Índice

1. [Introducción](#_Introducción)
2. Presentación del proyecto
3. Objetivos del proyecto
4. Justificación del proyecto
5. [Análisis de requerimientos](#_Análisis_de_requerimientos)
6. Identificación de necesidades y requerimientos
7. Identificación de publico
8. Estudio de mercado y competencia
9. [Diseño y planificación](#_Diseño_y_planificación)
10. Definición de la arquitectura del proyecto
11. Diseño de la interfaz de usuario
12. Planificación de las tareas y los recursos necesarios
13. [Implementación y pruebas](#_IMPLEMENTACIÓN_Y_PRUEBAS)
14. Desarrollo de las funcionalidades del proyecto
15. Pruebas unitarias y de integración
16. Corrección de errores y optimización del rendimiento
17. [Documentación](#_DOCUMENTACIÓN)
18. Documentación técnica
19. Documentación de usuario
20. Manual de instalación y configuración
21. [Mantenimiento y evolución](#_Mantenimiento_y_evolución)
22. Plan de mantenimiento y soporte
23. Identificación de posibles mejoras y evolución del proyecto
24. Actualizaciones y mejoras futuras
25. [Conclusiones](#_Conclusiones)
26. Evaluación del proyecto
27. Cumplimiento de objetivos y requisitos
28. Lecciones aprendidas y recomendaciones para futuros proyectos
29. [Bibliografía y referencias](#_Bibliografía_y_referencias)
30. Fuentes utilizadas en el proyecto
31. Referencias y enlaces de interés

# Introducción

1. **PRESENTACIÓN DEL PROYECTO**

Este trabajo de fin de grado se centra en el desarrollo de una aplicación web llamada **TaskSync**, creada con el objetivo de facilitar a los usuarios una gestión más eficiente y flexible de sus tareas diarias. La aplicación se basa en la técnica Pomodoro, pero con la particularidad de que cada usuario puede adaptar y configurar los intervalos de trabajo y descanso según sus propias necesidades y preferencias, ofreciendo así una experiencia más personalizada.

TaskSync permite la creación de tareas principales y la posibilidad de desglosarlas en subtareas, lo que facilita una organización más detallada y estructurada del trabajo o estudio. Además, el usuario puede asignar tiempos específicos para cada periodo de concentración y descanso, así como visualizar en tiempo real el progreso de sus sesiones, lo que ayuda a mantener la motivación y el control sobre sus actividades.

Además de su funcionalidad principal, la aplicación busca proporcionar una interfaz de usuario intuitiva y atractiva que invite a la interacción y sea fácil de usar, incluso para quienes no están familiarizados con este tipo de herramientas.

El diseño se ha enfocado en crear un espacio que no solo ayude a planificar, sino que también fomente la concentración y reduzca la sensación de sobrecarga que a menudo acompaña a la acumulación de tareas, buscando en definitiva un apoyo real para mejorar la productividad y el bienestar en el día a día.

1. **OBJETIVOS DEL PROYECTO**

El principal objetivo de este proyecto es crear una aplicación web práctica y eficiente que facilite a los usuarios la gestión de sus tareas personales y académicas, adaptándose a sus necesidades y estilos de organización. Se busca que la aplicación sea intuitiva y permita un manejo sencillo de las tareas, incluyendo la posibilidad de crear subtareas para estructurar mejor el trabajo.

Además, se pretende garantizar la seguridad y privacidad de los datos mediante un sistema de autenticación robusto que asegure que solo los usuarios autorizados puedan acceder a su información, protegiendo así la confidencialidad y la integridad de los datos.

Desde el punto de vista tecnológico, se aspira a desarrollar una solución escalable, capaz de manejar un crecimiento en la base de usuarios o en las funcionalidades sin perder rendimiento ni estabilidad. Esto se logra con una arquitectura bien diseñada, utilizando tecnologías modernas tanto en el backend como en el frontend.

Por último, se contempla que la aplicación pueda evolucionar con el tiempo, integrando en el futuro nuevas funcionalidades que mejoren la experiencia del usuario, como opciones para trabajo colaborativo, análisis de productividad personalizada y un modo offline para garantizar el acceso y uso en cualquier situación.

1. **JUSTIFICACIÓN DEL PROYECTO**

En la actualidad, la gestión eficiente del tiempo y la organización de tareas se han vuelto aspectos esenciales tanto en la vida personal como en el ámbito profesional. La cantidad de actividades y responsabilidades que enfrentamos a diario hace necesario contar con herramientas que no solo ayuden a planificar, sino que también se adapten a las particularidades y estilos de trabajo de cada persona.

Aunque en el mercado existen muchas aplicaciones destinadas a la gestión de tareas, muchas de ellas resultan demasiado rígidas o genéricas, sin ofrecer la flexibilidad necesaria para personalizar tiempos, prioridades o modos de trabajo específicos, como el método Pomodoro. Esta falta de adaptabilidad puede limitar su utilidad para usuarios que buscan optimizar su productividad de forma más ajustada a sus propias rutinas.

El proyecto **TaskSync** responde a esta necesidad, ofreciendo una solución más flexible y completa que centraliza la administración de tareas, permitiendo la creación de subtareas, la configuración personalizada de intervalos de trabajo y descanso, y la visualización del progreso en tiempo real.

Por otro lado, este proyecto representa también una oportunidad importante para aplicar y consolidar los conocimientos adquiridos en el desarrollo web moderno, empleando tecnologías actuales como Spring Boot y JavaScript. Esto aporta un valor práctico y real a la formación, enfocándose en crear una herramienta que no solo sea funcional, sino que también mejore la experiencia del usuario y contribuya a su productividad de manera tangible.

# Análisis de requerimientos

**A. IDENTIFICACIÓN DE NECESIDADES Y REQUIRIMIENTOS**

La necesidad principal que impulsa este proyecto es facilitar a los usuarios una gestión eficiente, flexible y personalizada de sus tareas diarias, que les permita organizarse mejor y aumentar su productividad. Muchas personas encuentran dificultades a la hora de planificar su tiempo y priorizar actividades, por lo que resulta imprescindible contar con una herramienta que se adapte a distintos estilos de trabajo y que simplifique el seguimiento de objetivos.

A partir de esta necesidad, se han identificado los siguientes requerimientos funcionales para la aplicación:

* Registro y autenticación segura de usuarios mediante tokens JWT, garantizando que cada persona tenga acceso exclusivo a sus datos.
* Creación, edición, eliminación y visualización de tareas, permitiendo además desglosarlas en subtareas para un control más detallado.
* Configuración personalizada de tiempos para la técnica Pomodoro, incluyendo intervalos de trabajo y descanso ajustables.
* Visualización en tiempo real del progreso de las sesiones Pomodoro y gestión automática del encadenamiento de tareas.
* Organización de tareas mediante categorías y etiquetas personalizables para facilitar su clasificación.
* Notificaciones y recordatorios que ayuden al usuario a mantenerse enfocado y cumplir con sus plazos.
* Interfaz intuitiva, clara y responsiva, adaptable a diferentes dispositivos (ordenadores, tablets, móviles).

En cuanto a los requerimientos no funcionales, se destacan:

* Usabilidad: diseño sencillo y accesible que facilite la interacción y reduzca la curva de aprendizaje.
* Seguridad: protección de los datos personales y control de acceso a través de un sistema robusto de autenticación.
* Escalabilidad: arquitectura preparada para soportar crecimiento en funcionalidades y número de usuarios sin pérdida de rendimiento.
* Compatibilidad: correcto funcionamiento en los principales navegadores web actuales.
* Mantenibilidad: código estructurado y modular para facilitar futuras ampliaciones y mantenimiento.

Estos requerimientos buscan crear una aplicación que no solo cumpla con su función principal, sino que además ofrezca una experiencia agradable y confiable, capaz de evolucionar y adaptarse a las necesidades de sus usuarios.

**B. IDENTIFICACIÓN DEL PÚBLICO**

El público objetivo de esta aplicación abarca un perfil amplio pero concreto de usuarios que buscan mejorar la organización y gestión de sus tareas diarias, tanto en el ámbito académico, profesional como personal.

Principalmente, la aplicación está dirigida a:

* **Estudiantes**, que necesitan organizar sus tareas, proyectos, fechas de entrega y horarios de estudio, optimizando su tiempo y evitando el estrés por acumulación de responsabilidades.
* **Profesionales**, que desean gestionar eficazmente sus actividades laborales y personales, manteniendo un equilibrio entre ambas áreas mediante una planificación clara y estructurada.
* **Usuarios generales**, que buscan una herramienta flexible y personalizable que les permita organizar distintos aspectos de su vida diaria, desde listas de compras hasta metas personales, adaptándose a sus preferencias y rutinas.

El rango de edad estimado de los usuarios va desde los 16 hasta los 50 años, un grupo que generalmente cuenta con conocimientos básicos o intermedios en el uso de tecnologías digitales y aplicaciones web. Se espera que estos usuarios valoren especialmente una interfaz intuitiva y agradable visualmente.

Esta diversidad en el público objetivo ha sido considerada para que la aplicación ofrezca funcionalidades y una experiencia de uso que resulten accesibles y útiles para distintos perfiles y necesidades.

**C. ESTUDIO DE MERCADO Y COMPETENCIA**

En el mercado actual existen diversas aplicaciones dedicadas a la gestión de tareas, cada una con características y enfoques particulares que atienden diferentes necesidades. Algunas de las herramientas más populares incluyen:

* **Todoist**: Esta aplicación destaca por su interfaz limpia y sencilla, que facilita la gestión de tareas diarias. Además, ofrece funcionalidades de colaboración y sincronización entre dispositivos. Sin embargo, su versión gratuita presenta limitaciones que pueden restringir la experiencia completa del usuario.
* **Trello**: Más orientada a la gestión de proyectos colaborativos, Trello utiliza un sistema de tableros y tarjetas que permite visualizar el progreso de equipos de trabajo. Aunque es potente para proyectos complejos, puede resultar menos intuitiva para usuarios que buscan una gestión individual y rápida de tareas.
* **Microsoft To Do**: Integrada dentro del ecosistema Microsoft, esta herramienta ofrece una buena sincronización con otros productos de la compañía. No obstante, es menos personalizable y carece de algunas funcionalidades avanzadas que usuarios más exigentes podrían necesitar.
* **Google Tasks**: Con un diseño muy básico, Google Tasks es útil para listas simples pero no cuenta con opciones para categorías, etiquetas o recordatorios avanzados, lo que limita su capacidad para una organización más profunda.

Frente a estas opciones, la aplicación **TaskSync** presenta varias ventajas competitivas que la hacen destacar:

* Una **interfaz completamente personalizable**, que permite al usuario adaptar la visualización y organización de sus tareas según sus preferencias, facilitando así una experiencia única y más efectiva.
* La **posibilidad futura de incluir un modo colaborativo**, que permitirá a grupos de trabajo o equipos compartir proyectos y tareas de manera eficiente, ampliando el alcance de la aplicación.
* Un claro **enfoque en la experiencia del usuario y la flexibilidad visual**, buscando que la aplicación sea no solo funcional sino también agradable y motivadora para su uso diario.

