.

## Aplicación de Frontend

En la actualidad las tecnologías más usadas para el desarrollo de aplicaciones web están Angular y React. Ambas con los mismos lenguajes de programación: Typescript, para la lógica que va a realizar un usuario de la aplicación. HTML, para la interfaz que va a ver el usuario en la web. Y por último SCSS, para los estilos que va a tener la interfaz.

Con esos rasgos comunes la principal diferencia es lo que aporta cada Framework. Finalmente se ha optado por usar Angular por las siguientes razones:

Se trata de un framework completo (no solo una biblioteca), lo que significa que incluye desde el routing hasta la gestión del estado, formularios, internacionalización, herramientas de testing y más, todo con una estructura clara y coherente.

Promueve buenas prácticas desde el inicio: uso de componentes, servicios, inyección de dependencias, tipado con TypeScript, y separación de responsabilidades.

Se integra perfectamente con APIs mediante HttpClient, ideal para consumir los endpoints expuestos por un backend en .NET.

Y por último las herramientas integradas que facilitan la creación de componentes, módulos, servicios, etc.

## Desarrollo del código

El desarrollo del proyecto TFGInfo ha requerido el uso de herramientas modernas que facilitan la escritura, organización, control y mantenimiento del código fuente. A continuación, se describen las principales:

**Visual Studio Code**

Visual Studio Code (VS Code) ha sido el entorno de desarrollo integrado (IDE) elegido para el desarrollo del frontend y del backend. Es una herramienta ligera, rápida, multiplataforma y altamente extensible mediante extensiones.

Se ha utilizado para:

* Editar archivos TypeScript, HTML y CSS en el frontend con Angular.
* Editar archivos C# y JSON en la configuración del backend con .NET.
* Gestionar proyectos mediante terminal integrada.
* Depurar código y realizar pruebas de forma eficiente.

**Git y GitHub**

Para el **control de versiones** se ha utilizado Git como sistema distribuido, junto con GitHub como plataforma de alojamiento y colaboración.

Git y GitHub han permitido:

* Controlar el historial de cambios del proyecto.
* Trabajar con ramas para implementar funcionalidades de forma ordenada.
* Documentar cada avance mediante mensajes de commit.
* Hacer copias seguras del repositorio y restaurar versiones anteriores en caso necesario.
* Gestionar versiones entregables.

## Creación de diagramas

Durante el desarrollo de TFGInfo, se ha utilizado **Mermaid**, una herramienta web basada en texto, para generar de forma rápida y estructurada distintos tipos de diagramas útiles en documentación técnica y de software.

¿Qué es Mermaid?

Mermaid es un lenguaje de marcado ligero que permite crear diagramas a partir de sintaxis textual. Está especialmente orientado a desarrolladores y documentadores técnicos que necesitan incluir diagramas dinámicos y fácilmente editables en sus proyectos.

Ventajas de usar Mermaid

* **Sintaxis simple**: Se escribe con un lenguaje similar a Markdown, sin necesidad de herramientas gráficas complejas.
* **Versatilidad**: Permite generar diagramas de flujo, de clases, de casos de uso, de secuencia, Gantt, entre otros.
* **Visualización inmediata**: Puede visualizarse en navegadores mediante plataformas como mermaid.live o integrarse en editores como Visual Studio Code o herramientas como Notion, Obsidian o GitHub.
* **Mantenimiento fácil**: Al tratarse de código de texto, los diagramas pueden versionarse junto al código fuente y modificarse rápidamente.

En este proyecto se ha usado Mermaid para representar:

* Diagramas de flujo de procesos clave (inicio de sesión, solicitud de TFG, etc.).
* Diagramas de secuencia que muestran la interacción entre usuarios y el sistema.
* Modelos de dominio y relaciones entre entidades.
* Casos de uso y escenarios funcionales.

Gracias a Mermaid, ha sido posible documentar gráficamente el comportamiento y la estructura de la aplicación de forma clara, reproducible y sin depender de herramientas de diseño externas.

Especificación y analisis

## Gestión de riesgos

El propósito de un análisis de riesgos es identificar, clasificar y mitigar los riesgos asociados al desarrollo e implementación de una aplicación centrada en la gestión de Trabajos de Fin de Grado (TFG). Esta herramienta maneja datos sensibles, opera en un entorno multiusuario y debe integrarse con procesos institucionales, lo que introduce múltiples desafíos.

**Riesgos de la seguridad de la información**

Uno de los riesgos más importantes en este tipo de sistemas es la seguridad de los datos personales y académicos. La aplicación trata con información sensible, como nombres, correos institucionales, roles académicos y la asignación de Trabajos de Fin de Grado.

|  |  |  |  |
| --- | --- | --- | --- |
| **Riesgo** | **Descripción** | **Impacto** | **Mitigación** |
| Acceso no autorizado | Usuarios sin permisos acceden a datos sensibles (TFGs, notas, perfiles) | Alto | Implementar autenticación por rol, tokens de acceso y cifrado de credenciales |
| Inyección SQL / XSS | Uso de formularios para ejecutar código malicioso | Alto | Mediante el uso de un ORM como Entity Framework para sanitizar datos |
| Pérdida de datos | Caída del servidor o corrupción de la base de datos | Muy alto | Copias de seguridad automáticas y redundancia |
| Exposición de datos personales | Mal manejo de datos de alumnos y profesores | Alto | Campos mínimos requeridos y acceso restringido |

