UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Plataforma web de trabajo colaborativo

Presentado por Álvaro Díez Sáenz en
Universidad de Burgos — 8 de julio de 2024
Tutor: Rubén Ruiz González

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Rubén Ruiz González, profesor del departamento de Digitalización, área de Ingeniería de Sistemas y Automática.

Expone:

Que el alumno Álvaro Díez Sáenz, con DNI 17499664E ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Plataforma web de trabajo colaborativo.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección delque suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 8 de julio de 2024

Vº. Bº. del Tutor:

RUIZ GONZALEZ RUBEN - 71302208T

Firmado digitalmente por RUIZ GONZALEZ RUBEN - 71302208T
Fecha: 2024.07.09 08:28:42 +02'00'

D. Rubén Ruiz González

Resumen

El siguiente Trabajo de Fin de Grado se centra en el desarrollo de una plataforma web colaborativa para la gestión de archivos y tareas de manera individual y entre grupos de trabajo, utilizando tecnologías modernas y escalables además de manteniendo unas buenas prácticas de desarrollo de software, para que sea una base para una posible evolución del proyecto, logrando realmente llegar a una completa plataforma de trabajo colaborativo, para que pequeños grupos puedan utilizar en el desarrollo de sus trabajos.

El objetivo principal del TFG es desarrollar una plataforma web que permita a los usuarios gestionar sus archivos y tareas tanto de forma individual como en grupos de trabajo. Englobados aquí todos los pequeños objetivos individuales como, por ejemplo: implementar en el sistema de gestión de usuarios que permita su registro e inicio de sesión, la gestión de los archivos y tareas de manera tanto individual como en grupo, así como una correcta gestión de los propios grupos.

Para el desarrollo de este trabajo, se ha utilizado una metodología de desarrollo iterativo e incremental, ya que en un principio no estaba claro poder establecer distintas “metas” u objetivos, debido a agentes externos y carga de trabajo externas, por esto se ha abordado el proyecto manejando la carga de trabajo y realizando mejoras y adaptaciones en los diferentes ciclos cortos, asegurando una mejora continua y adaptación a los cambios.

Sobre las tecnologías se han intentado utilizar tecnologías modernas y escalables como, por ejemplo, Flask, SQLite, SQL, Alchemy, entre otras. Todo esto englobando el sistema en un patrón Modelo-Vista-Controlador, para facilitar la separación de responsabilidad y mejora la escalabilidad y mantenimiento del código.

Aun estando abierta a muchas mejoras, el desarrollo de este TFG ha concluido con el desarrollo inicial de una plataforma web colaborativa, escalable y fácil de mantener la cual ha sido desarrollada con tecnologías sencillas y modernas.

Con los resultados obtenidos se cumplen los primeros objetivos plantados, además de plantear una base robusta para las futuras ampliaciones y mejoras que podrían realizarse en el proyecto, pudiendo llegar a ser una plataforma realmente completa para el uso de pequeños grupos en proyectos de trabajo.

Descriptores

Aplicación web colaborativa, Flask, SQLite, MVC, SQL, Alchemy, Desarrollo web, HTML, Bootstrap

Abstract

The following Final Degree Project focuses on the development of a collaborative web platform for the management of files and tasks individually and between work groups, using modern and scalable technologies and maintaining good software development practices, to be a basis for a possible evolution of the project, really achieving a complete collaborative work platform, so that small groups can use in the development of their work.

The main objective of the Final Degree Project is to develop a web platform that allows users to manage their files and tasks both individually and in work groups. Included here are all the small individual objectives such as, for example: to implement in the user management system that allows registration and login, the management of files and tasks both individually and in groups, as well as a correct management of the groups themselves.

For the development of this work, an iterative and incremental development methodology has been used, since at first it was not clear to establish different "goals" or objectives, due to external agents and external workload, so the project has been approached by managing the workload and making improvements and adaptations in the different short cycles, ensuring continuous improvement and adaptation to changes.

Regarding the technologies, we have tried to use modern and scalable technologies such as Flask, SQLite, SQL, Alchemy, among others. All this encompassing the system in a Model-View-Controller pattern, to facilitate the separation of responsibility and improve the scalability and maintenance of the code.

Even being open to many improvements, the development of this Final Degree Project has concluded with the initial development of a collaborative, scalable and easy to maintain web platform which has been developed with simple and modern technologies.

With the results obtained, the first objectives have been met, in addition to providing a robust basis for future extensions and improvements that could be made to the project, which could become a truly complete platform for the use of small groups in work projects, in addition to creating a robust base for future extensions and improvements that could be made in the project, being able to become a really complete platform for the use of small groups in work projects.

Keywords

Collaborative web application , Flask, SQLite, MVC, SQL, Alchemy, web development, HTML, Bootstrap

Índice general

Índice de figuras

Índice de tablas

1. Introducción

En la actualidad, la gestión eficiente de archivos y tareas es fundamental tanto en entornos personales como profesionales. Debido a la constante tendencia hacia el trabajo colaborativo y remoto, esto genera una necesidad de utilizar distintas herramientas que faciliten la organización y el intercambio de información de manera cómoda, rápida y accesible. En este trabajo se va a realizar el desarrollo de una aplicación web colaborativa diseñada para compartir archivos y gestionar tareas tanto para el propio usuario de una forma individual como con la posibilidad de la creación y uso de grupos para el trabajo colaborativo.

Durante estos últimos años el avance de las tecnologías web, y las necesidades de los usuarios han conllevado el nacimiento de numerosas aplicaciones para la gestión de proyectos, gestión de documentos y en general la coordinación de equipos. Muchas de estas herramientas como pueden ser Office 365, Trello, o Notion entre otras, se han vuelto indispensables en los grupos de trabajo, ya que todos estas aportando valor y funcionalidad a los trabajos en equipo. Estas herramientas en general se enfocan en ser el centro de trabajo para los distintos grupos.