# Diseño y planificación

**A. DEFINICIÓN DE LA ARQUITECTURA DEL PROYECTO**

El proyecto **TaskSync** se ha desarrollado siguiendo una arquitectura moderna basada en contenedores Docker. Esta elección no solo facilita el despliegue y la portabilidad del sistema, sino que también mejora la organización del entorno de desarrollo al separar claramente cada uno de los servicios que componen la aplicación.

El sistema completo se encuentra definido mediante un archivo docker-compose.yml, que orquesta todos los servicios necesarios para que la aplicación funcione correctamente. En total, se utilizan cuatro servicios:

1. **Frontend**:
   * Se utiliza una imagen ligera de **Nginx (alpine)** para servir los archivos estáticos del frontend, que se montan desde una carpeta local (./frontend) al contenedor.
   * Este servicio está expuesto en el puerto **4200**, lo que permite acceder a la aplicación desde el navegador de forma rápida.
2. **Backend**:
   * El backend está construido con **Spring Boot** y se lanza desde una carpeta local (./spring-backend) mediante una imagen personalizada.
   * Expone el puerto **8081** y se conecta directamente con el contenedor de **MySQL** usando las variables de entorno definidas (SPRING\_DATASOURCE\_URL, USERNAME, PASSWORD).
   * Además, se monta un volumen local (./uploads) que permite guardar y acceder a los archivos subidos por los usuarios, como por ejemplo el currículum.
   * Incluye una política de reinicio automática en caso de fallo (restart: on-failure) y espera a que MySQL esté completamente operativo antes de arrancar (depends\_on con service\_healthy).
3. **MySQL**:
   * Utiliza la imagen oficial de **MySQL 8.0** para la base de datos.
   * Se configuran el usuario, la contraseña y la base de datos (midb) a través de variables de entorno.
   * Expone el puerto **3306** y guarda los datos en un volumen persistente (mysql-data) para que no se pierdan si el contenedor se reinicia.
   * También cuenta con un healthcheck que garantiza que la base de datos esté lista antes de iniciar los servicios dependientes.
4. **phpMyAdmin**:
   * Ofrece una interfaz web para gestionar visualmente la base de datos, muy útil durante el desarrollo y las pruebas.
   * Está enlazado directamente al contenedor de MySQL y se expone en el puerto **8080** del host.

Diagrama

El contenido generado por IA puede ser incorrecto.

Todos los servicios están conectados a una red Docker personalizada llamada **tfg-network**, que facilita la comunicación entre ellos sin necesidad de exponer innecesariamente puertos o datos sensibles.

Esta arquitectura modular y bien organizada permite que la aplicación sea **escalable, mantenible y fácilmente desplegable**, tanto en entornos de desarrollo como en producción. Además, el uso de **volúmenes persistentes** asegura que los datos de los usuarios no se pierdan, y la configuración con Docker Compose simplifica muchísimo todo el proceso de ejecución del proyecto.

1. **DISEÑO DE LA INTERFAZ DE USUARIO**

Uno de los aspectos en los que más me he centrado a la hora de desarrollar TaskSync ha sido el diseño de la interfaz de usuario. Quería que fuera algo intuitivo, visualmente agradable y, sobre todo, funcional. Al tratarse de una aplicación pensada para organizar tareas y mejorar la productividad, la experiencia de usuario debía ser lo más clara posible, sin distracciones innecesarias y adaptada a todo tipo de usuarios, desde estudiantes hasta profesionales.

La interfaz está desarrollada en HTML, CSS y JavaScript, utilizando una estructura sencilla pero eficaz. He trabajado especialmente en lograr un diseño **responsivo**, es decir, que se adapte tanto a pantallas grandes (como ordenadores) como a dispositivos móviles, para que el usuario pueda acceder y gestionar sus tareas desde cualquier sitio.

**Estructura general**

La aplicación cuenta con varias vistas principales, entre ellas:

* **Vista de tareas**: es la pantalla central, donde se pueden crear, editar, completar y eliminar tareas. Las tareas pueden tener subtareas, lo que permite una organización jerárquica.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

* **Vista Inicio de sesión:** Es la pantalla que primero ve el usuario, y nos da la posibilidad de registrarnos y de iniciar sesión.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

* **Perfil de usuario**: permite a cada persona subir su currículum , añadir un texto "Sobre mí" y personalizar algunos aspectos de su experiencia.

Interfaz de usuario gráfica, Patrón de fondo

El contenido generado por IA puede ser incorrecto.

En cuanto a la estética, he utilizado una paleta de colores suaves, con etiquetas y categorías destacadas mediante colores personalizados. Esto ayuda a visualizar rápidamente qué tipo de tareas hay, cuáles están completas y cuáles están pendientes.

También he añadido pequeños detalles para mejorar la interacción, como botones con animaciones suaves, mensajes de confirmación y estilos que indican el estado de cada tarea. Aunque no soy diseñadora profesional, he intentado seguir principios básicos de usabilidad, como la coherencia visual, la jerarquía de información y el feedback inmediato al usuario.

**Personalización y accesibilidad**

Otro punto clave ha sido permitir que cada persona adapte la herramienta a su forma de trabajar. Por eso se pueden usar **etiquetas**, **colores**, **categorías** y elegir diferentes vistas para organizarse mejor.

Aunque no es un proyecto centrado en accesibilidad, he procurado usar fuentes legibles, colores con suficiente contraste y botones bien distribuidos, pensando en que cualquier usuario pueda utilizarlo sin complicaciones.

En resumen, el diseño de la interfaz de TaskSync no busca ser espectacular, sino práctico, claro y agradable de usar. La idea es que el usuario se sienta cómodo organizando su día a día y que no pierda tiempo aprendiendo cómo funciona la herramienta.

1. **PLANIFICACIÓN DE LAS TAREAS Y LOS RECURSOS NECESARIOS**

Para poder desarrollar TaskSync de forma ordenada y eficiente, me organicé dividiendo el proyecto en varias fases. Al tratarse de una aplicación completa, con backend, frontend y lógica de negocio, era importante tener bien claro qué debía hacer en cada momento y qué necesitaba para ello.

**Fases del desarrollo**

1. **Análisis y definición de requisitos**  
   Antes de empezar a programar, recopilé todas las funcionalidades que quería incluir en la aplicación, como la creación de tareas, su anidamiento, el temporizador Pomodoro, la posibilidad de subir un currículum, etc. También pensé en cómo debía ser la interfaz, cómo se iban a comunicar el cliente y el servidor, y qué datos iba a necesitar.
2. **Diseño de la base de datos y la arquitectura**  
   Una vez definidos los requisitos, pasé a diseñar las entidades principales , Tarea, Categoría, Usuario y Pomodoro, sus relaciones y cómo se almacenarían. Usé JPA para facilitar el acceso a los datos desde el backend. También planifiqué la arquitectura general: Spring Boot para el backend y JavaScript con HTML/CSS para el frontend.
3. **Implementación del backend (API REST con Spring Boot)**  
   En esta fase me centré en desarrollar los controladores, la configuración y los repositorios para gestionar las tareas, usuarios y otras entidades. Implementé la lógica del método Pomodoro, incluyendo el encadenamiento automático de tareas y la transición entre ellas.
4. **Desarrollo del frontend (HTML, CSS y JavaScript)**  
   Con el backend funcionando, empecé a crear la interfaz de usuario. Hice varias páginas: una principal con todas las tareas, otra dedicada al modo Pomodoro, y otra para el perfil. En JavaScript añadí la lógica para consumir la API, mostrar las tareas de forma jerárquica, gestionar el temporizador y permitir acciones como completar, editar o eliminar.
5. **Pruebas, correcciones y mejoras visuales**  
   Al tener ya todo implementado, pasé a probar el funcionamiento general. Corregí errores, mejoré la experiencia visual con CSS y me aseguré de que el comportamiento fuera coherente en todas las páginas.
6. **Redacción de la memoria del TFG y documentación**  
   Finalmente, documenté todo el proceso: desde el análisis hasta las decisiones técnicas, explicando por qué elegí ciertas tecnologías y cómo resolví los problemas que me encontré.

**Recursos utilizados**

Durante el desarrollo utilicé los siguientes recursos:

* **IDE**: IntelliJ IDEA para el backend y el frontend.
* **Lenguajes y frameworks**: Java con Spring Boot, HTML5, CSS3, JavaScript.
* **Base de datos**: H2 en desarrollo y compatibilidad para PostgreSQL o MySQL en producción.
* **Control de versiones**: Git y GitHub.
* **Documentación y ayuda**: Documentación oficial de Spring, foros como Stack Overflow, y ejemplos encontrados en GitHub.
* **Planificación personal** Para organizarme, usé una lista donde apuntaba lo que tenía que hacer, el orden de importancia y si ya lo había hecho o no.

Esta planificación me ayudó a no perderme en el desarrollo y a tener siempre claro qué pasos debía seguir. Al ir cumpliendo cada fase, me motivaba ver cómo poco a poco el proyecto iba tomando forma.

# Implementación y pruebas

**A. DESARROLLO DE LAS FUNCIONALIDADES DEL PROYECTO**

Durante la fase de implementación me centré en transformar toda la planificación previa en una aplicación funcional y operativa. El desarrollo se dividió principalmente entre el backend, realizado con **Spring Boot**, y el frontend, implementado con **HTML, CSS y JavaScript**, con integración completa mediante peticiones REST.

Una de las funcionalidades clave fue la implementación del **método Pomodoro personalizado**, permitiendo al usuario definir el tiempo de trabajo y descanso según sus preferencias. Además, programé la lógica necesaria para que, al terminar una tarea, se iniciara automáticamente la siguiente en caso de estar encadenadas como subtareas. Esto se traduce en una experiencia de flujo continuo, sin interrupciones, que ayuda a mantener la concentración.

**Configuración de seguridad**

Para proteger las funcionalidades de la aplicación, he implementado un sistema de autenticación basado en tokens JWT (JSON Web Token). Esta solución permite que el backend valide la identidad del usuario en cada petición sin necesidad de mantener sesiones activas en el servidor.

1. **JwtTokenProvider**

Esta clase se encarga de **generar, validar y leer tokens JWT**. Utiliza una clave secreta (SECRET\_KEY) para firmar los tokens, y así garantizar que no se puedan falsificar.

**Funciones principales:**

* **generateToken(Authentication)**: genera un token JWT a partir de los datos del usuario autenticado. En el token incluyo el id del usuario como subject y el nombre de usuario como un claim.
* **isValidToken**: valida que el token tenga una firma válida y no esté caducado.
* **getUsernameFromToken**: extrae el nombre de usuario del token, lo cual se usa luego para cargar los datos del usuario desde la base de datos.

Todo el proceso utiliza la librería jjwt (Java JWT), y se firma con HMAC usando la clave secreta que debe tener al menos 32 caracteres.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**2. JwtFilter**

Esta clase es un filtro que se ejecuta **una vez por cada petición HTTP** que llega a la aplicación. Su función principal es **interceptar la petición, extraer el token JWT** de la cabecera Authorization, y si el token es válido, establecer el usuario autenticado en el contexto de seguridad de Spring.

Extrae el token de la cabecera y valida si dicho token es correcto usando JwtTokenProvider. Si el token es válido, obtiene el nombre de usuario que contiene el token y carga los detalles del usuario con UserDetailsService. Con esos datos, crea un Authentication válido y lo asigna al contexto de seguridad de Spring.

Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.

**3. PasswordEnconderConfig**

En esta clase defino un PasswordEncoder que se usará para **encriptar las contraseñas** de los usuarios y también para compararlas al hacer login. Uso **BCryptPasswordEncoder**, que es uno de los métodos más seguros y recomendados actualmente.

Esto es importante para no guardar contraseñas en texto plano en la base de datos, y además proteger al sistema contra ataques como fuerza bruta.

Interfaz de usuario gráfica, Texto, Sitio web

El contenido generado por IA puede ser incorrecto.

**4. SecurityConfig**

Esta clase es la que configura toda la **seguridad de la aplicación**, incluyendo:

**A. securityFilterChain(HttpSecurity http)**

Define cómo se protegen las rutas:

* **CSRF desactivado** (csrf.disable()): porque no usamos sesiones, sino tokens JWT.
* **CORS configurado** para permitir peticiones desde el frontend (http://localhost:4200).
* **Sesiones desactivadas** (STATELESS): no se guarda estado entre peticiones.
* Se definen rutas públicas (/auth/\*\*, Swagger) y rutas que requieren autenticación (/tareas/\*\*).
* Se añade el JwtFilter antes del filtro por defecto de autenticación de Spring (UsernamePasswordAuthenticationFilter), para que las peticiones se validen con el token JWT.

**B. authenticationManager(...)**

Crea el AuthenticationManager, que es el encargado de procesar el login. Utiliza un DaoAuthenticationProvider, que compara el usuario y contraseña con los guardados en la base de datos y encripta la contraseña usando BCrypt.

**C. corsConfigurationSource()**

Define la configuración de CORS (Cross-Origin Resource Sharing), necesaria para que el frontend (en otro dominio o puerto) pueda comunicarse con el backend. Permito los métodos GET, POST, PUT, DELETE y OPTIONS, así como los headers necesarios para enviar el token (Authorization, Content-Type).

**Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto.**

Funcionalidades CRUD completas para tareas y subtareas, con **visualización jerárquica** y control del estado de finalización.

La posibilidad de **subir una imagen de currículum** desde el perfil del usuario, junto con un campo de “Sobre mí”, lo que aporta un toque más personal a la herramienta.

Notificaciones visuales para avisar al usuario del cambio entre tarea y descanso.

Conexión con **una base de datos MySQL**, gestionada a través de Docker y visualizada con phpMyAdmin durante la fase de pruebas y depuración.

Para que todo esto funcionara correctamente, trabajé de forma modular, dividiendo el código en distintas capas y asegurándome de que cada componente cumpliera con una única responsabilidad. De esta forma, fue más fácil mantener y escalar el proyecto conforme avanzaba.

**B. PRUEBAS UNITARIAS Y DE INTEGRACIÓN**

Durante el desarrollo de la aplicación, he ido realizando pruebas manuales constantes para asegurarme de que todo funcionaba correctamente. Estas pruebas han sido especialmente importantes en las funcionalidades relacionadas con la gestión de tareas y el temporizador Pomodoro, ya que su lógica implicaba distintas transiciones automáticas, encadenamientos entre tareas y cálculos de tiempo que debían ejecutarse con precisión.

Desde el principio, fui comprobando que al crear una tarea esta se guardara correctamente en la base de datos, que apareciera bien en pantalla y que se pudiera completar o eliminar sin errores. También probé la visualización jerárquica de tareas y subtareas, asegurándome de que fueran coherentes visualmente y funcionalmente.

La funcionalidad Pomodoro fue una de las que más pruebas necesitó, ya que implicaba gestionar el tiempo, detectar cuándo terminaba una tarea y activar la siguiente automáticamente. Comprobé que el temporizador se iniciaba con la duración correcta, que respetaba los descansos, y que la tarea hija se activaba sin necesidad de intervención del usuario.

En cuanto al perfil de usuario, verifiqué que se podía actualizar correctamente el campo de “Sobre mí” y que la subida de imagen de currículum funcionaba sin errores, mostrando la imagen inmediatamente tras subirla.

No llegué a utilizar herramientas de testing automáticas como JUnit en este proyecto, ya que preferí centrarme en el desarrollo funcional y en asegurarme de que todo funcionara correctamente desde el punto de vista del usuario. Por eso, opté por realizar pruebas manuales durante todo el proceso, comprobando cada funcionalidad a medida que la iba implementando.

Estas pruebas me permitieron detectar y corregir pequeños errores en el funcionamiento de la lógica del temporizador, en la visualización de las tareas hijas o en la forma en que se actualizaban los datos del usuario, mejorando poco a poco la estabilidad y experiencia final de la aplicación.

**C. CORRECCIÓN DE ERRORES Y OPTIMIZACIÓN DEL RENDIMIENTO**

Durante el desarrollo encontré distintos errores o comportamientos inesperados, que fui corrigiendo según los detectaba. Algunos ejemplos fueron:

* El **temporizador Pomodoro**, que al principio no se detenía correctamente al cambiar de tarea o recargar la página, lo que resolví controlando mejor los setInterval y almacenando correctamente el estado del temporizador.
* La **transición entre tareas encadenadas**, que no funcionaba de forma fluida hasta que implementé una lógica que detecta el fin de una tarea e inicia automáticamente la siguiente.
* En el frontend, también **optimizé el renderizado** de listas de tareas y subtareas, limitando el acceso innecesario al DOM y simplificando funciones repetidas para hacer el código más limpio y eficiente.

En cuanto al rendimiento general de la aplicación, al estar desarrollada con tecnologías ligeras y utilizando una arquitectura basada en contenedores Docker, no he encontrado grandes problemas de carga o velocidad. No obstante, he tratado de mantener todo el código organizado y modular, para facilitar futuras mejoras o ampliaciones.

# Documentación

**A. DOCUMENTACIÓN TÉCNICA**

La documentación técnica recoge toda la información necesaria para entender cómo está construida la aplicación TaskSync por dentro. He estructurado el backend utilizando Spring Boot, con un diseño basado en controladores, entidades y repositorios, lo que me ha permitido mantener el código organizado y fácil de mantener.

Cada entidad principal del sistema (Tarea, Categoria , Usuario y Pomodoro) tiene su correspondiente clase en el backend, y están bien conectadas entre sí. Para acceder a la base de datos, he utilizado Spring Data JPA, lo que me ha simplificado mucho las operaciones CRUD y me ha permitido centrarme más en la lógica de negocio.

En el frontend, he trabajado con HTML, CSS y JavaScript. Me he apoyado bastante en la manipulación del DOM para actualizar dinámicamente las tareas, controlar el temporizador Pomodoro y gestionar las categorías y las tareas. Todo el código JavaScript está dividido en archivos como tareas.js, categorias.js… Cada uno con funciones bien diferenciadas.

También he añadido comentarios en el código para que sea más fácil entender cada parte.

**B. DOCUMENTACIÓN DEL USUARIO**

Desde el punto de vista del usuario, he intentado que TaskSync sea lo más intuitiva posible. Aun así, dejo aquí una pequeña guía para que cualquier persona pueda entender cómo usarla:

* **Inicio de sesión**: El usuario puede acceder con sus credenciales. Una vez dentro, verá el panel de tareas.
* **Gestión de tareas**: Se pueden crear tareas nuevas indicando su nombre, duración, categoría, etc. Las tareas se muestran en una lista jerárquica si tienen subtareas.
* **Funcionalidad Pomodoro**: Al pulsar el botón de iniciar tarea, la tarea comenzará automáticamente y se iniciará el temporizador. Si la tarea tiene una subtarea, esta se iniciará al terminar la anterior.
* **Perfil de usuario**: En el apartado de perfil se puede subir un currículum en formato imagen y escribir un texto sobre uno mismo.
* **Gestión de categorías**: Se pueden crear nuevas categorías indiciando su nombre y su color, al hacerlo aparecerán en dicha vista.

La interfaz es sencilla y está pensada para que cualquier persona pueda empezar a usarla sin necesidad de manuales complejos.

**C. MANUAL DE INSTALACIÓN Y CONFIGURACIÓN**

Para que TaskSync funcione correctamente, he utilizado **Docker** junto con **Docker Compose**, lo que permite levantar todos los servicios necesarios (frontend, backend, base de datos y phpMyAdmin) con un solo comando. Esto facilita mucho tanto la instalación como el despliegue, ya que no hace falta tener nada instalado más allá de Docker.

A continuación, explico paso a paso cómo instalar y ejecutar la aplicación:

**Requisitos previos**

Antes de empezar, es necesario tener instalado en el sistema:

* [Docker](https://www.docker.com/)
* Docker Compose

**Estructura del proyecto**

La estructura básica del proyecto es la siguiente:

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Cómo iniciar la aplicación**

1. **Clonar el repositorio:**

**Con los siguientes comandos en la terminal:**

*git clone* <https://github.com/teresarey8/mi-tfg>

*cd mi-tfg*

1. **Verificar que todos los archivos están en su lugar:**
   * El frontend debe estar dentro de la carpeta frontend.
   * El backend debe estar dentro de la carpeta spring-backend.
   * El archivo docker-compose.yml debe estar en la raíz del proyecto.
   * Crear si hace falta la carpeta uploads, que se usará para almacenar los currículums de los usuarios.
2. **Levantar todos los servicios con Docker Compose:**

**Con el siguiente comando en la terminal:**

*docker-compose up --build*

Esto levantará los siguientes servicios:

* + frontend: Servido con Nginx, accesible en <http://localhost:4200>.
  + backend: Aplicación Spring Boot, accesible en <http://localhost:8081>.
  + mysql: Base de datos MySQL con los datos de acceso definidos.
  + phpmyadmin: Para gestionar la base de datos visualmente, en <http://localhost:8080>.

1. **Acceso a la aplicación:**

Una vez levantado todo, puedes abrir el navegador en http://localhost:4200 y empezar a usar TaskSync. También puedes acceder a la base de datos desde http://localhost:8080 usando:

* + Usuario: teresa
  + Contraseña: 1234
  + Servidor: mysql

**Detalles técnicos del entorno**

* **Base de datos**: MySQL 8, con una base de datos llamada midb y usuario teresa.
* **Backend (Spring Boot)**: expuesto en el puerto 8081, con configuración para conectarse automáticamente a MySQL usando variables de entorno.
* **Frontend**: servido desde Nginx, mostrando directamente los archivos HTML y JS de la carpeta frontend.
* **Persistencia de datos**: Los datos de MySQL se guardan en un volumen (mysql-data), y los archivos subidos (como los currículums) se almacenan en la carpeta uploads.

**Parar los servicios**

Para parar todo el entorno, puedes ejecutar:

*docker-compose down*

Esto apagará y eliminará los contenedores, pero los datos de la base de datos se conservarán.

# Mantenimiento y evolución

**A. PLAN DE MANTENIMIENTO Y SOPORTE**

Aunque TaskSync ha sido un proyecto realizado de forma personal como Trabajo de Fin de Grado, he intentado dejarlo preparado para que pueda mantenerse en el tiempo sin problemas. Para ello, me he asegurado de que el código esté organizado, documentado y estructurado de manera sencilla, lo que facilita posibles futuras modificaciones o arreglos.

El mantenimiento que tendría que llevarse a cabo de forma regular incluye:

* **Revisar que los contenedores Docker funcionen correctamente** y actualizar las imágenes si es necesario.
* **Controlar el espacio ocupado por los archivos subidos**, ya que los currículums en imagen podrían ocupar bastante si el uso crece.
* **Hacer copias de seguridad de la base de datos** si se quiere conservar el historial de tareas y usuarios.
* **Actualizar dependencias** tanto del backend (Spring Boot, librerías de Maven) como del frontend (si se usan nuevas librerías JS en el futuro).

Al estar montado con Docker, el despliegue de nuevas versiones o la reparación de errores es bastante ágil, ya que simplemente hay que rehacer las imágenes con docker-compose build.

**B. IDENTIFICACIÓN DE POSIBLES MEJORAS Y EVOLUCIÓN DEL PROYECTO**

Aunque TaskSync está completamente funcional, a lo largo del desarrollo se me han ido ocurriendo algunas ideas que no me ha dado tiempo a implementar pero que mejorarían bastante la aplicación. Algunas de ellas son:

* **Notificaciones automáticas** al acabar una tarea o descanso, ya sea con sonidos o mensajes visuales bonitos.
* **Historial de productividad**, donde se muestre un resumen semanal o mensual de las tareas completadas y tiempo trabajado.
* **Modo oscuro / claro** para adaptar la interfaz a las preferencias del usuario.
* **Internacionalización**, permitiendo cambiar el idioma de la app fácilmente (por ejemplo, español, inglés, etc.).
* **Control de versiones de tareas**, por si se quiere recuperar información modificada anteriormente.
* **Integración con calendarios externos** como Google Calendar para sincronizar eventos y tareas.

Estas ideas están pensadas tanto para mejorar la experiencia del usuario como para ampliar las posibilidades de uso de la herramienta.

**C. ACTUALIZACIONES Y MEJORAS FUTURAS**

Si se quisiera seguir desarrollando TaskSync en el futuro, estas serían algunas de las prioridades que marcaría:

1. **Pasar el frontend a un framework moderno** como React o Angular para una interfaz más dinámica.
2. **Refactorizar el backend con pruebas automáticas** y añadir test con JUnit para asegurar su fiabilidad a largo plazo.
3. **Montar un sistema de subida de archivos más completo**, permitiendo PDF además de imágenes para el currículum.
4. **Incluir un panel de administración** para gestionar usuarios, tareas o categorías.
5. **Desplegar en la nube**, por ejemplo usando servicios como AWS o Railway para que la aplicación esté accesible desde cualquier lugar.

Estas mejoras no solo harían que la app sea más potente y profesional, sino que también serían un buen paso si quisiera convertirla en un producto más serio o incluso abrirla al público.

# Conclusiones

**A. EVALUACIÓN DEL PROYECTO**

Después de muchos meses de trabajo, puedo decir que desarrollar **TaskSync** ha sido una experiencia intensa pero muy enriquecedora. A lo largo del proyecto he tenido que enfrentarme a retos reales de programación, organización y diseño, lo que me ha permitido aplicar todos los conocimientos adquiridos durante el ciclo formativo.

Estoy bastante satisfecha con el resultado final. La aplicación funciona correctamente, tanto a nivel de backend como de frontend, y permite gestionar tareas usando el método Pomodoro, con características como la anidación de tareas, la transición automática entre ellas, la personalización de tiempos o la gestión por categorías. Además, pude añadir funcionalidades extra como el perfil de usuario con currículum e información personal.

Aunque siempre hay cosas que se pueden mejorar, siento que he sido capaz de crear una aplicación funcional, completa y con una interfaz amigable, lo cual era uno de mis principales objetivos.

**B. CUMPLIMIENTO DE OBJETIVOS Y REQUISITOS**

Desde el inicio, el desarrollo de **TaskSync** ha estado guiado por los objetivos marcados tanto por mí como por los requisitos oficiales del proyecto fin de ciclo. En ese sentido, puedo afirmar con seguridad que **se han cumplido todos los requisitos establecidos**, y en muchos casos incluso se han superado.

A nivel técnico, la aplicación cuenta con **una base de datos relacional** compuesta por varias tablas (usuarios, tareas, categorías, pomodoro), todas con claves primarias y Pomodoro con campos de fecha como la hora de inicio y de fin. El backend, desarrollado en **Spring Boot**, cumple con las medidas mínimas de seguridad exigidas, como la encriptación de contraseñas, validaciones, control de acceso y gestión de sesiones mediante tokens. Además, se han integrado correctamente respuestas con los códigos HTTP adecuados.

El **frontend** se ha realizado en **HTML, CSS y JavaScript**, asegurando que sea responsive y usable desde cualquier dispositivo. He cuidado mucho la experiencia del usuario, creando una interfaz clara e intuitiva que permite gestionar tareas, iniciar temporizadores Pomodoro y visualizar jerarquías entre tareas. También se han implementado **conexiones asíncronas** mediante fetch para mantener la fluidez y evitar recargas innecesarias.

Respecto a los apartados obligatorios:

* Se ha incluido un **registro y login de usuario** completamente funcional.
* He creado un **logotipo original** para la aplicación.
* Hay un apartado de **“Sobre mí”** donde se puede añadir una carta de presentación y un currículum en formato imagen.
* Toda la estructura del proyecto se ha subido a **GitHub** en un único repositorio, con la documentación, un manual para el correcto despliegue de la aplicación, código backend y frontend, y la presentación.

En resumen, todos los **objetivos funcionales y técnicos** planteados en los requisitos del proyecto han sido abordados y completados. Esto incluye desde lo más básico (estructura técnica y seguridad) hasta apartados más personales como la presentación y el portfolio. Además, a lo largo del proyecto he sido capaz de añadir mejoras e ideas propias, lo cual demuestra que no solo he cumplido lo necesario, sino que también he buscado aportar valor añadido al desarrollo.

**C. LECCIONES APRENDIDAS Y RECOMENDACIONES PARA FUTUROS PROYECTOS**

Este proyecto me ha enseñado muchísimas cosas, tanto técnicas como personales. A nivel técnico, he aprendido a trabajar con frameworks robustos como Spring Boot y a integrar un frontend que se comunica eficientemente con el backend. También he mejorado mucho en la estructuración del código, en la organización de los archivos y en la lógica de negocio.

Pero, más allá del código, he aprendido la importancia de **planificar bien las tareas**, de ser constante y de no rendirme cuando algo no sale a la primera. Muchas veces he tenido que buscar soluciones por mi cuenta, leer documentación, hacer pruebas y volver a empezar, y todo eso me ha hecho crecer como desarrolladora.

Si tuviera que dar algún consejo para alguien que empiece un proyecto similar, sería:

* **Divide el proyecto en partes pequeñas y realistas.** Si intentas hacerlo todo de golpe, te agobiarás.
* **Ten claro lo que quieres conseguir desde el principio, pero deja espacio para mejorar ideas sobre la marcha.**
* **No subestimes la importancia del diseño visual** ni de la experiencia de usuario. Una aplicación puede ser muy potente, pero si no es fácil de usar, pierde mucho valor.
* Y por supuesto, **documenta todo lo que puedas**, porque eso te ayudará tanto a ti como a los que vean tu trabajo después.

Resumiendo, este proyecto ha sido una gran oportunidad para demostrar todo lo que he aprendido durante el ciclo y para seguir creciendo en este mundo del desarrollo web, que me encanta. Me voy con muchas ideas, más seguridad en mis capacidades y, sobre todo, con ganas de seguir mejorando.

# Bibliografía y referencias

**A. FUENTES UTILIZADAS EN EL PROYECTO**

Durante la realización de este Trabajo Fin de Grado, he consultado diversas fuentes que han sido fundamentales para la comprensión y desarrollo de la aplicación TaskSync, así como para la implementación de funcionalidades específicas como el método Pomodoro, la gestión de tareas anidadas y el desarrollo con Spring Boot y JavaScript. Entre ellas, destaco:

* Documentación oficial de [Spring Boot](https://spring.io/projects/spring-boot), que me ha servido para entender el framework y su aplicación en la creación de APIs REST eficientes y seguras.
* Guías y tutoriales sobre JavaScript moderno, especialmente para la manipulación del DOM y la gestión de eventos, consultados en sitios como [MDN Web Docs](https://developer.mozilla.org/es/).
* Artículos y recursos sobre el método Pomodoro, para adaptar correctamente la técnica al funcionamiento de la app, incluyendo páginas como [PomodoroTechnique](https://www.pomodorotechnique.com/pomodoro-web-app-saas/).
* Cursos y documentación sobre HTML5 y CSS3 para el diseño de interfaces accesibles y responsivas.
* Repositorios y ejemplos en GitHub relacionados con la gestión de tareas y temporizadores, que me han servido como referencia para la estructuración del código y la interacción entre frontend y backend.

**B. REFERENCIAS Y ENLACES DE INTERÉS**

Para ampliar conocimientos o profundizar en algunos de los temas tratados en este proyecto, recomiendo las siguientes páginas y recursos:

* [Spring Framework Documentation](https://docs.spring.io/spring-framework/docs/current/reference/html/)
* [JavaScript Info](https://javascript.info/), un recurso muy completo para aprender y refrescar conceptos de JavaScript.
* [Stack Overflow](https://stackoverflow.com/), comunidad imprescindible para resolver dudas y encontrar soluciones a problemas comunes durante el desarrollo.
* [GitHub](https://github.com/), especialmente repositorios relacionados con aplicaciones de gestión de tareas y técnicas Pomodoro.
* [FreeCodeCamp](https://www.freecodecamp.org/), plataforma con tutoriales prácticos de desarrollo web.
* Blogs técnicos como [CSS-Tricks](https://css-tricks.com/) para consejos sobre diseño y maquetación.