Tabla 1. Riesgos de la seguridad de la información

**Riesgos técnicos**

A nivel técnico, uno de los principales retos está en la complejidad del modelo de datos. La aplicación debe gestionar relaciones múltiples y cruzadas entre entidades como TFGs, estudiantes, profesores, líneas de investigación y departamentos. Si el modelo no está bien diseñado, pueden surgir problemas de integridad y eficiencia.

|  |  |  |  |
| --- | --- | --- | --- |
| **Riesgo** | **Descripción** | **Impacto** | **Mitigación** |
| Complejidad de relaciones entre entidades | TFGs, alumnos, departamentos y profesores se relacionan de forma compleja | Medio | Modelado cuidadoso con diagramas ER y uso de claves foráneas |
| Fallos en la sincronización frontend-backend | Problemas de integración entre Angular y API .NET | Medio | Establecer comunicación usando objetos DTOs para asegurar la integridad |
| Falta de escalabilidad | Aumento de usuarios podría saturar la app | Medio | Usar buenas prácticas y arquitectura modular |

Tabla 2. Riesgos Técnicos

**Riesgos de gestión y planificación**

Como en cualquier proyecto de desarrollo, existe el riesgo de subestimar los tiempos necesarios para cada tarea. Una mala planificación puede llevar a retrasos importantes. También es posible que, durante el desarrollo, los requisitos del sistema cambien por decisiones del cliente o descubrimientos funcionales.

|  |  |  |  |
| --- | --- | --- | --- |
| **Riesgo** | **Descripción** | **Impacto** | **Mitigación** |
| Subestimación de tiempos | Tareas de desarrollo más largas de lo previsto | Medio | Planificación ágil (sprints), revisión frecuente de avances |
| Cambios en los requisitos | Los usuarios finales cambian su opinión sobre cómo debe funcionar la app | Medio | Reuniones regulares de revisión, diseño modular para adaptabilidad |
| Dependencia de datos de terceros | Problemas al importar datos desde CSV o sistemas externos | Alto | Validación previa de datos, limpieza automatizada de entradas |

**Riesgos organizativos y de adopción**

Aunque el sistema funcione correctamente, puede haber resistencia al cambio por parte del profesorado o personal administrativo. La falta de formación o el rechazo a abandonar métodos tradicionales (como hojas de cálculo o gestión por correo) puede obstaculizar la adopción de la herramienta.

|  |  |  |  |
| --- | --- | --- | --- |
| **Riesgo** | **Descripción** | **Impacto** | **Mitigación** |
| Resistencia al cambio | Profesores o administrativos rehúsan usar la plataforma | Medio | Formación básica, interfaz amigable y documentación clara |
| Falta de formación | Los usuarios no saben cómo usar correctamente el sistema | Medio | Incluir un manual de usuario detallado y videotutoriales |
| Incompatibilidad con sistemas existentes | Falta de integración con bases de datos u otras herramientas institucionales | Alto | Uso de formatos estándar (CSV, JSON), diseño API-first para integraciones futuras |

**Riesgos legales**

La aplicación debe cumplir con la legislación vigente en protección de datos, especialmente con el Reglamento General de Protección de Datos (GDPR) y la Ley Orgánica de Protección de Datos (LOPDGDD en España). Cualquier fallo en este aspecto, como almacenar datos personales sin consentimiento o no permitir su eliminación, puede tener consecuencias legales.

|  |  |  |  |
| --- | --- | --- | --- |
| **Riesgo** | **Descripción** | **Impacto** | **Mitigación** |
| Incumplimiento de la LOPD/GDPR | Almacenar datos personales sin base legal | Muy alto | Solicitar consentimiento explícito, limitar datos innecesarios, cifrado en reposo y en tránsito |

Desarrollar una aplicación de gestión universitaria implica gestionar riesgos técnicos, organizativos y legales. TFGInfo, como plataforma que maneja información académica y personal, debe diseñarse con una arquitectura segura, escalable y centrada en el usuario. La anticipación y mitigación de riesgos es clave para garantizar su éxito y adopción institucional.

La combinación de buenas prácticas de desarrollo, protección de datos y una estrategia de formación y soporte para los usuarios son los pilares para minimizar estos riesgos.

## Análisis de requisitos

El análisis de requisitos es una etapa clave en el desarrollo de cualquier sistema de software, ya que define qué debe hacer la aplicación y cómo debe comportarse. En el caso de TFGInfo, una aplicación destinada a gestionar Trabajos de Fin de Grado en el entorno universitario es fundamental identificar de forma clara las necesidades de cada tipo de usuario, así como los requisitos técnicos, de seguridad y de usabilidad.

TFGInfo debe permitir la gestión de estudiantes, profesores, departamentos, líneas de TFG, reservas y asignaciones, todo ello en un entorno multiusuario con distintos roles y niveles de acceso.

### Historias de usuario - Requisitos funcionales

Los requisitos funcionales describen las funcionalidades específicas que el sistema debe ofrecer. Estos se derivan directamente de las tareas que deben realizar los distintos actores del sistema.