El propósito principal de este trabajo es proporcionar una plataforma donde tanto los usuarios de forma individual como en sus grupos de trabajo puedan gestionar sus archivos y tareas de una forma sencilla, distinguiéndose de otras plataformas sobre todo por su sencillez y escalabilidad mediante el uso de herramientas y tecnologías de código libre, siendo así además de una primera plataforma, un entorno de desarrollo accesible y preparado para nuevas expansiones, mejoras y líneas futuras.

Por lo cual, en este trabajo no solo se busca la creación de una plataforma sencilla y práctica para la gestión de archivos y tareas tanto individualmente como en grupo, si no tambien servir como referencia y guía para poder continuar agregando soluciones y explorar el uso de las distintas técnologias modernas y escalables como las utilizadas durante todo el desarrollo del trabajo.

2. Objetivos del proyecto

El objetivo del trabajo es el desarrollo de una aplicación web la cual permita tanto compartir información entre los integrantes de un grupo, como también de manera personal por cada usuario.

2.1. Objetivos del software

1. **Facilitar el intercambio de archivos**: Desarrollar una plataforma que permita a los usuarios subir, descargar y gestionar archivos de manera eficiente y segura. Cada usuario deberá tener la capacidad de mantener su propio espacio de almacenamiento, así como compartir archivos con otros usuarios.
2. **Gestión de tareas**: Implementar una funcionalidad que permita a los usuarios crear, asignar y gestionar tareas. Esto incluirá la capacidad de definir descripciones de tareas.
3. **Soporte para trabajo colaborativo**: Permitir la creación y gestión de grupos de usuarios, facilitando la colaboración en proyectos compartidos. Los usuarios podrán compartir archivos y tareas dentro de sus respectivos grupos, promoviendo un entorno de trabajo en equipo.
4. **Seguridad y privacidad**: Asegurar que solo los usuarios autenticados tengan acceso a sus propios archivos y tareas. Implementar controles de acceso que protejan la privacidad y la integridad de los datos del usuario.

2.2. Objetivos técnicos

1. **Implementación con Flask**: Utilizar el framework Flask para desarrollar una aplicación web ligera y modular. Flask es conocido por su simplicidad y flexibilidad, lo que lo hace ideal para este tipo de proyectos.

2. **Diseño responsivo con Bootstrap**: Emplear Bootstrap para diseñar una interfaz de usuario que sea atractiva y responsiva, garantizando que la aplicación se vea y funcione bien en una variedad de dispositivos, incluidos móviles y tabletas.

3. **Gestión de bases de datos con SQLite**: Utilizar SQLite como sistema de gestión de bases de datos para almacenar información de usuarios, archivos y tareas de manera eficiente y segura. SQLite es una opción adecuada debido a su simplicidad y facilidad de integración con Flask.

4. **Autenticación y autorización**: Implementar mecanismos de autenticación y autorización utilizando Flask-Login. Esto garantizará que solo los usuarios registrados puedan acceder a la aplicación y gestionar sus archivos y tareas de manera segura

5. **Gestión de archivos**: Integrar métodos seguros para la carga y almacenamiento de archivos, utilizando herramientas compatibles con Flask para evitar problemas relacionados con la seguridad de los archivos su acceso y su correcto almacenamiento.

6. **Modularidad y escalabilidad**: Desarrollar la aplicación con un estilo de programación modular, el cual permita una fácil ampliación y mantenimiento. Mediante una buena generación de la estructura del trabajo se intentará facilitar la incorporación de nuevas funcionalidades y poder adaptarse a futuros requisitos para el desarrollo de líneas futuras de trabajo.

3. Conceptos teóricos

En esta sección se van a revisar de una forma breve algunos de los distintos conceptos teóricos que deben conocerse para el correcto entendimiento del trabajo.

HTTP

Llamado Protocolo de transferencia de hipertexto, este es el protocolo que permite realizar peticiones de recursos y datos. Tiene una estructura cliente-servidor, lo cual quiere decir que estas peticiones son iniciadas por el cliente, el que recibirá los datos.[1]

HTTP define la estructura y transmisión de los mensajes para poder establecer comunicación entre un navegador web y un cliente web, también establece las reglas para la comunicación entre el cliente y el servidor, permitiendo tanto el envío de datos, texto, imágenes y más archivos multimedia

Funcionamiento

HTTP es un protocolo que no tiene estado, por lo que cada solicitud que se realiza es independiente e individual y no guardar información sobre peticiones anteriores. Las peticiones son enviadas por el agente usuario y recibidas por el servidor el cual gestiona esa petición y envía una respuesta.

Métodos HTTP

- **GET**: Solicita una serie de datos al servidor.
- **POST**: Envía una serie de datos al servidor.
- **PUT**: Remplaza los datos especificados.
- **DELETE**: Elimina los datos especificados.

REST

REST es un estilo de arquitectura de software para sistemas hipermedia distribuidos. Esta arquitectura utiliza los protocolos y principios de la web, como hemos visto previamente son los ya definidos por HTTP, y de esta forma REST define la manera en que los sistemas se comunican entre sí. El uso de REST permite tener un estilo de arquitectura que facilite la creación de sistemas web escalables y fáciles de mantener.[2]

Funcionamiento

REST, al estar definido por HTTP al igual que este, no tiene estado, y a si los datos se exponen como recursos, los cuales están identificados por URLs únicas. Sobre estos recursos se realizan las operaciones mediante los métodos HTTP que se han mencionado anteriormente.

1. **Solicitud HTTP del Cliente**: El cliente envía una solicitud a una URL especifica los distintos métodos que ya se han visto de HTTP
2. **Recepción en el servidor**: Se recibe la solicitud, y el servidor la procesa y realiza las operaciones correspondientes con los datos.
3. **Respuesta del servidor**: Cuando se ha procesado la solicitud, este responde con el recurso correspondiente.
4. **Recepción en el cliente**: El cliente recibe la respuesta y se actualiza la interfaz de usuario, almacenan datos locales o se realizan las acciones necesarias.