**Gestión de usuarios**

El sistema debe permitir registrar, editar y eliminar usuarios (alumnos, profesores).

**Como** administrador **Quiero** poder registrar nuevos usuarios (alumnos, profesores, administradores) **Para** proporcionar acceso a la aplicación a quienes lo necesiten

Cada usuario debe autenticarse con credenciales y un rol específico

**Como** administrador **Quiero** poder asignar roles a los usuarios **Para** que solo tengan acceso a las funciones correspondientes a su perfil

Los roles determinarán el acceso a las distintas funciones del sistema

**Como** usuario registrado **Quiero** poder iniciar sesión con mis credenciales **Para** acceder a mis funcionalidades personalizadas de forma segura

**Gestión de Trabajos de Fin de Grado (TFG)**

El sistema debe permitir crear, editar, buscar y eliminar registros de TFG.

**Como** administrador  
**Quiero** poder crear, editar registros de TFG y asociar profesores y departamentos  
**Para** proponer trabajos a los estudiantes y profesores

La lista de TFGs debe ser accesible para los usuarios, permitiendo consultar la información y solicitar una línea de TFG para su desarrollo.

**Como** estudiante  
**Quiero** poder ver los TFG disponibles para mi titulación  
**Para** elegir el que más se ajuste a mis intereses

**Como** profesor  
**Quiero** poder ver los TFG disponibles a mi cargo  
**Para** aceptar o rechazar las solicitudes de los estudiantes

**Gestión de líneas temáticas y reservas**

Los estudiantes podrán enviar solicitudes de reserva para una línea de TFG.

**Como** estudiante

**Quiero** poder enviar una solicitud de reserva de una línea de TFG

**Para** expresar mi interés en trabajar con un profesor específico

El sistema debe permitir al tutor aceptar o rechazar una solicitud.

**Como** profesor

**Quiero** poder aceptar o rechazar reservas de estudiantes

**Para** seleccionar a quienes considero más adecuados para el trabajo

El sistema notificará por correo electrónico el resultado de la solicitud.

**Como** sistema

**Quiero** enviar una notificación por correo electrónico al estudiante

**Para** informarle del estado de su solicitud de reserva

**Gestión de departamentos, universidades y titulaciones**

La plataforma debe permitir administrar departamentos y carreras, asociando profesores y TFGs a cada uno.

**Como** administrador

**Quiero** poder crear y administrar departamentos y titulaciones

**Para** reflejar la estructura académica real dentro de la aplicación

Debe existir una estructura jerárquica donde las titulaciones pertenezcan a universidades y los profesores a departamentos.

**Como** profesor

**Quiero** estar asociado a un departamento

**Para** que mi perfil refleje correctamente mi afiliación institucional

**Gestión de grupos de trabajo**

El sistema permitirá la creación de grupos de alumnos que comparten un mismo TFG.

**Como** estudiante

**Quiero** poder formar un grupo de trabajo con otros compañeros

**Para** colaborar en un mismo TFG

Los grupos podrán tener asignaciones de profesores y TFG asociados.

**Como** profesor

**Quiero** poder asignar un grupo de estudiantes a un TFG

**Para** dirigirlos de forma conjunta en el desarrollo del proyecto

**Comunicación interna básica**

El sistema debe enviar correos electrónicos automáticos para notificar acciones como aceptación/rechazo de TFG, confirmaciones de asignación, etc.

**Como** sistema

**Quiero** enviar correos automáticos al aceptar o rechazar una solicitud

**Para** mantener informados a estudiantes y profesores sin intervención manual

### Historias de usuario – Requisitos no funcionales

**Seguridad**

Toda la información debe transmitirse de forma segura (HTTPS).

**Como** usuario de la aplicación

**Quiero** que todas las comunicaciones se realicen de forma segura

**Para** garantizar que mis datos personales no puedan ser interceptados

Los datos personales deben estar protegidos cumpliendo con el RGPD.

**Como** responsable del sistema

**Quiero** que los datos personales estén protegidos según el RGPD

**Para** cumplir con la legislación y proteger la privacidad de los usuarios

El sistema debe implementar autenticación y autorización basada en roles.

**Como** administrador del sistema

**Quiero** que se implementen roles y permisos de acceso

**Para** asegurar que cada usuario solo pueda ver y hacer lo que le corresponde

**Rendimiento**

Las búsquedas y cargas de datos deben ser rápidas, incluso con grandes volúmenes de información.

**Como** usuario

**Quiero** que las búsquedas y cargas de información sean rápidas

**Para** poder trabajar de forma fluida sin interrupciones

El sistema debe implementar filtros para optimizar el rendimiento.

**Como** usuario

**Quiero** que se permita hacer un filtrado en las búsquedas

**Para** realizar consultas más rápidas y directas

**Escalabilidad**

La arquitectura debe permitir añadir nuevas funcionalidades sin reestructurar el núcleo.

**Como** desarrollador del sistema

**Quiero** que la arquitectura esté diseñada de forma modular

**Para** poder añadir nuevas funcionalidades sin rehacer el sistema completo

**Usabilidad**

La interfaz debe ser intuitiva y accesible para usuarios con distintos niveles técnicos.

**Como** usuario sin conocimientos técnicos

**Quiero** que la interfaz sea clara e intuitiva

**Para** poder usar la aplicación sin necesidad de formación previa

Debe adaptarse a dispositivos móviles (diseño responsive).

**Como** estudiante o profesor

**Quiero** poder acceder desde mi móvil o tablet

**Para** gestionar tareas en cualquier momento y lugar

**Mantenibilidad**

El código debe estar organizado en módulos fácilmente actualizables.

**Como** desarrollador

**Quiero** que el código esté organizado en módulos bien definidos

**Para** poder mantenerlo, depurarlo o ampliarlo fácilmente

Se debe mantener una documentación técnica y funcional clara.

**Como** equipo técnico

**Quiero** contar con documentación técnica y funcional

**Para** asegurar la continuidad del proyecto en futuras versiones

**Compatibilidad**

El sistema debe funcionar en navegadores modernos (Chrome, Firefox, Edge).

**Como** usuario

**Quiero** que la aplicación funcione correctamente en los navegadores más comunes

**Para** poder acceder desde cualquier entorno sin problemas

## Casos de uso

Los casos de uso describen de forma estructurada **cómo interactúan los distintos actores del sistema TFGInfo con las funcionalidades principales de la aplicación**. Esta sección tiene como objetivo detallar los flujos de comportamiento esperados, mostrando cómo profesores, estudiantes y administradores utilizan el sistema para realizar tareas clave relacionadas con la gestión de Trabajos de Fin de Grado.

Cada caso de uso parte de una acción concreta iniciada por un actor (como "iniciar sesión", "enviar solicitud", "aceptar TFG" o "crear una entidad"), e incluye los pasos principales que realiza el sistema, así como las decisiones y resultados posibles.

Los casos de uso abordados incluyen acciones fundamentales como:

* **Inicio de sesión y autenticación de usuarios**
* **Consulta, filtrado y solicitud de TFGs por parte del estudiante**
* **Aceptación o rechazo de solicitudes de TFG por parte del profesor**
* **Envío de mensajes por parte del profesor a grupos de trabajo**
* **Creación de entidades institucionales (universidades, departamentos, carreras)** por parte del administrador
* **Acceso a canales de trabajo y comunicación grupal entre profesores y alumnos**

Estos diagramas de flujo ayudan a visualizar el comportamiento del sistema y sirven como guía para el desarrollo, las pruebas y la validación del software. Además, permiten identificar posibles mejoras o extensiones futuras de la plataforma.

Diagrama, Forma

El contenido generado por IA puede ser incorrecto.

**Caso de Uso:** Inicio de sesión

**Actor principal:** Usuario (estudiante, profesor o administrador)  
**Objetivo:** Acceder al sistema de forma segura mediante credenciales personales.  
**Resumen:**  
Este caso de uso describe el proceso mediante el cual un usuario registrado accede a la plataforma TFGInfo introduciendo su correo electrónico y contraseña. El sistema valida las credenciales contra la base de datos. Si son correctas, el usuario es redirigido a su panel correspondiente según su rol. En caso de error, el sistema muestra un mensaje de autenticación fallida y permite reintentar el acceso.

**Precondiciones:**

* El usuario debe estar registrado previamente en el sistema.
* Debe disponer de un correo electrónico y una contraseña válidos.

**Flujo principal:**

1. El usuario accede a la página de inicio de sesión.
2. Introduce su correo y contraseña.
3. El sistema envía los datos al backend para validación.
4. Si las credenciales son válidas, se genera un token de acceso y se redirige al panel correspondiente.
5. Si no son válidas, se muestra un mensaje de error.

**Postcondición:**  
El usuario queda autenticado en el sistema y puede acceder a las funcionalidades correspondientes a su perfil.

Diagrama

El contenido generado por IA puede ser incorrecto.

**Caso de Uso:** Solicitud de TFG

**Actor principal:** Estudiante  
**Objetivo:** Enviar una solicitud formal para reservar un Trabajo de Fin de Grado (TFG) de su interés.  
**Resumen:**  
Este caso de uso describe el proceso mediante el cual un estudiante, tras iniciar sesión en la plataforma TFGInfo, accede al listado de TFGs disponibles, aplica filtros de búsqueda, consulta los detalles de un TFG y realiza una solicitud para reservarlo. El sistema registra la solicitud y notifica automáticamente al profesor responsable. La solicitud queda pendiente de aprobación.

**Precondiciones:**

* El estudiante debe estar registrado y autenticado en el sistema.
* Debe existir al menos un TFG disponible asociado a su titulación.

**Flujo principal:**

1. El estudiante inicia sesión y accede al módulo de TFGs.
2. Filtra y consulta las líneas de TFG disponibles.
3. Selecciona un TFG que le interese.
4. Envía una solicitud de reserva desde el sistema.
5. El sistema guarda la solicitud y notifica al profesor responsable por correo electrónico.
6. Se muestra un mensaje de confirmación al estudiante.

**Postcondición:**  
La solicitud queda registrada como pendiente y el profesor podrá aceptarla o rechazarla posteriormente.

Diagrama

El contenido generado por IA puede ser incorrecto.