La arquitectura REST, proporciona una alta escalabilidad de los proyectos y un fácil mantenimiento ya que separa claramente el cliente del servidor para facilitar gestión independiente de cada componente. Esto hace a REST una opción muy popular para la construcción de AIS webs, o aplicaciones web.

Framework

Un Framework, también llamado marco de trabajo, es una base estructurada para el desarrollo de aplicaciones. Estos proporcionan componentes, bibliotecas y herramientas predefinidas para agilizar el proceso de desarrollo, el uso de un framework ofrece distintas ventajas:[3]

- **Reducción de tiempo del desarrollo**: Al incluir distintos componentes y bibliotecas no es necesario construir todo desde cero.
- **Estandarización**: Los framewoks marcan distintos estándares a seguir, lo cual a futuro genera un código más consistente y de fácil mantenimiento.
- **Escalabilidad**: Al tener los fremeworks una estructura organizada y modular, esto en gran medida facilita la escalabilidad de los proyectos, por lo que permiten añadir o modificar funcionalidad de una forma sencilla y afectar directamente a todo el proyecto.

Existen dos tipos principales de Frameworks:

- **Frameworks Front-End:** Estos están diseñados para centrarse en la interfaz y la experiencia de los usuarios en las aplicaciones web, existen diversos Frameworks de este tipo, como Angular, React o Bootstrap.
- **Frameworks Back-end:** Estos están diseñados para gestionar la lógica del lado del servidor, así como el manejo de la base datos, gestión de rutas, autenticación, etc… Existen muchas alternativas y para casi todos los lenguajes, en el caso de Python algunas de estas son Flask, Django o FastApi

Object-Relational Mapping (ORM)

Object-Relational Mapping (ORM) es una técnica de ingeniería de software la cual es utilizada para poder interactuar con una base de

datos relacional, utilizando un lenguaje de programación orientado a objetos (POO)

ORM brinda la posibilidad de en lugar de escribir consultas SQL directamente en el código, poder interactuar directamente con estas BD como objetos del lenguaje de programación, esto simplifica en gran medida el desarrollo y mantenimiento de la aplicación.

Este actúa como intermediaria entre la base de datos relacional y el lenguaje de programación, se debe mapear los objetos del código con las tablas correspondientes en la base de datos y los atributos de estos como las columnas de la tabla.

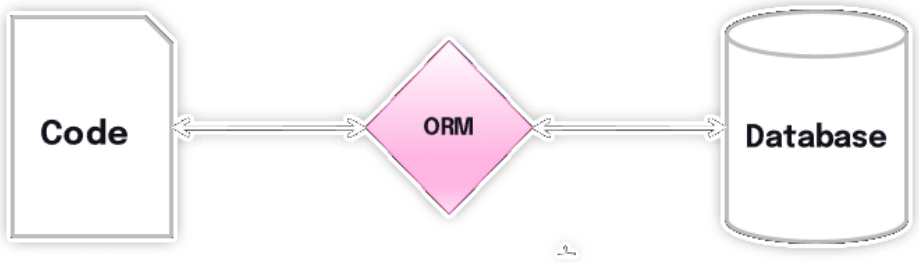

Ilustración 3.1 Estructura ORM[4]Para poder trabajar con las tablas de la base de datos se utilizan las operaciones CRUD (Create, Read, Update, Delete) estas se realizan sobre los objetos y el ORM las traduce en las consultas SQL adecuadas.

- **Create**: Se utilizada para añadir nuevos registros en la base de datos. Es decir, realiza la instrucción “INSERT INTO”
- **Read**: Se utiliza para leer y recuperar uno o más registros de la base de datos. Por lo que realiza la instrucción “SELECT”
- **Update**: Se utiliza para modificar datos existentes en la base de datos. Realiza la instrucción “UPDATE”
- **Delete**: Se utiliza para eliminar uno o más registros de la base de datos. Ejecuta la instrucción “DELETE FROM”

Modelo-Vista-Controlador (MVC)

El patrón de diseño Modelo-Vista-Controlador (MVC) es un patrón de arquitectura de software ampliamente utilizada en el desarrollo de aplicaciones de software, sobre todo en el ámbito de aplicaciones web. Su objetivo es separar la lógica de trabajo y la gestión de los datos, de la interfaz de usuario y el control del flujo de la aplicación, facilitando el mantenimiento y la escalabilidad del software. [5]

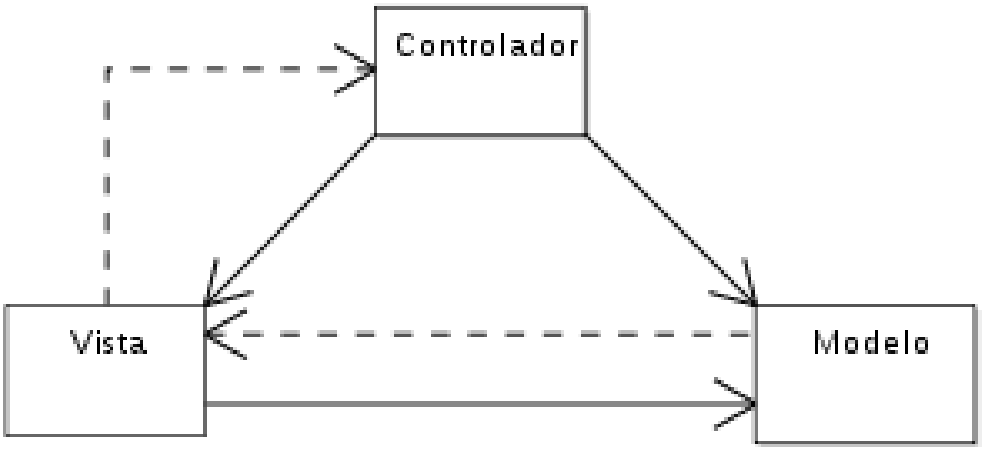

Ilustración 3.2 Estructura MVC [6]

Modelo: en este componente se incorporan los datos y la lógica de la aplicación. Este es el responsable de la gestión y utilización de los datos, así como de la implementación de estructuras. En las aplicaciones web, el modelo normalmente interactúa con la base de datos para recuperar, almacenar y actualizar la información.

Vista: este componente es el responsable de la representación de la información para la interfaz de usuario. Se encarga de renderizar y mostrar la interfaz de usuario, representado los datos proporcionados por el modelo de manera adecuada. En las aplicaciones web, las vistas se implementan utilizando tecnologías como por ejemplo HTML, CSS y JavaScript.

Controlador: este actúa como intermediario entre el modelo y la vista. Es el responsable de recibir las entradas desde la interfaz de usuario, procesarlas y notificar tanto al modelo como la vista en consecuencia. En las aplicaciones web, los controladores gestionan las solicitudes HTTP, ejecutan la lógica apropiada y determinan qué vistas deben ser presentadas.

Interacción entre los componentes

1. **Usuario interactúa con la vista:** Se interactúa con la

aplicación mediante la interfaz de usuario.

2. **Vista envía solicitud al controlador:** Llamada al controlador que puede incluir entrada del usuario.
3. **Controlador procesa la solicitud:** El controlador interactúa con el modelo de datos.
4. **Modela realiza operaciones con los datos:** Realiza operaciones en la base de datos si fuera necesario.
5. **Controlador elige vista para representar el resultado:** Tras realizar operaciones, el controlador selecciona la vista que devuelve al usuario.
6. **Vista se actualiza y muestra la interfaz correspondiente:** Se muestra la interfaz de usuario, que ha proporcionado el controlador.

4. Técnicas y herramientas

En esta sección veremos las distintas técnicas y herramientas de desarrollo utilizadas para llevar a cabo el proyecto. A lo largo del desarrollo, se han considerador diversas alternativas, evaluando sus características, ventajas y desventajas y así finalmente se han seleccionado aquellas que se adaptan mejor a los requisitos del proyecto. A continuación, se listan las distintas herramientas seleccionadas.

Herramientas	Web	BD	Memoria
HTML5	X		
CSS3	X		
Bootstrap	X		
Python	X		
Flask	X	X	
SQLite		X	
Visual Studio Code	X	X	
Git	X	X	
GitHub	X	X	
Office 365			X

Tabla 4.1: Herramientas y tecnologías utilizadas

4.1 Lenguaje de programación

Python ha sido el lenguaje de programación elegido para este proyecto debido a su simplicidad y legibilidad. Python brinda una sintaxis clara y concisa la cual permite a los desarrolladores desarrollar código de manera eficiente y rápida. Además, Python tiene una gran comunidad de desarrolladores y una gran cantidad de bibliotecas y recursos disponibles, por lo que se facilita la resolución de problemas y el acceso a infinidad de herramientas ya desarrolladas para este lenguaje

Entre esta cantidad de bibliotecas de las cuales dispone Python disponemos de Flask, el microframework elegido para el backend. Flask, al estar desarrollado en Python, se beneficia de todas las ventajas del lenguaje, brindando así una rápida implementación de funcionalidades y una fácil integración con otras tecnologías utilizadas en el proyecto.

Alternativas Consideradas:

- **JavaScript (Node.js)**: Node.js es popular para el desarrollo backend debido a su capacidad para manejar múltiples conexiones simultáneamente. Sin embargo, la elección de Python se basó en su simplicidad y mi familiaridad con el lenguaje.

4.2 Desarrollo web

HTML y CSS son fundamentales para la creación de la estructura y el diseño visual de la aplicación web. HTML se utiliza para definir la estructura de las páginas web, incluyendo y declarando todos los elementos necesarios en el desarrollo de una página web y CSS se utiliza para estilizar y mejorar la apariencia visual de las páginas web, permitiendo personalizar colores, fuentes, tamaños y la disposición de los elementos siendo estos dos los principales lenguajes para el desarrollo web.

A su vez para potenciar un diseño más responsivo se ha utilizado **Boostrap**, mediante un diseño de cuadrícula este nos permite generar diseños mucho más responsivos que se adaptan a diferentes tamaños de pantalla. En la actualizad eso es necesario para asegurar la accesibilidad de la aplicación en dispositivos móviles y tablets. Bootstrap ofrece una amplia gama de componentes y utilidades CSS predefinidos, como botones, formularios, modales y menús de navegación, lo que permite acelerar el desarrollo frontend y asegura una apariencia más coherente y profesional en toda la aplicación.

Al igual que el lenguaje Python, en Bootstrap también disponemos con una extensa documentación y una comunidad activa, lo que proporciona un buen suporte durante el desarrollo. La facilidad de uso y la disponibilidad de numerosos ejemplos y recursos en línea han sido factores claves para la elección de este en el desarrollo del trabajo.

Alternativas Consideradas:

- **Materialize**: Basado en Material Design de Google, Materialize ofrece un estilo minimalista y moderno. Aun así, Boostrap ha sido la elección dada la familiaridad y su popularidad, sumado a su extensa documentación y la cantidad de proyectos ya desarrollados.

4.3 Framework Backend

Flask es un microframework de desarrollo web para Python conocido por su simplicidad y flexibilidad. Este permite el construir aplicación de una manera fácil y modular, a diferencia de otros Framework más grandes Flask es minimalista y no necesita unas estructuras muy específicas en la aplicación. Además, este facilita la creación de rutas para manejar todas las solicitudes HTTP (GET, POST) también permite renderizar las plantillas HTML y gestionar directamente la lógica del servidor.[7]

Flask es el componente principal del backend de la aplicación. Fue elegido por varias razones clave:

- **Ligereza y Modularidad**: Flask es un Framework minimalista que brinda la posibilidad de únicamente añadir solo los componentes necesarios, permitiendo de manera continua agregar nuevos componentes si el desarrollo de la aplicación lo requiere. Esta característica es ideal para trabajos como este donde se busca mantener la simplicidad sin sobrecargar el sistema.
- **Facilidad de Uso**: Flask es un considerado un microframeworks y tiene una curva de aprendizaje sencilla y sobre todo un gran comunidad y documentación, lo que facilita la implementación rápida de nuevas funcionalidades. Gracias a esto, es posible comenzar a trabajar con Flask en poco tiempo y de forma fácil, además de poder encontrar soluciones a problemas comunes fácilmente.
- **Amplia biblioteca**: Debido a su activa comunidad Flask tiene una amplia biblioteca al igual que Python todo disponible como código libre, por esto Flask permite la integración fácil de extensiones para manejar tareas de uso común como son por ejemplo la autenticación (Flask-Login), la interacción con bases de datos (Flask-SQLAlchemy), entre otras muchas extensiones, lo que agiliza el desarrollo y la implementación de nuevas características.
- **Versatilidad**: Al ser Python un lenguaje multiparadigma, es decir que soporte tanto una programación estructura como una programación orientada a objetos, Flask también dispone de esta característica, siendo así adecuado tanto para scripts simples web como para aplicaciones webs más completas o de análisis de datos.