**Caso de Uso:** Gestión de solicitudes de TFG

**Actor principal:** Profesor  
**Objetivo:** Gestionar las solicitudes recibidas por parte de estudiantes para reservar un Trabajo de Fin de Grado (TFG).  
**Resumen:**  
Este caso de uso describe cómo un profesor, una vez autenticado en la plataforma, accede al módulo de gestión de solicitudes, revisa las peticiones realizadas por estudiantes para optar a uno de sus TFGs, y decide aceptarlas o rechazarlas. El sistema actualiza el estado de la solicitud y notifica automáticamente al estudiante por correo electrónico.

**Precondiciones:**

* El profesor debe estar registrado y autenticado.
* Debe haber al menos una solicitud pendiente de TFG en la que él sea tutor asignado.

**Flujo principal:**

1. El profesor inicia sesión y accede al módulo de solicitudes.
2. Revisa la lista de solicitudes pendientes asociadas a sus TFGs.
3. Consulta los detalles del estudiante y del TFG solicitado.
4. Decide aceptar o rechazar la solicitud.
5. El sistema actualiza el estado de la solicitud.
6. Se envía una notificación automática al estudiante con la resolución.

**Postcondición:**  
La solicitud queda registrada como aceptada o rechazada, y el estudiante es informado del resultado.

Diagrama

El contenido generado por IA puede ser incorrecto.

**Caso de Uso:** Administración de entidades

**Actor principal:** Administrador  
**Objetivo:** Registrar nuevas entidades académicas dentro del sistema, como universidades, departamentos o titulaciones.  
**Resumen:**  
Este caso de uso describe cómo un administrador puede acceder al módulo de gestión institucional y crear nuevas entidades que forman parte de la estructura académica. Estas entidades son necesarias para organizar correctamente los TFGs, los usuarios y sus relaciones jerárquicas. El sistema valida los datos introducidos y guarda la información en la base de datos.

**Precondiciones:**

* El administrador debe estar autenticado.
* Debe tener permisos de gestión estructural.

**Flujo principal:**

1. El administrador inicia sesión y accede al módulo de entidades institucionales.
2. Selecciona el tipo de entidad a crear: universidad, departamento o carrera.
3. Rellena el formulario correspondiente con los datos necesarios (nombre, siglas, jerarquía, etc.).
4. El sistema valida los campos introducidos.
5. Si los datos son correctos, se registra la nueva entidad en la base de datos.
6. Se muestra una confirmación del alta exitosa.

**Postcondición:**  
La nueva entidad queda registrada y disponible para su uso en la configuración de usuarios, TFGs y relaciones institucionales.

Diagrama

El contenido generado por IA puede ser incorrecto.

**Caso de Uso:** Estudiante forma parte de un canal

**Actor principal:** Estudiante  
**Objetivo:** Formar parte de un canal.  
**Resumen:**  
Este caso de uso describe cómo un estudiante puede visualizar, unirse o formar parte de un canal. El canal permite que varios alumnos colaboren en el mismo proyecto bajo la dirección de uno o varios tutores. La aplicación gestiona las acciones disponibles desde el punto de vista del alumno.

**Precondiciones:**

* El estudiante debe estar registrado y autenticado en el sistema.
* Debe haber sido invitado o asignado a un grupo, o tener habilitada la opción de unirse a uno.

**Flujo principal:**

1. El estudiante accede a su panel personal.
2. Entra en la sección de grupos de trabajo.
3. Consulta los grupos disponibles o ya asignados.
4. Puede unirse a un canal (si está público).
5. Una vez dentro del canal, accede a la información compartida: integrantes, TFG asignado, mensajes del profesor, fechas clave, etc.

**Postcondición:**  
El estudiante queda formalmente vinculado a un canal y puede recibir notificaciones de los profesores que se encuentren en él.

Diagrama

El contenido generado por IA puede ser incorrecto.

**Caso de Uso:** Gestión de un Canal de Trabajo por parte del Profesor

**Actor principal:** Profesor  
**Objetivo:** Administrar un canal de trabajo en el que se agrupan estudiantes asignados a un mismo TFG.  
**Resumen:**  
Este caso de uso describe cómo un profesor puede gestionar un canal de trabajo académico. Si el profesor ya es miembro del canal (o accede a uno público), podrá realizar varias acciones sobre los estudiantes o sobre el canal en sí, incluyendo enviar notificaciones, consultar información, invitar nuevos integrantes, expulsar miembros o incluso eliminar el canal. También puede decidir abandonarlo, si las condiciones lo permiten.

**Precondiciones:**

* El profesor debe estar autenticado.
* Debe ser miembro del canal, o el canal debe ser público.

**Flujo principal:**

1. El profesor inicia sesión y accede al módulo de grupos de trabajo.
2. Selecciona un canal de la lista.
3. Si es miembro, o si el canal es público, entra al canal.
4. Una vez dentro, puede realizar las siguientes acciones:
   * **Consultar información del grupo** (integrantes, mensajes, TFG asignado).
   * **Enviar notificaciones** o mensajes al grupo.
   * **Invitar nuevos alumnos** al canal.
   * **Expulsar miembros** del grupo.
   * **Borrar el canal** (si es propietario o único responsable).
   * **Abandonar el canal**, siempre que no sea el único miembro con rol de profesor.
5. Si intenta abandonar el canal y no hay otros profesores o es el último miembro, se bloquea la salida por motivos de integridad.