Alternativas Consideradas:

- **Django**: Django es un framework de Python muy conocido en la comunidad el cual incluye una gran cantidad de funcionalidades listas para usar. Sin embargo, disponer de tanas funcionalidad quizás sea excesiva para un proyecto que no requiere tanto como las que este ofrece. La cantidad de carga que esto puede generar hace Flask una opción más considerable para este trabajo.
- **FastAPI**: FastAPI es conocido por su rendimiento y por utilizar Python asíncrono, lo que lo hace muy adecuado para aplicaciones que requieren alta concurrencia. A pesar de estas ventajas, la prioridad en la búsqueda de framework fue la simplicidad y familiaridad, por lo que Flask sigue dando una visión más adecuada para el trabajo.

4.3 Base de datos

SQLite es la base de datos que se ha utilizado en base a su simplicidad y ligereza. No requiere la configuración de un servidor de base de datos, lo que facilita el desarrollo y despliegue. SQLite se integra fácilmente con Flask a través de Flask-SQLAlchemy [8], este es uno de los ejemplos de la gran variedad de extensiones de las que dispone Flask, gracias a SQLAlchemy podemos utilizar Object-Relational Mapping (ORM) como ya se ha comentado previamente, esto brinda la posibilidad de trabajar con los modelos como si de objetos Python se tratasen, gracias al uso de esta técnica conseguimos simplificar en gran medida la interacción con la base de datos.

SQLite es ideal para proyectos de tamaño mediano, ya que ofrece suficiente capacidad y rendimiento sin la complejidad adicional de sistemas de bases de datos más avanzados. Su naturaleza de archivo único también facilita la portabilidad y la copia de seguridad de la base de datos.

Alternativas Consideradas:

- **PostgreSQL**: PostgreSQL es una base de datos muy poderosa que ofrece características avanzadas y es altamente escalable. Sin embargo, añade una capa de complejidad innecesaria para este proyecto, que se beneficia de la simplicidad de SQLite.
- **MySQL**: MySQL es otra opción robusta y ampliamente utilizada. Aunque es una buena opción para proyectos más grandes, la simplicidad y facilidad de uso de SQLite lo hicieron más adecuado para este proyecto.

4.4 Entorno

Visual Studio Code es el entorno de desarrollo integrado (IDE) utilizado, este es un entorno muy familiar y el cual ya se ha utilizado en otros proyectos, este entorno se caracteriza principalmente por sus características avanzadas y facilidad de uso [9]:

- **Extensibilidad**: VS Code permite la instalación de extensiones para soporte de Flask, Python, Git, y otras herramientas utilizadas en el proyecto, mejorando de esta manera la productividad.
- **Facilidad de Uso**: Ofrece una interfaz intuitiva y distintas herramientas que facilitan la escritura, depuración y gestión del código.
- **Popularidad y Comunidad**: Al igual que el resto de tecnologías utilizadas en este trabajo, VS Code también dispone de una gran y activa comunidad de desarrolladores los cuales proporcionan un gran soporte y recursos, facilitando la resolución de problemas y la optimización del trabajo.

Alternativas Consideradas:

- **PyCharm**: Un IDE muy popular entre los desarrolladores de Python, ofrece características avanzadas y depuración integrada. Sin embargo, su versión completa es de pago y su peso puede ser excesivo para un proyecto que busca simplicidad.
- **Sublime Text**: Es un editor rápido y ligero, pero carece de algunas de las características avanzadas y la extensibilidad de VS Code, lo que lo hace menos adecuado para un desarrollo como el de este proyecto.

4.5 Control de versiones

Como el proyecto iba a mantener su control de versiones mediante **Git** [10], era necesario una plataforma de donde se puedan alojar todos estos cambios, la plataforma utilizada ha sido **GitHub** la cual es una perfecta plataforma de alojamiento de repositorios y de las más utilizadas. Estas herramientas permiten tener un seguimiento detallado y organizado del proceso realizado.

Tanto el uso de Git como GitHub brinda a todos los proyectos una serie de ventajas para la gestión de código como para la colaboración.

- **Historial de cambios**: Git nos permite realizar un seguimiento de todos los cambios en el código, cada cambio queda registrado como un “commit”, esto brinda la posibilidad de ver los cambios que se han realizado y cuando han sido realizados.
- **Revertir cambios**: Git permite revertir a versiones anteriores del código de una manera sencilla. Estos son útiles cuando se decide descartar una implementación o solucionar errores que han surgido debido a algún mal desarrollo.
- **Ramas de trabajo**: Git facilita en gran medida la creación de nuevas ramas de trabajo, para poder desarrollar o experimentar con nuevas características sin afectar a la versión principal y funcional del proyecto. Cuando las ramas ya han sido verificadas están pueden unirse a la rama principal.
- **Alojamiento centralizado**: GitHub ofrece un lugar centralizado para alojar el código fuente del proyecto. Lo que centraliza todos los cambios en un solo lugar el cual es accesible para todos los colaboradores.
- **Seguimiento de Issue**s: GitHub también ofrece un sistema de seguimiento de problemas, el cual permite reportar, discutir y plantear soluciones para los distintos problemas que se encuentran

4.4 Metodología

Durante todo el trabajo se ha planteado un desarrollo iterativo y e incremental [11], ya que al tratarse del Trabajo de Fin de Grado este se realiza de manera individual, además debido a agentes externos y cargas de trabajo no se podía asegurar que se completase al 100% unas “metas” u “objetivos” semanales. Por eso con este enfoque es posible desarrollar el trabajo en distintos ciclos cortos, en los que cada uno se consigue una mejora continua y adaptación a los cambios.