**Postcondición:**  
El profesor ha gestionado correctamente su canal y sus miembros, manteniendo el flujo de comunicación y supervisión del grupo de trabajo. Si decide abandonarlo, se asegura que no se comprometa la gestión académica del grupo.

## Modelo de dominio

Diagrama

El contenido generado por IA puede ser incorrecto.

## Diagramas de secuencia

**Registro de un usuario**

El siguiente diagrama de secuencia representa el flujo completo del proceso de registro de una nueva persona en TFGInfo. Este proceso puede ser iniciado por un administrador que crea el usuario manualmente, o por un sistema de preinscripción.

Diagrama

El contenido generado por IA puede ser incorrecto.

**Comenzar TFG**

Este diagrama de secuencia describe el proceso mediante el cual un estudiante solicita comenzar un trabajo de fin de grado y el profesor decide si aceptarlo o rechazarlo.

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**Envío de correo por canal**

Este diagrama muestra como un profesor perteneciente a un canal puede notificar a todos los estudiantes que estén en este.

**Diagrama

El contenido generado por IA puede ser incorrecto.**

Diseño del sistema

## Interfaz de la aplicación

En el desarrollo de cualquier sistema software, la usabilidad y la interfaz de usuario no deben considerarse elementos secundarios o accesorios, sino componentes fundamentales del producto final. De hecho, el éxito o fracaso de una aplicación no suele depender únicamente de lo potente que sea su backend o de lo bien estructurada que esté su base de datos, sino de la experiencia que ofrece a sus usuarios cuando la utilizan. Las características que se han buscado a la hora de diseñar los primeros bocetos de la interfaz han sido los siguientes:

**Diseño limpio y minimalista**

* La interfaz usa un diseño minimalista, con un fondo blanco y elementos bien espaciados, que favorecen la lectura y reducen la sobrecarga cognitiva.
* Las líneas y bordes suaves transmiten una sensación de modernidad y profesionalismo, evitando distracciones.

**Estructura clara y jerarquía visual**

* La pantalla está dividida en un menú superior de navegación y un cuerpo principal con la vista actual, logrando una jerarquía muy clara. Manteniendo el menú superior constante durante toda la navegación de la aplicación
* En las vistas de listados cada entidad se presenta en una tarjeta horizontal con campos clave (nombre, departamento, profesores, plazas), lo que permite comparar rápidamente las opciones disponibles.

**Funcionalidad intuitiva**

* Los listados incluyen un campo de búsqueda bien destacado y un icono de filtros avanzados, que son acciones esperadas por los usuarios para mejorar la experiencia en la exploración de los elementos de la aplicación.
* Los iconos son reconocibles y estándar (persona para el perfil, embudo para filtro, avión de papel para enviar/solicitar), reforzando la comprensión sin necesidad de textos adicionales.

**Coherencia y consistencia**

* Todos los elementos mantienen un estilo coherente: misma tipografía, colores uniformes y distribución homogénea, lo que genera una experiencia de uso fluida.
* Las columnas están alineadas correctamente, facilitando la lectura horizontal de los datos.

**Navegación sencilla**

* En la parte superior, la barra de navegación incluye accesos directos a secciones clave como Mis reservas, TFG, Grupos de estudio y Profesores, así como un icono de usuario para el perfil y otro de menú desplegable de idioma.
* Esto facilita que cualquier usuario pueda desplazarse de manera rápida a otras funcionalidades sin perderse.

**Preparada para la adaptabilidad**

* Se ha adaptado todas las vistas para cualquier tipo de dispositivo, permitiendo que los usuarios puedan acceder a la aplicación desde cualquier dispositivo y visualizarla con claridad independientemente del tamaño de la pantalla.

### Bocetos de la aplicación

Los bocetos de la aplicación han supuesto un elemento clave a la hora del desarrollo de esta ya que se parte de una base concreta. Al hacer bocetos sobre la interfaz se han podido detectar posibles errores de usabilidad en momentos previos al desarrollo de la aplicación y de esta forma no se ha gastado tiempo posteriormente en corrección de errores. En el resultado final se puede visualizar la inspiración en los bocetos para el desarrollo de la interfaz, pero en el proceso se han hecho algunos cambios. Estas son algunas imágenes comparando la interfaz final con los bocetos en las que se han basado.

**Resultado Final Inicio de sesión Boceto Inicio de sesión**

Imagen de la pantalla de un celular con letras

El contenido generado por IA puede ser incorrecto. Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

En estas vistas la diferencia es muy sutil, ya que tras el desarrollo de la aplicación y el analisis de requisitos se ha eliminado la posibilidad de que un usuario pueda registrarse por si mismo en la aplicación. Además se ha optado por un estilo más minimalista para el formulario de inicio de sesión.

**Boceto de listado de TFGs**

**Interfaz de usuario gráfica, Tabla

El contenido generado por IA puede ser incorrecto.**

**Resultado final listado de TFGs**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

En las vistas de listado se sigue el formato para la búsqueda de entidades, se han agregado el botón de buscar y de limpiar filtros. Sobre el menú superior se ha optado por darle un color diferente al color de fondo habitual en el resto de las vistas, para buscar que destaque por encima ya que se trata de un componente que se va a mostrar constante en la aplicación.