Con el desarrollo iterativo e incremental se realizan repeticiones de ciclo de desarrollo es decir distintos ciclos cortas donde se revisa el trabajo y también se agrega funcionalidad de forma gradual

Por lo que el ciclo de vida del proyecto ha seguido 3 fases distintas, las cuales estas se han repetido de manera iterativa, en cada ciclo corto de trabajo.

1. **Análisis de Requisitos**: En esta fase de identifican las necesidades y los requisitos funcionales y no funcionales de la parte que se plantea desarrollar
2. **Diseño**: Se elaboran distintos diagramas de arquitectura o E/R sobre la base de datos, para de esta forma poder panificar como se van a estructurar los nuevos componentes agregados al sistema.
3. **Implementación**: Se desarrolla esta funcionalidad planteada realizando pequeños incrementos. Con cada iteración se introducen nuevas características y mejoras de las ya existentes
4. **Pruebas**: En cada una de las interacciones finalmente se realizan pequeñas pruebas unitarias y de integración para asegurar que la nueva funcionalidad se ha integrado correctamente con lo ya previamente desarrollado.

5. Aspectos relevantes del desarrollo del proyecto

En el siguiente apartado va a centrarse en la información de aspectos relevantes en el desarrollo del proyecto, el trabajo se centra en la creación de una plataforma colaborativa sencilla, intuitiva y funcional que pueda ser utilizado por cualquier usuario. A continuación, se van a detallar los aspectos más relevantes de este desarrollo del proyecto, el cual para nada ha sido una cosa trivial ya que durante las diferentes interacciones se fueron conociendo y evaluando distintas tecnologías, y tomando decisiones en cada fase del desarrollo.

Fase de Análisis

Cuando se comenzó el proyecto no se estaba seguro de qué tecnologías se iban a utilizar exactamente, por lo cual durante esta fase se plantearon las necesidades del trabajo y a continuación se investigaron los distintos lenguajes, Framework o tecnologías que se considerasen adecuadas para el desarrollo del trabajo.

Lo primero se identificaron las principales necesidades y requisitos del proyecto, en este primer análisis surgieron 4 principales requisitos del proyecto.

1. **Gestión de Usuarios**
2. **Gestión de archivos**
3. **Gestión de tareas**
4. **Gestión de grupos**

Estos fueron los primeros requisitos que guiaron la implementación y diseño del trabajo.

Una vez definidos los requisitos se comenzó con la búsqueda de las tecnologías más adecuadas para el desarrollo del trabajo.

Python y Flask: Python es un lenguaje el cual ya conocía y ha sido previamente utilizado para otros proyectos, este brinda una gran comunidad y es un lenguaje extremadamente versátil, por lo que era una alternativa muy a tener en cuanto para el desarrollo del backend de esta aplicación, ya que como se ha dicho previamente la sencillez, modularidad, y escalabilidad del del proyecto son los valores principales del trabajo. Siendo Python una opción a tener en cuenta, se barajaron distintos Frameworks de los que este dispone. Aquí nos encontramos una gran variedad de Frameworks, pero Flask siendo un microframeworks flexible, modular y ligero era la opción que mejor se adaptaba al proyecto. Permitiendo así un desarrollo eficiente y modular.

SQlite y ORM: A la hora de seleccionar la base de datos utilizada, también se evaluaron distintas opciones, entre ellas se barajaron opciones como MySQL, PostgreSQL, o incluso el uso de almacenamiento y BD externas como ofrece por ejemplo la solución “Firebase”, pero como ya se ha dicho previamente el objetivo principal del trabajo era buscar la simplicidad y facilidad de configuración.[11]

Por lo tanto, tras esta evaluación se decidió el uso de SQLite, el cual aportaba los valores principales del trabajo, sumada a una gran flexibilidad y portabilidad. Sumado a esto Flask también nos brindaba una gran solución para acompañar a SQLite, como es SQLAlchemy, esta biblioteca permite actuar como un ORM, lo cual daba una capa de abstracción al proyecto y facilito mucho el trabajo de implementación de la base de datos con Python, ya que el uso de SQLAlchemy permite el trabajo de Objetos con Python en lugar de tener que realizar las consultas directamente sobre la base de datos. Ademas el uso de SQLAlchemy no solo funciona correctamente con SQLite, sino que además facilita la migración a distintas bases de datos más robustas como hemos comentado previamente, sin tener que realizar grandes cambios en la lógica de la aplicación.

Fase de Diseño

Arquitectura del Sistema

En la fase de diseño, lo más interesante era encontrar una arquitectura que permitiera la modularidad y escalabilidad del sistema, para esto era necesario mantener una separación entre la lógica, la presentación y los datos. Flask proporcionaba la flexibilidad necesaria para un desarrollo escalable y modular.

Por esto y debido a que Flask se brinda muy cómodamente a ello, se buscó una arquitectura basada en el patrón Modelo-Vista-Controlador (MVC), para de esto modo tener una clara separación de los distintos componentes del trabajo. En nuestro caso adaptamos los distintos componentes del MVC para nuestro trabajo.

- **Modelo**: En este componente es donde se aloja toda la lógica de datos y reglas del trabajo. En el ámbito del trabajo SQLAlchemy fue el utililzado para manejar las interacción con la base de datos. Esto permitió el uso de ORM defininiendo distintos objetos en Python, para poder de este modo interactuar de forma sencilla con la Base de datos, facilitando la creación, modificación y eliminación de los datos, y pudiendo realizar consultar complejas sin escribir directamente SQL.
- **Vistas**: Este componente es el responsable de toda la presentación de los datos, mediante el uso de plantillas de Flask, en nuestro trabajo vemos el uso de varias plantillas “.html” las cuales permiten directamente sobrescribir con los datos necesarios y ser devueltas al usuario final.
- **Controladores**: Este componente es el responsable de manejar la lógica de la aplicación además de la interacción con el usuario. En nuestro trabajo todas estas rutas están definidas en “app.py” y estas actúan como controladores recibiendo las peticiones, e interactuando con los modelos necesarios, para posteriormente devolver las vistas correspondientes al usuario.

Gestión de Usuarios y autentificación

La autentificación y gestión de los usuarios es uno de los puntos más importantes del trabajo, ya que es este el principal objetivo de este, una buena gestión de los usuarios y sus correctos espacios de trabajo. Tras evaluar distintas opciones, de nuevo Flask nos brinda una solución mediante su librería “Flask-Login”. Esta nos brinda directamente la capacidad de gestionar las sesiones de usuarios de una manera segura. Flask-login, ofrece varias medidas de seguridad, además de incluir directamente funcionalidades para la autentificación de los usuarios.

1. **Gestión de sesiones de usuario**: Flask-login permite manejar de una forma cómoda tanto el inicio, como el cierre de sesión de los usuarios. Para esto se utilizan los métodos “login_user()” y “logout_user()”, facilitando de este modo un inicio y cierre de sesión seguro y eficiente.
2. **Modelos de usuario personalizados**: Flask-Login es capaz de integrarse fácilmente con los modelos personalizados de usuarios decritos en el archivo “models.py” como ocurre con el modelo “Usuario”, gracias a esto puede accederse a las propiedades especificas desde usuario como su ID, o cualquiera de sus datos.
3. **Decoradores de Rutas:** Flask-Login permite el uso de decoradores como por ejemplo “@login-requieres” con elc ual podemos proteger las distintas rutas y vistas para que solo puedan ser visualizadas por usuarios previamente autenticados. Si un no autentificado quiere acceder a estar rutas “Flask-login” es capaz de redirigir estos directamente a la página de inicio de sesión.

Además de estas funcionalidades que nos proporciona “Flask-login” para garantizar la integridad de las sesiones de usuarios, también se incorporó el uso de contraseñas seguras mediante el almacenamiento cifrado de estas gracias a “werkzeug” lo que protege frente ataques de fuerza bruta o “hash cracking”.

Estructura Base de datos

Cuando se planteó el diseño de los modelos de la base de datos, se buscó aplicar los principios de normalización para evitar redundancias y poder asegurar la integridad de los datos.

Las tablas se diseñaron pensando en seguir las formas normales hasta la (3NF) tercera forma normal. Para las relaciones entre usuarios, archivos, tareas y grupos también se buscó el uso de claves foráneas y tablas intermedias de relación muchos a muchos (N:N), para poder gestionar estas relaciones algo más complejas de las entidades.[13]

La **primera Forma Normal**, requiere que todas las columnas tengan únicamente valores atómicos además de cada valor de las columnas sea del mismo tipo.

- Como se ve se cumple esto en todas las tablas ya que solo contienen valores atómicos en cada columna
- No existen conjuntos de valores en ninguna

La **segunda Forma Normal**, requiere que la base de datos esté en (1NF), cosa que ya se había realizado y que todas las columnas no clave sean dependientes únicamente de la clave primaría completa.

- Todas las tablas, cumplen que sus columnas y dependen únicamente de la clave primaria de cada tabla “id”
- Las tablas de relaciones muchos a muchos disponen de clave compuesta, y no tienen más columnas clave

Para finalmente conseguir la **tercera Forma Normal** (3NF) [12], se requiere que la base de datos esté en (2NF), y que todas las columnas no clave, sean únicamente dependiente de la clave primaria y no de otras columnas para evitar dependencias transitivas

- No existen dependencias transitivas ya que las únicamente dependen de su clave primaria “id”

6. Trabajos relacionados

En el ámbito de las plataformas colaborativas, en la actualidad han nacido numerosos proyectos, la mayoría con soluciones de código cerrado, todo esto son herramientas centradas en facilitar la gestión de tareas, la comunicación, y el intercambio de archivos entre usuarios. A continuación, veremos algunas de estas herramientas con una pequeña descripción de sus características y funcionalidades principales.

- **Sharepoint**: Sharepoint es una parte del paquete Office 365 ofertado por Microsoft, este paquete combina todos los elementos necesarios para un correcto trabajo colaborativo, como es almacenamiento, gestión de tareas, llamadas, flujos de trabajo, además de incluir en si todas las herramientas office, tanto en versión escritorio como en versión Web. Sin lugar a duda el paquete Office 365 es una de las mejores herramientas para el trabajo colaborativo en la actualidad [13].
- **Trello**: Trello es una herramienta muy popular la gestión de proyectos y tareas, mediante un sistema de tableros y tarjetas que permite una muy sencilla visualización de las tareas además de establecer fechas límite y seguir el progreso de las tareas, esta es una muy buena línea futura para mejorar y dar más utilidad a la gestión de tareas en la aplicación. Trello no esta tan centrado en la gestión de archivos, aunque si tiene una pequeña gestión de estos, todo esto hacen de ella una perfecta aplicación para la gestión de tareas y proyectos [14].

- **Slack:** Slack es una plataforma de comunicación para los equipos que permite la gestión de canales tanto públicos como privados, el cual facilita la organización entre equipos, su principal característica es la capacidad de soportar la integración de una gran cantidad de aplicaciones y servicios, por ejemplo, herramientas de Google, como pueden ser Drive, Calendar, y también distintas herramientas de Office, como pueden ser OneDrive o Sharepoint. Por lo que Slack es una plataforma ideal para gestionar trabajos en equipo donde se requiera el uso de distintas plataformas colaborativas, y es el lugar idóneo para unificarlas todas [15].

- **Notion**: Notion es una plataforma centrada en la toma de notas, gestión de proyectos y bases de datos. Se centra en ofrecer a los usuarios páginas muy personalizadas donde poder combinar textos, con tareas, calendarios, gráficos y una gran cantidad de personalización dentro de una única interfaz, no está estrictamente centrada en la compartición de archivos, pero si es una muy buena plataforma para la colaboración en tiempo real, ya que facilita en gran medida el trabajo conjunto sobre una única interfaz [16].

7. Conclusiones y Líneas de trabajo futuras

Personalmente el desarrollo de este Trabajo de Fin de Grado ha sido cohibido por el tiempo y agentes externos que han reducido el objetivo final, inicialmente planteado. Aun así, se ha realizado la finalmente el desarrollo inicial de una plataforma colaborativa, que puede llegar a facilitar la gestión de archivos y tareas entre usuarios, aprovechando sobre todo tecnologías escalables y accesibles. En conclusión, se han alcanzado los siguientes objetivos:

- **Desarrollo una aplicación web funcional**: Mediante el uso de las tecnologías como son Flask, HTML, SQLite entre otras, se ha conseguido desarrollar una aplicación web funcional que permite el registro, acceso, gestión de archivos, tareas y gestión de grupos
- **Arquitectura escalable y modular:** Se diseñó una aplicación escalable la cual permite fácilmente el mantenimiento y el desarrollo de líneas futuras, gracias a una buena estructura será posible la implementación de nuevas funcionalidades.
- **Autenticación y autorización segura:** Se logró una gestión sencilla y robusta de los usuarios, gracias a bibliotecas del framework Flask, como es Flask-login que garantizar recursos seguros, y permitiendo únicamente el acceso de usuarios autenticados.
- **Sencillez y eficacia**: El uso de herramientas modernas y adaptables como son el uso de ORM, junto a SQLite proporciona soluciones sencillas y eficientes para el desarrollo de un trabajo de estas características.

En conclusión, este Trabajo de Fin de Grado sienta las bases para una plataforma colaborativa más completa y efectiva, ya que se demuestra la viabilidad técnica del proyecto. Las posibles líneas futuras pueden brindar mejoras que aumenten la utilidad de la plataforma.

Líneas Futuras

Tras los logros alcanzados en este Trabajo de Fin de Grado, se puede plantear como únicamente el comienzo de una plataforma colaborativa más avanzada con una experiencia más completa y agradable para los usuarios.

Entre otras, algunas de las posibles líneas futuras del trabajo son:

- **Edición de archivos en Línea:** Incorporar la funcionalidad de editar archivos en línea, la cual pueda permitir a los usuarios trabajar directamente en la plataforma y colaborar rápidamente con sus compañeros, haciendo la plataforma un punto más central para los trabajos colaborativos.
- **Gestión de tareas más completa:** Implementar una mejor gestión de las tareas es una de las líneas de futuro más importantes, ya que actualmente se dispone de ellas como si de un tablón virtual se tratase. Buscando una implementación más al estilo Trello sería posible gestionar las tareas según columnas o “estados” que permitiesen a todo el equipo entender mejor que en momento se encuentra las tareas, además de poder asignar estas directamente a alguno integrantes del grupo.
- **Chat de comunicación en tiempo real:** Desarrollar un sistema de mensajes en tiempo real, esto puede fomentar aún más el centrar esta plataforma como lugar principal de trabajo colaborativo.
- **Asignación y notificaciones:** Permitir la capacidad de notificar a los usuarios mediante la asignación de archivos o tareas permitiría que fuera posible mantener aún más la utilidad de la plataforma ya que el usuario recibiría una notificación en cuanto alguno de sus archivos haya sido modificado, o tenga alguna nueva tarea asignada, etc.
- **Acceso móvil:** Mejorar la aplicación en su versión móvil ya que al disponer de tecnologías como Bootstrap es posible la continuación de un proyecto muy responsivo en todas las plataformas, haciendo que para los integrantes de grupos sea posible estar siempre al tanto de las actualizaciones que ocurren en estos.

Bibliografía

[1] MDN Web Docs. "HTTP Overview". Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Overview

[2] Red Hat. "¿Qué es una API REST?". Disponible en: https://www.redhat.com/es/topics/api/what-is-a-rest-api

[3] Webempresa. "¿Qué es un framework y para qué sirve?". Disponible en: https://www.webempresa.com/blog/que-es-un-framework-y-para-que-sirve.html

[4] Dev.to. "Object-Relational Mapping (ORM): Explicación completa, definición, uso, estructura y práctica". Disponible en: https://dev.to/ingeniela/object-relational-mapping-orm-explicacion-completa-definicion-uso-estructura-y-practica-4j56

[5] MDN Web Docs. "MVC". Disponible en: https://developer.mozilla.org/es/docs/Glossary/MVC

[6] Wikipedia. "Modelo–vista–controlador". Disponible en: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

[7] Flask. "Documentación de Flask". Disponible en: https://flask-es.readthedocs.io/

[8] J2Logo. "Tutorial Flask: Lección 5 – Base de datos con Flask-SQLAlchemy". Disponible en: https://j2logo.com/tutorial-flask-leccion-5-base-de-datos-con-flask-sqlalchemy/

[9] Microsoft. "Visual Studio Code Documentation". Disponible en: https://code.visualstudio.com/docs

[10] Git. "Documentación de Git". Disponible en: https://git-scm.com/doc

[11] Proyectos Ágiles. "Desarrollo iterativo incremental". Disponible en: https://proyectosagiles.org/desarrollo-iterativo-incremental/

[12] Wikipedia. "Third normal form". Disponible en:

https://en.wikipedia.org/wiki/Third_normal_form

[13]Microsoft Teams. "Microsoft Teams Collaboration Platform." Disponible en: https://www.microsoft.com/en-us/microsoft-teams/group-chat-software

[14]Trello. "Trello Project Management Tool." Disponible en: https://trello.com/

[15]Slack. "Slack Collaboration Hub." Disponible en: https://slack.com/

[16]Notion. "Notion All-in-One Workspace." Disponible en: https://www.notion.so/