Las tarjetas de las entidades mostradas están contenidas en una tabla con un color algo más oscuro para diferenciarla del fondo de la aplicación.

**Boceto perfil de un alumno**

**Interfaz de usuario gráfica, Sitio web

El contenido generado por IA puede ser incorrecto.**

**Resultado final perfil de un alumno**

**Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.**

Se han cambiado el formato de los formularios pasando a tener un estilo más minimalista.

### Usabilidad

La usabilidad es fundamental en cualquier producto digital, especialmente en aplicaciones web, porque se trata de qué tan fácil, eficiente y agradable es para los usuarios interactuar con esa aplicación. Algunas ventajas de tener en cuenta la usabilidad en este proyecto son las siguientes:

**Navegación consistente y predecible**  
La interfaz mantiene siempre los mismos elementos de menú y cabecera, lo que reduce la curva de aprendizaje y permite que el usuario sepa siempre dónde está y cómo volver a las secciones principales. Esto disminuye la frustración y acelera las tareas.

**Acciones claras y confirmadas**  
Antes de realizar acciones importantes (como eliminar o importar datos) aparecen mensajes de confirmación. Esto genera seguridad y evita errores costosos, aumentando la confianza del usuario en la herramienta.

**Búsqueda y filtrado inmediato**  
La posibilidad de buscar y filtrar directamente sin navegar por varias páginas reduce el tiempo para encontrar información, algo clave en entornos donde se gestiona mucho contenido.

**Identidad visual y estructura estable**  
Elementos como el logotipo y la cabecera permanecen visibles, ayudando a mantener el contexto y reforzando la orientación dentro de la aplicación.

### Usabilidad

La accesibilidad en aplicaciones web es esencial para garantizar que cualquier persona, independientemente de sus capacidades físicas, sensoriales o cognitivas, pueda interactuar con la plataforma sin barreras. Tener en cuenta la accesibilidad en este proyecto aporta ventajas como las siguientes:

**Contraste y legibilidad adaptables**

La posibilidad de alternar entre un modo claro y uno oscuro permite a los usuarios ajustar el contraste según sus necesidades visuales. Esto es especialmente útil para personas con baja visión, daltonismo o sensibilidad a la luz, facilitando la lectura y reduciendo la fatiga ocular.

**Cambio de idioma en tiempo real**

La interfaz ofrece la opción de cambiar el idioma sin recargar o interrumpir la navegación. Esto garantiza que personas con distintos niveles de dominio del idioma puedan comprender y utilizar la aplicación con fluidez, mejorando su experiencia global.

**Contenido clave siempre accesible**

Elementos como la cabecera fija y la navegación persistente evitan que el usuario tenga que desplazarse continuamente para acceder a funciones esenciales. Esto beneficia a personas con movilidad reducida o que usan dispositivos de entrada alternativos, como teclados adaptados.

**Diseño adaptable a diferentes dispositivos**

La interfaz se reorganiza de manera responsive, garantizando que las personas que usan móviles o tablets —incluyendo aquellas que dependen de un tamaño de letra mayor— puedan acceder a toda la información sin necesidad de hacer zoom o desplazamientos excesivos.

### Responsive

El diseño responsive es clave para que una aplicación web se adapte de forma óptima a diferentes dispositivos y tamaños de pantalla, garantizando una experiencia de uso fluida y coherente en ordenadores, tablets y móviles. En este proyecto, su aplicación aporta ventajas como las siguientes:

**Adaptación automática del contenido**

La interfaz reorganiza y ajusta sus elementos según el ancho y alto de la pantalla, evitando que el usuario tenga que desplazarse lateralmente o hacer zoom para visualizar la información.

**Priorización de la información esencial**

En pantallas pequeñas, se muestran de forma destacada las funciones y contenidos más relevantes, lo que facilita la interacción y evita la sobrecarga visual.

**Interacción cómoda en cualquier dispositivo**

Los elementos táctiles, como botones y menús, se dimensionan y separan adecuadamente en dispositivos móviles, reduciendo errores al pulsar y mejorando la experiencia de uso en pantallas táctiles.

**Consistencia visual en distintos entornos**

El diseño mantiene la misma identidad y estructura general independientemente del dispositivo, lo que permite al usuario reconocer fácilmente la aplicación y usarla con confianza desde cualquier lugar.

Implementación y pruebas

## Estructura del proyecto

### Servidor - Backend

El backend de este proyecto está organizado siguiendo un patrón modular y escalable, pensado para separar responsabilidades y facilitar tanto el mantenimiento como la futura ampliación de funcionalidades. La base del backend se encuentra en la carpeta webapi, que centraliza toda la lógica de servidor, la configuración y los controladores que gestionan las distintas áreas de la aplicación.

**Organización de carpetas**

Dentro de webapi la estructura se divide en varias secciones clave:

* **Common/**  
  Esta carpeta contiene utilidades y clases base reutilizables en distintos módulos. Por ejemplo, BaseController.cs y BaseManager.cs proporcionan una estructura común para los controladores y gestores de lógica, asegurando que todos los módulos sigan un mismo patrón. Aquí también se encuentran clases para manejar excepciones específicas (NotFoundException.cs, UnprocesableException.cs), lo que permite capturar y devolver errores de forma controlada y uniforme.
* **Data/**  
  Contiene ApplicationDbContext.cs, que es el núcleo del acceso a datos a través de Entity Framework. Aquí se definen los DbSet que representan las tablas y se gestionan las conexiones y el mapeo entre objetos y base de datos.
* **Modules/**  
  Esta es la sección más extensa y donde realmente se observa el diseño modular del backend. Cada entidad o área de negocio (por ejemplo, Career, Department, Professor, Student, TFG, etc.) se organiza en su propia carpeta con tres elementos principales:
  + **Controller**: punto de entrada de las peticiones HTTP para esa entidad. Se encarga de recibir las solicitudes, validarlas y delegar la lógica al manager correspondiente.
  + **Manager**: capa intermedia donde reside la lógica de negocio. Aquí se gestionan validaciones más complejas, reglas específicas y procesamiento de la información.
  + **Model y Object**: clases que definen la estructura de datos. El *Model* suele representar la entidad tal y como se gestiona internamente, mientras que el *Object* puede actuar como un DTO (Data Transfer Object) para controlar qué datos se envían o reciben.

Este esquema de **Controller → Manager → Model** se repite en todos los módulos, consiguiendo que el código sea más limpio, reutilizable y fácil de localizar.

**Controladores y Managers**

El uso de controladores y managers bien diferenciados es una de las fortalezas de este proyecto.

* Los **controladores** son ligeros y están orientados únicamente a gestionar la comunicación con el cliente. No contienen lógica compleja, lo que evita que se conviertan en piezas monolíticas difíciles de mantener.
* Los **managers** concentran la lógica de negocio. Este diseño sigue el principio de responsabilidad única (SRP), porque cada clase tiene un cometido claro: el controlador “habla” con el exterior y el manager “piensa” cómo resolver la petición.

Esto no solo facilita la lectura del código, sino que también permite que, si en el futuro se quiere cambiar la API o añadir un nuevo punto de entrada, gran parte de la lógica ya esté encapsulada y no sea necesario reescribirla.

**Uso de Entity Framework**

Entity Framework (EF) actúa como el ORM (Object Relational Mapper) del proyecto. Gracias a él, no es necesario escribir SQL manual para las operaciones más comunes, ya que EF traduce automáticamente las operaciones con objetos C# en consultas SQL a la base de datos.

En ApplicationDbContext.cs se definen las colecciones DbSet que representan las tablas de la base de datos (Student, Professor, Department, TFG, etc.). Este contexto también se encarga de aplicar configuraciones, relaciones y restricciones, siguiendo lo definido en los modelos.

El uso de EF aporta varias ventajas:

* **Abstracción del acceso a datos**: se trabaja directamente con objetos y colecciones en lugar de sentencias SQL.
* **Control de migraciones**: permite actualizar el esquema de la base de datos de forma controlada a medida que evolucionan los modelos.
* **Integridad referencial automática**: respeta y gestiona las relaciones definidas entre entidades.

Gracias a esta configuración, el backend consigue ser a la vez **potente** y **mantenible**, con una estructura lógica clara y preparada para crecer. El patrón modular, la separación entre controladores y managers, y el uso de Entity Framework convierten el sistema en una base sólida sobre la que construir nuevas funcionalidades sin comprometer la estabilidad.

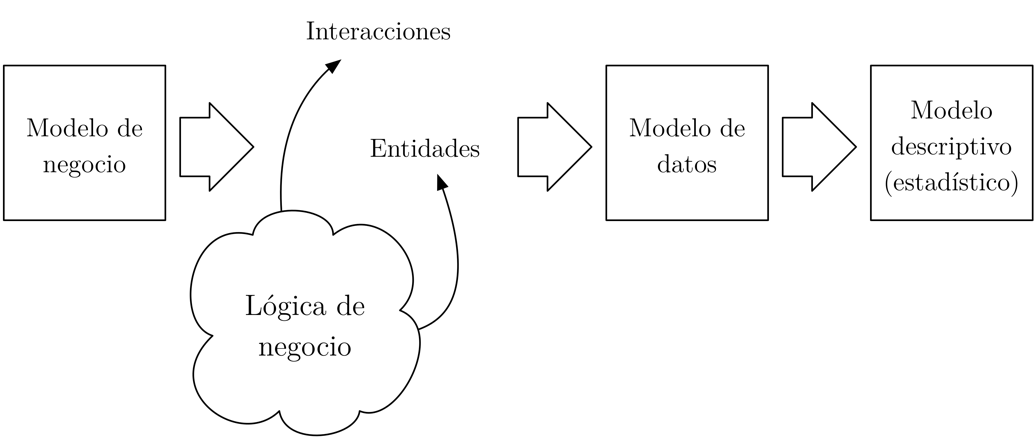


Fig. 1. Título de la Primera Figura

Referencias

1. Referencia primera
2. Manual de Instalación
   1. Requerimientos
3. Otros Recursos



E.T.S de Ingeniería Informática

Bulevar Louis Pasteur, 35

Campus de Teatinos

29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA