GESTOR DOCUMENTAL WEB EN BLAZOR

WEB-BASED DOCUMENT MANAGEMENT SYSTEM IN BLAZOR

TRABAJO FIN DE GRADO
CURSO 2024-2025

AUTORES
TANIA RODRÍGUEZ BRAVO Y VÍCTOR CAPARRÓS CRUZ

NOTAS
TANIA RODRÍGUEZ BRAVO : 9
VÍCTOR CAPARRÓS CRUZ : 8

DIRECTOR
MARTA LÓPEZ FERNÁNDEZ

TRABAJO DE FIN DE GRADO DEL
DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

GESTOR DOCUMENTAL WEB EN BLAZOR

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

AUTORES
TANIA RODRÍGUEZ BRAVO
VÍCTOR CAPARRÓS CRUZ

DIRECTOR
MARTA LÓPEZ FERNÁNDEZ

CONVOCATORIA: JUNIO 2025

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

DEDICATORIA

Dedicatoria a nuestras familias que nos han aguantado durante toda la carrera de manera estoica.

AGRADECIMIENTOS

Agradecimiento especial a nuestras familias y amigos por aguantarnos todo el año además de la tutora por responder nuestras dudas de manera rápida y efectiva y a nosotros mismos por la realización de este trabajo.

RESUMEN

Gestor Documental Web en Blazor

El trabajo consiste en la realización de una aplicación web para la gestión de documentos compartidos entre el profesor y el alumno. Dicha página web está basada en Blazor combinando así HTML y CSS para la realización del diseño, además de C# para la codificación de las distintas funcionalidades. También nos hemos apoyado en SQL Server para alojar nuestra base de datos.

El objetivo del trabajo es brindar al profesor la capacidad de compartir documentos con los alumnos de manera sencilla e intuitiva, brindando el control y las estadísticas pertinentes de acceso y subida de diferentes tipos de documentos.

Palabras clave

Gestor Documental, Documentos, Subir, Carpetas, Estadísticas, Etiquetas, Base de datos, Notificaciones, Curso, Grupo

ABSTRACT

Gestor Documental Web en Blazor

The work consists in the realization of a web page for the management of documents between the teacher and the student. This website is based on Blazor combining HTML and CSS for the design, as well as C# for the coding of the different functionalities. We have also relied on SQL Server to host our database.

The objective of the work is to provide the professor the ability to share documents with students in a simple and intuitive way, providing control and relevant statistics of access and uploading diverse types of documents.

Keywords

Document Management System, Documents, Upload, Folders, Statistics, Tags, Database, Notifications, Course, Group.

ÍNDICE DE CONTENIDOS

Contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

En el mundo de la docencia siempre se ha requerido de recursos para ser usados en las aulas y compartirlos entre el docente y el estudiantado. Tanto si se trata de material teórico como práctico, es conveniente que estos recursos puedan compartirse de forma fácil y accesible. Además, es esencial que puedan localizarse cómodamente, usando filtros y criterios de búsqueda similares a los que ofrecen multitud de aplicaciones actuales.

Gracias a la evolución de la tecnología, en particular de las herramientas web, ha habido un notable aumento en el uso de recursos accesibles desde cualquier lugar y en cualquier momento. Esto abre la puerta a crear soluciones que integren estas posibilidades para mejorar la experiencia educativa. Así que, ¿qué mejor que desarrollar una aplicación web que permita subir fácilmente documentación y que los receptores (en este caso, los alumnos) puedan consultarla de forma ágil, sencilla y en cualquier momento?

En este documento se describe el desarrollo de una aplicación web que permite gestionar documentación compartida entre el profesorado y el alumnado. La aplicación ha sido diseñada con el objetivo de facilitar tanto la subida y organización de materiales por parte del docente como el acceso y filtrado por parte de los estudiantes, optimizando así la comunicación y el flujo de recursos educativos.

El desarrollo de este software ha comenzado con el estudio de las aplicaciones similares existentes, con el fin de identificar la funcionalidad concreta a implementar, tal como se describe en el capítulo 2. A continuación, el capítulo 3 presenta la definición de las tecnologías utilizadas en el desarrollo de la aplicación, detallando las decisiones técnicas adoptadas. El capítulo 4 se centra en el proceso de desarrollo, explicando las etapas y metodologías empleadas, mientras que el capítulo 5 expone los resultados obtenidos. Finalmente, el capítulo 6 recoge las conclusiones generales y las posibles líneas de mejora y evolución futura del proyecto.

1.1 Motivación

En entornos universitarios la gestión de documentos y diferentes archivos tanto por docentes como por estudiantes es un punto clave para facilitar y mejorar el aprendizaje.

Por un lado, como estudiantes valoramos y agradecemos tener a nuestra disponibilidad recursos que nos puedan servir de guía ya sea tanto a la hora de estudiar como a la hora de realizar trabajos de una asignatura o un tema en concreto. Además, es importante que el acceso a esos archivos sea lo más rápido y sencillo posible. Por eso un gestor documental donde la búsqueda y organización de los documentos se muestre de forma sencilla, intuitiva y rápida es necesario hoy en día.

Por otro lado, los profesores pueden beneficiarse de un sistema de gestión de documentos para ir almacenando y mostrando en cursos posteriores trabajos hechos por otros estudiantes o diversos documentos de interés que vayan desarrollando. Es importante asegurarse de no perder contenido valioso y fomentar el aprendizaje colaborativo.

1.2 Objetivos

El principal objetivo es el análisis, diseño y desarrollo de un Gestor Documental para el ámbito universitario.

Tras la investigación descrita en el capítulo 2, llegamos a la conclusión de que la aplicación debe ser capaz de gestionar el acceso y compartición de documentos, la organización por carpetas, la búsqueda eficiente de archivos y la asignación de roles de usuario diferenciados entre profesores y alumnos.

Además, contará con un panel de control personalizado para cada tipo de usuario, notificaciones automáticas, retención documental y estadísticas de acceso.

Otro objetivo fundamental es que la aplicación tenga una interfaz sencilla y cómoda para los usuarios, facilitando así su uso en las universidades. Los profesores podrán subir documentos para que los alumnos de diferentes cursos puedan acceder a ellos como material de referencia y apoyo académico.

El proyecto se dividirá en los siguientes subobjetivos:

1. Creación de la base de datos: que sirva de soporte a toda la información necesaria para hacer funcionar correctamente el sistema. Se incluirán las estructuras para gestionar documentos, usuarios, roles, estadísticas de acceso, organización por carpetas, etc.
2. Diseño de la aplicación: diseño tanto el backend como el frontend. Se elaborará un diseño de interfaz que permita navegar y acceder a la información con la mayor facilidad posible. Asegurando incluir todas las funcionalidades mencionadas previamente resultado de la investigación.
3. Realización de pruebas para verificar su correcto funcionamiento: se llevarán a cabo pruebas unitarias para verificar que la lógica de negocio no tiene fallos y cumple su función, así como pruebas para verificar que la integración con la base de datos es correcta. Además, se realizarán pruebas con usuario.

1.3 Plan de trabajo

Para poder desarrollar el proyecto dentro de los márgenes de tiempo establecidos, se propuso un plan de trabajo inicial dividido en fases generales, adaptado a la metodología Scrum mediante la realización de sprints sucesivos. A continuación, se describen las fases principales:

1. Investigación y análisis previo

- **Fecha inicio:** 24 de noviembre
- **Fecha fin:** 8 de diciembre
- **Objetivos:**
    - Realizar una búsqueda y análisis de aplicaciones similares en el ámbito de la gestión documental educativa.
    - Analizar y decidir las herramientas, tecnologías y lenguajes que se usarían a lo largo del proyecto.

- Diseñar las primeras pantallas e interfaces de usuario para establecer la base visual del proyecto.

2. Diseño y desarrollo

- **Fecha inicio:** 8 de diciembre
- **Fecha fin:** 11 de mayo
- **Objetivos generales:**
  - Desarrollar de manera incremental la aplicación web mediante diez sprints, abordando tareas específicas en cada iteración.
  - Las tareas principales incluyeron: creación de la base de datos, navegación de cursos, gestión y organización de documentos, implementación de funcionalidades como login y registro de usuarios, desarrollo de filtros y búsquedas, gestión de grupos, y realización de pruebas completas.
  - Al finalizar cada sprint, se revisaban los avances para ajustar y planificar las siguientes tareas, garantizando así una evolución continua del producto.

3. Documentación del proyecto

- **Fecha inicio:** durante todo el proyecto
- **Fecha fin:** 13 de mayo
- **Objetivos:**
  - Recoger apuntes y registros detallados de cada paso realizado, así como decisiones técnicas y funcionales adoptadas.
  - Redactar la memoria técnica final, incluyendo todas las fases del proyecto, tecnologías empleadas, procesos de desarrollo resultados obtenidos y posibles mejoras futuras.

Chapter 1- Introduction

In the field of education, the need for resources to be used and shared in the classroom between teachers and students has always been essential. Whether the material is theoretical or practical, it is important that these resources can be shared easily and accessibly. Additionally, it is crucial that they can be conveniently located, using filters and search criteria similar to those offered by many modern applications.

Thanks to the advancement of technology, especially web tools, there has been a notable increase in the use of resources accessible from anywhere at any time. This opens the door to creating solutions that leverage these possibilities to enhance the educational experience. So, what could be better than developing a web application that allows for the easy uploading of documentation, enabling students to consult it in a simple, quick, and accessible manner?

This document describes the development of a web application that facilitates the management of shared documentation between teachers and students. The application has been designed with the objective of simplifying both the uploading and organization of materials by teachers and the access and filtering of such resources by students, thereby optimizing communication and the flow of educational content.

The development of this software began with the study of similar existing applications in order to identify the specific functionalities to implement, as detailed in Chapter 2. Chapter 3 then outlines the technologies used in the development process, describing the technical decisions made. Chapter 4 focuses on the development process itself, explaining the stages and methodologies followed, while Chapter 5 presents the results obtained. Finally, Chapter 6 provides the general conclusions as well as potential improvements and future development lines for the project.

1.1 (BIS) Introduction

In university environments, managing documents and files by both teachers and students is a key aspect to facilitating and improving the learning process.

From the students' perspective, we value and appreciate having access to resources that can guide us both when studying and when working on assignments or specific topics. Furthermore, it is important that access to these files is as fast and straightforward as possible. For this reason, a document manager where the search and organization of files is intuitive, simple, and quick is necessary nowadays.

On the other hand, teachers can benefit from a document management system to store and showcase works from previous students or various useful documents that they develop over time. It is important to ensure that valuable content is not lost and to promote collaborative learning.

1.2 (BIS) Objectives

The main objective of this project is the analysis, design, and development of a Document Management System tailored to the university context.

Following the research described in Chapter 2, it was concluded that the application should be capable of managing document access and sharing, folder organization, efficient file searching, and user role differentiation between teachers and students.

Additionally, the system will feature a personalized control panel for each user type, automatic notifications, document retention, and access statistics.

Another key objective is to ensure that the application has a simple and user-friendly interface, thereby facilitating its use in universities. Teachers will be able to upload documents for students in various courses to access as academic support material.

The project is divided into the following sub-objectives:

- **Database creation**: To support all the information needed for the system to function correctly. This includes structures for managing documents, users, roles, access statistics, folder organization, etc.
- **Application design**: Involves both backend and frontend development. A user interface will be designed to ensure easy navigation and access to information, including all previously identified functionalities from the research phase.
- **Testing to ensure proper functionality**: Unit tests will be carried out to verify the correctness of the business logic, as well as integration tests to validate the connection with the database. Additionally, user testing will be conducted.

1.3 (BIS) Work plan

To develop the project within the established time frame, an initial work plan was proposed, structured into general phases and adapted to the Scrum methodology through successive sprints. The main phases are as follows:

1. **Research and Preliminary Analysis**

   Start date: November 24

   End date: December 8

   ***Objectives*:**

   - Research and analyze similar applications in the field of educational document management.
   - Decide on the tools, technologies, and programming languages to be used throughout the project.
   - Design the initial user interfaces to establish the visual foundation of the application.

2. **Design and Development**

   ***Start date*:** December 8

   ***End date*:** May 11

   ***General objectives*:**

- Incrementally develop the web application through ten sprints, tackling specific tasks in each iteration.
- Main tasks included: database creation, course navigation, document management and organization, implementation of login and user registration functionalities, development of filters and searches, group management, and comprehensive testing.
- After each sprint, progress was reviewed to adjust and plan upcoming tasks, ensuring continuous evolution of the product.

3. Project Documentation

***Start date*:** Throughout the entire project

***End date*:** May 13

***Objectives*:**

- Maintain detailed notes and records of each step taken, as well as the technical and functional decisions made.
- Write the final technical report, including all project phases, technologies used, development processes, results obtained, and possible future improvements.

Capítulo - 2 Estado de la cuestión

En este capítulo se muestra los resultados obtenidos tras un estudio de programas o aplicaciones softwares existentes en el mercado actual que abordan la gestión documental. El objetivo es recopilar las diferentes funcionalidades que se implementan estas aplicaciones software ya existentes y analizar cuáles son fundamentales o básicas para realizar una gestión documental. Además, este estudio ha permitido descubrir diferentes perspectivas que han ampliado nuestra idea inicial, así como necesidades que aún no estén cubiertas para realizar nuestro proyecto: un gestor documental web.

Como se explica a continuación la investigación se divide en dos fases: en la primera nos centramos en buscar aplicaciones software que están relacionadas con el ámbito universitario o de clase en general; y, concretamente que estén centradas en la compartición de archivos. Tras analizar los resultados obtenidos se planteó una nueva investigación, considerando necesidades más específicas para nuestro proyecto.

2.1 Primera investigación

El estudio se inició con la selección de un conjunto de palabras clave que han permitido recopilar un conjunto de aplicaciones actuales relacionadas con el ámbito del trabajo.

Las palabras claves seleccionadas son: "Gestor Documental", "Homework Assignment Management System", "Classroom assignment management tools" o "Classroom Management Tools".

En esta primera fase no se ha realizado ningún otro filtro tal como el tipo de software pudiendo ser aplicación web, móvil o basadas en la nube. En la siguiente sección se presentan de forma breve las cinco aplicaciones encontradas y analizadas. Sobre cada una investigamos cómo funciona el software y sus principales funcionalidades.

De esta primera fase y basándonos en las aplicaciones estudiadas se obtiene un conjunto de funcionalidades que suelen ser comunes en todas ellas y se ha creado una tabla-resumen presentada en la sección 2.1.2.

2.1.1 Aplicaciones analizadas

A. ownCloud

ownCloud es una plataforma de almacenamiento en nube que permite acceder, compartir y editar documentos de forma segura y desde cualquier dispositivo, ofreciendo control total sobre tu información [1][2].

- Garantiza acceso mediante registro (se puede dar permisos como invitado).
- Colaboración y el uso compartido de archivos en la nube.
- Tanto estudiantes como profesores pueden subir archivos.
- Se pueden crear grupos personalizados de trabajo (spaces).
- Se pueden organizar los archivos por carpetas.
- Se pueden buscar archivos por nombre tanto dentro de una carpeta como en general.
- Almacenamiento y búsqueda de metadatos de los archivos.
- Se puede integrar con Collabora Online, ONLYOFFICE y Microsoft Office
- Control de versiones de archivos
- Control de permisos.

B. Docuware

Docuware es un software basado en la nube y aplicación web/móvil de Gestión Documental y Automatización de Flujos de Trabajo. Permite organizar y automatizar procesos de documentos, aportando acceso seguro y eficiente desde todos los lugares [3][4].

- Almacena documentos en archivadores electrónicos de cualquier fuente de forma centralizada e incluyendo metadatos o información relacionada.
- Búsqueda flexible, visión completa: búsqueda por texto completo, filtros específicos o vista de archivos.
- Editar documentos en el programa original, control de versiones.
- Se puede integrar con Microsoft Teams, Outlook, ERP o CRM.
- Disponible local, nube o ambas.
- Control de permisos.

C. Moodle

Moodle es un software gratuito de aprendizaje en línea (LMS, Learning Management System). Permite a los miembros de la comunidad educativa crear un sitio en línea privado para registrar un curso en cualquier momento [5]. Ofrece versión web y también una aplicación móvil.

- Ofrece páginas para el curso personalizadas.
- Desde el dashboard o escritorio se pueden visualizar los eventos próximos a lo largo del tiempo.
- Contiene herramientas colaborativas como wikis y glosarios.
- Permite subir documentos mediante drag & drop o desde OneDrive, Dropbox o Google Drive.
- Tiene un editor de texto simple e intuitivo.
- Los alumnos pueden recibir notificaciones sobre actividades.
- Los alumnos y profesores pueden seguir el progreso y la completitud de actividades.

D. Additio App

Es una aplicación web y móvil para la gestión educativa. Aporta una solución todo en uno para escuelas que permite unificar la administración y la comunicación entre alumnos y profesores [6].

- Toda la información de la escuela se guarda en un mismo sitio.
- Toda la parte administrativa se gestiona en una sola herramienta, planificador de tareas y calificaciones.
- Informes configurables con toda la información.
- Conecta a profesores, familias y estudiantes en una sola plataforma, mediante grupos para enviar mensajes, email y envío de documentos y circulares.
- Crea cualquier tipo de formulario y autorización personalizada con documentos anexos.
- Crea horarios personalizados en minutos.

E. EduCloud

Es una plataforma en la nube y aplicación web orientada a la gestión educativa. Permite la asignación y organización de deberes de manera diaria y otras funcionalidades relacionadas con las tareas docentes [7].

- Dashboard para asignaciones de tareas de manera sencilla mediante drag&drop.
- Creación de tareas con notificaciones automáticas a los estudiantes.
- Permite crear formularios en Google Drive para que los rellenen los estudiantes con notificación automática.
- Permite crear documentos de Google que serán asignados a cada estudiante.
- Los estudiantes pueden subir, documentos, enlaces de web, links de video y fotos a través de la web. Además, los profesores pueden dar feedback.

2.1.2 Funcionalidades implementadas

A continuación, se muestra en la Tabla 1la comparativa de todas las funcionalidades encontradas en las aplicaciones y softwares analizados.

Tras esta primera investigación se observa que tanto Moodle como Additio App y EduCloud no se ajustan a lo que se busca pues están más orientados a la gestión docente y de tareas de clase. Solo ownCloud y Docuware cumplen con tareas básicas de gestores de documentos como son la organización y búsqueda documental.

Por tanto, Moodle, Additio App y EduCloud se descartan y se realiza una nueva investigación, con el objetivo de encontrar herramientas más similares a nuestros requerimientos y completar el análisis ya realizado a ownCloud y Docuware.

	ownCloud	Docuware	Moodle	Additio App	EduCloud
Acceso, edición y compartición de documentos	Sí	Sí	Sí	Sí	Sí
Usuarios (registro)	Sí	Sí	Sí	Sí	Sí
Control de versiones de documentos	Sí	Sí	No	No	No
Búsqueda de documentos	Sí	Sí	No	No	No
Almacenamiento centralizado	Sí	Sí	No	Sí	No
Integración con otras plataformas	Sí	Sí	Sí	Sí	Sí
Control de permisos	Sí	Sí	Sí	Sí	No
Creación de grupos	Sí	Sí	Sí	Sí	No
Tareas asignadas y seguimiento	No	No	Sí	Sí	Sí
Notificaciones automáticas	No	No	Sí	Sí	Sí
Editor de texto	No	No	Sí	Sí	Sí (Google Docs)
Panel de control personalizado	No	No	Sí	Sí (horarios personalizados)	Sí (drag & drop de tareas)
Gestión de archivos (carpetas)	Sí	Sí	No	No	No

Tabla 1. Tabla comparativa I

2.2 Segunda investigación

Esta segunda fase comienza nuevamente escogiendo una serie de palabras clave que nos permitan esta vez ajustar la búsqueda a herramientas orientadas principalmente a la gestión de documentos dejando el ámbito docente atrás.

Las palabras claves utilizadas en este segundo estudio son "Gestión Documental", "Gestor de documentos" y "Document Manager". Se mantiene como en la primera investigación el no filtrado por tipo de aplicación o software.

A continuación, se presentan tres nuevas tecnologías investigadas y un breve resumen de éstas. No se duplica la descripción de ownCloud y Docuware, disponible en la sección 2.1.1.

2.2.1 Nuevas aplicaciones analizadas

F. Zoho Workdrive

Zoho Workdrive es una aplicación online (nube) para la gestión de archivos para equipos que trabajan juntos. Permite almacenar, compartir y colaborar en documentos de forma segura y la organización de los archivos [8].

- Organización de los documentos en carpetas que se pueden crear a su vez para cada grupo.
- Búsqueda de los documentos por nombre, palabras clave, ubicación o tipo de archivo.
- Control de versiones para poder ver los cambios.
- Roles para los miembros de cada equipo
    - Viewer: solo puede ver los documentos y las carpetas.
    - Comentador: todo lo que puede hacer el viewer incluyendo comentar.
    - Editor: todo lo que puede hacer el comentador incluyendo crear o cargar archivos, modificar y subir archivos, acceder a las estadísticas de acceso a los archivos y cambiar el nombre de los archivos/carpeta.

    - Organizador: todo lo que puede hacer el editor incluyendo mover archivos/carpetas entre carpetas del mismo equipo, compartir, restaurar, añadir o eliminar carpetas de equipo y manejar los permisos de los miembros del equipo.
    - Administrador: todo lo que el organizador puede hacer incluyendo eliminar carpetas del equipo, configurar la configuración de las carpetas de equipo (público/privado, agrega o quita miembros, mover documentos/carpetas a otras carpetas que no sean del equipo, eliminar documentos y carpetas de la papelera del equipo.

- Interfaz: se pueden fijar carpetas, elegir las carpetas a mostrar en el panel, se pueden replicar carpetas de equipo para trabajar con diferentes equipos a la vez.
- Se integra con otras aplicaciones de Zoho, como Zoho CRM, Zoho Projects y Zoho Mail.
- Proporciona herramientas para crear, formatear y editar documentos de forma colaborativa en tiempo real en Zoho Writer; trabajar con funciones, tablas dinámicas y gráficos en Zoho Sheet (alternativa a Excel); y realizar presentaciones en Zoho Show.

G. R2 Docuo

R2 Docuo es un gestor documental con inteligencia artificial basado en la nube. Permite almacenar, organizar y gestionar flujos de trabajo y de documentos además de manejar versiones de forma segura [9].

- Organización de los documentos por carpetas.
- Categorías de documentos.
- Taxonomía y metadatos: se puede ver información clave sin abrir el archivo.
- Gestión de flujos de trabajo.
- Colaboración: crear comentarios, recordatorios y tareas para el equipo en cualquier documento.

- Búsqueda de documentos: por palabras en el título o metadatos.

- Integración con otras aplicaciones: Excel, Teams, Outlook, etc.
- Control de versiones y de cambios.
- Base de datos relacional e integridad referencial: utiliza listas desplegables para conectar los documentos y no introducir el mismo dato dos veces.
- Retención documental: se puede poner fecha de caducidad a los documentos para tenerlos guardados solo un periodo de tiempo.
- Seguridad y permisos: permite especificar qué puede hacer (modificar, crear, eliminar o ver) cada usuario en cada documento y en qué carpetas.
- Intercambio de archivos

H. Open KM

Open KM es una aplicación web de gestión documental en la nube. Permite almacenar, organizar, compartir y buscar documentos además de controlar las versiones [10].

- Catalogación automática: tareas automáticas en base a eventos. Pueden establecerse reglas basadas en eventos; por ejemplo, mover un documento a una nueva ubicación, modificar la seguridad, aplicar transformaciones a los documentos (convertir a PDF con compresión grupo 4) entre otros. [10]
- Captura automática de metadatos.
- Previsualización avanzada: extensiones como AutoCAD y DICOM.
- Control de versiones.
- Auditoría.
- OCR: permite convertir diferentes tipos de documentos en datos editables sobre los que realizar consultas.

2.2.2 Funcionalidades implementadas

A continuación, se muestra la tabla comparativa de todas las funcionalidades encontradas en las aplicaciones y software investigados en esta segunda fase.

	ownCloud	**Docuware**	**Zoho Workdrive**	**R2 Docuo**	**Open KM**
Acceso y Compartición	Sí	Sí	Sí	Sí	Sí
Control de Permisos	Sí	Sí	Sí	Sí	Sí
Roles de Usuario	Si	Si	Sí	No	No
Creación de grupos	Sí	Sí	Sí	No	No
Colaboración en Documentos	Sí	Sí	Sí	Sí	Sí
Organización por Carpetas	Sí	Sí	Sí	Sí	Sí
Búsqueda por Nombre	Sí	Sí	Sí	Sí	Sí
Búsqueda por Metadatos	Sí	Sí	No	Sí	Sí

Búsqueda por Texto Completo	No	Sí	No	No	No
Control de Versiones	Sí	Sí	Sí	Sí	Sí
Integración con Microsoft Office	Sí	Sí	Sí	Sí	No
Integración con otras Aplicaciones	Sí	Sí	Sí	Sí	Sí
Notificaciones automáticas	No	No	Sí	Sí	Sí

Tabla 2. Tabla comparativa II

2.3 Resultados obtenidos

Como resultado de este estudio se han obtenido dos tablas en las que se recopilan las principales funcionalidades implementadas en aplicaciones software actualmente disponibles en el mercado.

Estos estudios han facilitado la selección de aquellas funciones que se implementarán en el software a desarrollar en este trabajo. Pero también ha permitido descartar procesos que aparecen en algunas aplicaciones pero que no son propios de un gestor documental.

Por ejemplo, todas las funcionalidades de edición de documentos, creación de grupos o seguimiento de tareas. También se han descartado otras funcionalidades avanzadas como la previsualización avanzada de documentos y el OCR ya que se considera que no aportan valor a este trabajo.

El conjunto de funcionalidades básicas del gestor documental que si vamos a implementar son:

- Acceso y compartición de documentos.
- Organización de estos en carpetas.
- Registro de usuarios y gestión de roles.
- Control de permisos asociados a los diferentes roles.
- Desarrollo de un método de búsqueda de documentos por nombre y etiquetas.
- Filtros por etiquetas pata facilitar la búsqueda.

Además de este conjunto básico de funcionalidades, se ha considerado interesante añadir otras que no están presente en todas las aplicaciones estudiadas y que indicamos a continuación.

- Estadísticas de acceso

  Por un lado, tenemos las estadísticas de acceso que solo encontramos en Zoho Workdrive y, que encontramos útil en nuestro proyecto para ver qué documentos son los más visitados en este caso por los estudiantes y cuáles no. Sería positivo para nuestro proyecto ya que otorgaría al profesor cuales son los documentos subidos que los estudiantes consultan con más frecuencia y por tanto se consideraría que es más útil para ellos. Esto ayudaría al docente a decidir qué tipo de documento subir en un futuro y cuáles no.
- Notificaciones automáticas

  Las notificaciones automáticas sí que las implementan la mayoría de los softwares analizados, pero no todos. Pero, se cree que aportan una clara ventaja a raíz de avisar a los usuarios de que se ha subido un nuevo documento y está disponible para su visualización. Se considera que mejora la comunicación entre docente y estudiantes y que ayudaría a los alumnos a no perderse ningún material que le pueda resultar interesante. Además, evitaría que el usuario en este caso el estudiante, tuviera que hacer consultas manuales para ver si hay nuevas subidas, mejorando la productividad y reduciendo tiempo perdido.

- Retención documental

  Por último, la retención documental, solo presente en una de las aplicaciones que creemos que aportará valor al proyecto al poder establecer fechas para las cuales el documento estará visible. Se podría implementar para documentos que estén disponibles en el año académico y que no sean necesarios para el siguiente curso.

Por tanto, como resultado final de este estado de la cuestión se ha obtenido el conjunto de procesos que se implementarán en este trabajo. Incluye la funcionalidad básica para realizar una gestión documental en un entorno universitario y se complementa con procesos que aportan un valor añadido para facilitar al usuario el uso de la herramienta y el control de documentos de una asignatura a lo largo del tiempo, en diversos cursos académicos.

Capítulo - 3 Tecnologías utilizadas

En este capítulo se listan y se explican brevemente las tecnologías que se han utilizado para el desarrollo de este Trabajo de Fin de Grado. Se han agrupado según se refieran a la arquitectura del software, proceso de desarrollo, implementación del software y gestor de base de datos seleccionado.

3.1 Arquitectura

3.1.1 Single Page Application

Una Single Page Application (SPA) es un tipo de aplicación web que se carga una única vez en el navegador y actualiza dinámicamente su contenido sin necesidad de recargar la página completa. Esto se consigue mediante el uso de tecnologías como JavaScript y frameworks modernos (por ejemplo, Blazor, React, Angular o Vue), que permiten manejar la navegación y la interacción del usuario directamente en el cliente, solicitando solo los datos necesarios al servidor. Para este trabajo se ha seleccionado el framework Blazor, descrito en la sección 3.3.1.

El principal objetivo de una SPA es mejorar la experiencia del usuario al proporcionar una navegación más fluida, rápida y similar a la de una aplicación de escritorio.

3.1.2 Arquitectura 3 capas

La arquitectura en capas es un modelo de diseño arquitectónico de software que se utiliza para organizar aplicaciones de manera modular, separando responsabilidades para mejorar la mantenibilidad, escalabilidad y reutilización del código.

Su base es la separación de las diferentes funcionalidades del sistema en capas o niveles, donde cada capa se encarga de un conjunto de tareas específicas y se comunica con los niveles adyacentes mediante interfaces bien definidas [16]. Estas

interfaces son un conjunto de definiciones de métodos y propiedades que proporciona una forma estandarizada de comunicación entre dos capas adyacentes.

En el desarrollo de aplicaciones web es común utilizar 3 capas, con las siguientes responsabilidades:

- La capa de presentación: encargada de mostrar la información al usuario, es decir, el desarrollo de la interfaz y cómo se muestra.
- La capa de negocio: encargada de procesar la información, normalmente es la que contiene un mayor nivel de procesamiento de datos y es la encargada de conectar la capa de presentación con la capa de acceso a datos.
- La capa de acceso a datos: encargada de obtener y modificar los datos requeridos mediante la conexión directa con la base de datos.

Esta arquitectura facilita el mantenimiento del software y proporciona una mayor claridad a nuestro flujo de datos entre capas.

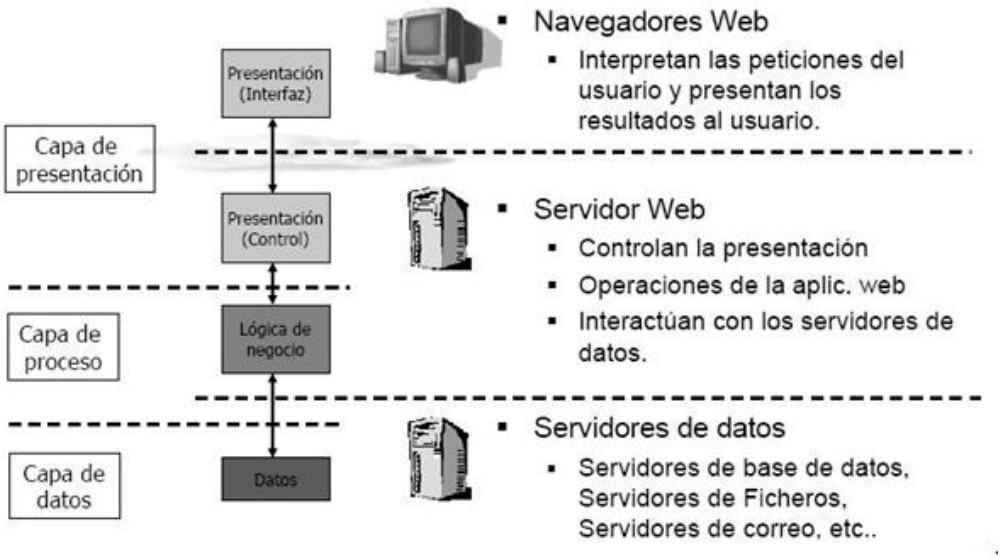

Figura 1. Arquitectura arquitectónica 3 capas

3.2 Proceso de desarrollo de software

3.2.1 Scrum

Scrum es un marco de trabajo ágil utilizado principalmente en el desarrollo de software, aunque aplicable a otros tipos de proyectos. Promueve buenas prácticas colaborativas en equipo para lograr los mejores resultados. Se basa en entregas parciales y regulares del producto, priorizadas según el valor que aportan al cliente, y es especialmente útil en entornos complejos con requisitos cambiantes, donde se requiere innovación, flexibilidad y productividad.

Los proyectos en Scrum se desarrollan en ciclos cortos y fijos, generalmente de dos semanas, llamados sprints. Cada sprint debe generar un incremento del producto que pueda ser entregado al cliente con mínimo esfuerzo. Al inicio se realiza el *Sprint Planning*, donde se definen las historias de usuario a trabajar y se estiman sus tiempos, seleccionando solo las que pueden completarse según la disponibilidad del equipo.

Durante el sprint, se llevan a cabo reuniones diarias (*dailys*) para identificar problemas y seguir el progreso. Al finalizar, se realizan dos revisiones: el *Sprint Review*, para verificar si se cumplieron los objetivos, y el *Sprint Retrospective*, donde se analizan mejoras para el próximo sprint y se discuten los obstáculos enfrentados.

Este enfoque permite dividir el trabajo en sprints, facilitando la organización y el seguimiento de los objetivos para alcanzar el resultado final: una aplicación web de gestión documental.

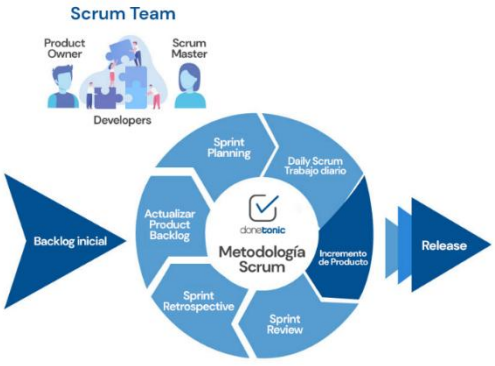

Figura 2. Metodología scrum

3.2.2 Jira

Jira es una herramienta de software diseñada para la gestión de proyectos y el seguimiento de incidencias. Es una plataforma que facilita la colaboración y el control del progreso de proyectos mediante un sistema de incidencias, esto proporciona una visión general del estado del proyecto [12].

Algunas de las características más importantes de Jira son: la gestión de tareas y proyectos, ya que permite la creación de estas; la asignación a miembros del equipo y establecer plazos de inicio y finalización; la personalización y la flexibilidad. Jira permite modificar casi todos los aspectos de la plataforma para que se adapte a los métodos de trabajo de cada equipo de desarrollo, proporcionando algunas vistas como la Scrum que es la aplicable a nuestro trabajo.

Por último, permite la integración con otras herramientas como Git para la gestión de los proyectos. En el caso concreto de Scrum, mediante Jira podemos llevar a cabo de manera eficiente la creación de sprints, la adición de historias de usuario y la creación de tareas de manera personalizada en base a lo que requiera el desarrollo de nuestro proyecto, la gestión de tareas en Jira se ejecutaría de la siguiente manera:

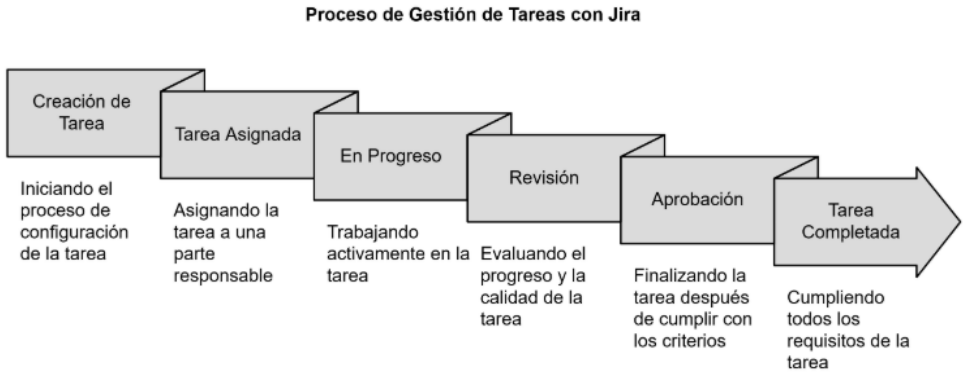

Figura 3. Proceso de gestión de tareas con Jira

Llevando a cabo la gestión mediante Jira, tenemos más claridad con respecto a los objetivos que queremos cumplir, como los vamos a realizar y cuál es el tiempo disponible para completar cada uno de ellos.

3.2.3 *GitHub*

GitHub es un servicio basado en la nube que aloja un sistema de control de versiones (CVS) llamado Git. Permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantienen un seguimiento detallado de su progreso [11].

Mediante Git cualquier desarrollador del equipo que tenga acceso puede gestionar el código fuente y el historial de cambios del mismo. A diferencia de otros sistemas, Git ofrece ramas, lo que hace que cada ingeniero de software pueda crear una rama que proporcionará un repositorio local para hacer cambio en el código ya que estos cambios no afectan a la rama maestra, lugar en la que se encuentra el código original del proyecto.

En el caso concreto de GitHub, es una plataforma de gestión y organización de proyectos que incorpora las funciones de control de versiones de Git. Además, la interfaz de usuario GitHub es más sencilla que la de Git lo cual permite que sea más accesible para personas con menores conocimientos técnico.

Hemos decidido escoger GitHub por el conocimiento previo que ya tenemos en dicha plataforma y por la sencillez de uso y gestión que nos proporciona la herramienta.

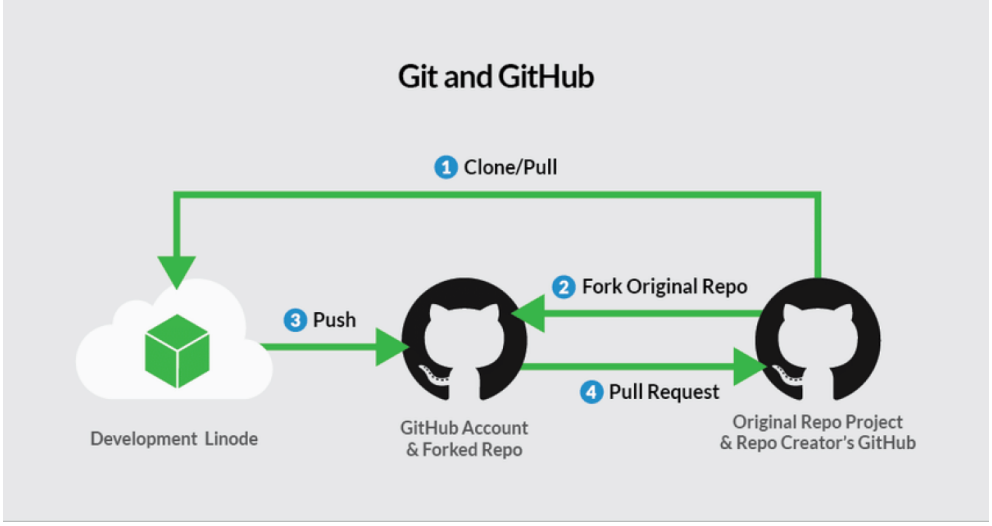

Figura 4. Git y GitHub

3.3 Desarrollo del software

3.3.1 Blazor

Blazor es un proyecto desarrollado por Microsoft creado para permitir crear SPAs únicamente usando como lenguajes de programación C# y Razor Pages, haciendo nula la necesidad de programar en Javascript o frameworks derivados. Esto permite reducir la complejidad de trabajar con Javascript utilizando únicamente C#, HTML y CSS. [14]

El uso de C# frente a JavaScript proporciona ciertas ventajas:

- Mediante la escritura de código en C# se puede mejorar la productividad en el desarrollo y el mantenimiento de aplicaciones.
- Aprovechamiento del ecosistema .NET ya existente, en base a las bibliotecas.
- Compilar sobre un conjunto común de lenguajes, marcos y herramientas que son estables, completos y fáciles de usar.

Las aplicaciones de Blazor están basadas en componentes. Un componente es un elemento de interfaz de usuario, como una página, un cuadro de diálogo o un formulario de entrada de datos.

La clase del componente normalmente se escribe en forma de página de marcado de Razor. Razor es una sintaxis para combinar HTML con código C# diseñada para aumentar la productividad del desarrollador.

En este proyecto se decidió utilizar Blazor frente a otros lenguajes o frameworks debido al conocimiento previo del equipo en C# y HTML, lo que facilita el desarrollo tanto en términos de velocidad como de reducción de complejidad. Además, tras una revisión exhaustiva de bibliotecas gratuitas disponibles para Blazor, se seleccionó Radzen Blazor Components, una biblioteca de componentes de interfaz de usuario de código abierto que incluye más de 90 elementos. Su enfoque profesional y su capacidad para facilitar el manejo de archivos fueron factores determinantes en la elección [20][21]. Entre sus ventajas destaca la incorporación de elementos como modales y menús interactivos, funcionales para los requerimientos del proyecto. Asimismo, ofrece una integración directa con Entity Framework y bases de datos SQL, resultando totalmente compatible con la arquitectura adoptada.

Característica	Blazor Server	Blazor WebAssembly
Ubicación de ejecución	En el servidor, usa SignalR para actualizar el cliente.	En el cliente; runtime .NET y la aplicación se descargan en el navegador.
Seguridad / Exposición de datos	Lógica y datos sensibles permanecen en el servidor, evitando exponerlos.	Parte de la lógica y el código se ejecuta en el cliente, aumentando la superficie de ataque.
Despliegue y actualizaciones	Cambios en el servidor se reflejan inmediatamente en todos los clientes.	Requiere redeploy de la aplicación cliente y recarga para ver cambios.
Carga inicial	Solo HTML y SignalR: mínima transferencia inicial.	Descarga completa del runtime .NET y ensamblados: mayor peso inicial.
Rendimiento en conexiones lentas	Mejor experiencia: poco tráfico y latencia reducida.	Peor experiencia si la descarga es lenta o la CPU del cliente es débil.
Uso de recursos del cliente	Muy bajo: casi toda la carga recae en el servidor.	Alto: la app ejecuta .NET en el navegador, consume memoria y CPU.
Acceso a base de datos	Directo desde el servidor, sin necesidad de APIs REST o gRPC.	Necesita exponer datos mediante APIs REST/gRPC

		para que el cliente los consuma.
Compatibilidad de navegadores	Compatible con cualquier navegador que soporte SignalR (WebSockets).	Requiere navegadores con soporte WebAssembly.
Integración con ASP.NET Core existente	Muy sencilla: la lógica permanece en el servidor, mínimo refactor.	Más compleja: hay que configurar APIs adicionales y mover lógica al cliente.

Tabla 3. Blazor server vs webassembly

Es importante entender el ciclo de vida de los componentes Blazor, para entender cómo se cargan y renderizan los diferentes elementos, y para comprender por qué se utiliza ese método en este caso concreto.

Blazor ejecuta una serie de eventos correspondientes al ciclo de vida de sus componentes, cuyas etapas principales son las siguientes [25][26]:

- Instanciación del componente: se crea una nueva instancia del componente.
- Inyección de dependencias: se inyectan los servicios y dependencias necesarias.
- Invocación de OnInitialized / OnInitializedAsync: se inicializa el componente. La versión síncrona es más eficiente y se utiliza cuando no es necesario recuperar datos externos, mientras que la versión asíncrona permite realizar operaciones como llamadas a APIs.
- Invocación de OnParametersSet / OnParametersSetAsync: se asignan los parámetros al componente y se gestionan las acciones en respuesta a su recepción. La variante asíncrona es adecuada cuando es necesario realizar consultas o actualizaciones basadas en dichos parámetros.
- Renderización del componente: se genera el marcado HTML y se actualiza la interfaz de usuario.

- Invocación de OnAfterRender / OnAfterRenderAsync: se ejecutan operaciones posteriores a la renderización que dependen del DOM. La variante asíncrona se utiliza cuando es necesario ejecutar código dependiente del DOM o de JavaScript.

3.3.2 BCrypt

Para la seguridad de los datos de usuario que se utiliza en el software se usa BCrypt, una función de hash específicamente diseñada para el almacenamiento seguro de contraseñas. BCrypt se basa en el algoritmo de cifrado simétrico Blowfish [37].

Blowfish es un algoritmo que permite claves de longitud variable de hasta 448 bits y opera sobre bloques de 64 bits. Su estructura se basa en una red de Feistel con 16 rondas de cifrado, lo cual incrementa su robustez frente a ataques criptográficos [35][36].

En particular, BCrypt presenta varias características que lo convierten en una solución segura y recomendable para el almacenamiento de contraseñas [34][38]:

- Uso de salt aleatorio de 16 bits: impide el uso efectivo de ataques basados en tablas rainbow.
- Parámetro de "work factor" (coste computacional): configurable para aumentar la dificultad de procesamiento según las necesidades de seguridad.
- Función de derivación de claves adaptativa: ajusta dinámicamente el esfuerzo computacional requerido en función del hardware disponible, dificultando ataques por fuerza bruta.
- Velocidad de procesamiento intencionadamente reducida: diseñada para mitigar ataques automatizados mediante fuerza bruta.

3.3.3 Visual Studio 2022

Visual Studio es un entorno de desarrollo integrado (IDE, en inglés) completo que se puede usar para escribir, editar, depurar y compilar código [13]. Por tanto, es una

herramienta de desarrollo eficaz que permite completar todo el ciclo de desarrollo en un único IDE.

Visual Studio proporciona muchas características que facilitan la escritura y la administración del código con confianza, además incluye soporte para GitHub. También incluye el explorador de soluciones, para organizar y explorar el código además de la vista de clases que nos permite visualizar el código organizado por clases.

Otras de las características fundamentales son su sencilla capacidad de compilación y depuración de código, permitiendo verificar errores y su posterior corrección, también permite la integración de pruebas para prevenir dichos errores.

Dicha aplicación ofrece soporte directo a aplicaciones web con Blazor lo cual nos ayuda a realizar nuestro proyecto sin la necesidad de instalación de elementos adicionales.

3.4 SQL Server

SQL Server es uno de los principales sistemas de gestión de bases de datos relacional del mercado que presta servicio a un amplio abanico de aplicaciones de software destinadas a la inteligencia empresarial y análisis sobre entornos corporativos. SQL Server es ideal para almacenar toda la información deseada en bases de datos relacionales, como también para administrar dichos datos sin complicaciones, gracias a su interfaz visual y a las opciones y herramientas que tiene [15].

Está compuesto por un motor relacional encargado del procesamiento de comandos, consultas, así como del almacenamiento de archivos, tablas y búferes de datos.

Las principales características de SQL Server son:

- Inteligencia en todos sus datos con clústeres de Big Data.
- Elección de lenguaje y plataforma.
- Capacidades de bases de datos inteligentes.
- Cifrado de datos y cumplimiento normativo.
- BI móvil y escalabilidad.

Se ha escogido SQL Server debido a su facilidad de gestión y conexión de las bases de datos con las diferentes aplicaciones y los conocimientos previos que tenemos sobre la plataforma.

Capítulo - 4 Desarrollo del software

En este capítulo se listan los diferentes sprints que se han llevado a cabo para realizar el proyecto, incluyendo una explicación detallada de cada una de las tareas específicas que se ha hecho para conseguir los diferentes objetivos según el sprint.

En la Tabla 4 se muestran los sprints realizados y las tareas correspondientes a cada uno de nosotros. Para mejorar la legibilidad de la tabla se usan solo nuestras iniciales en la última columna (Quién) para indicar la persona que ha desarrollado cada subtarea e historia: T (Tania Rodríguez Bravo) o V (Víctor Caparrós Cruz). Las historias de usuario se detallan en la sección 4.1.2. Para cada historia-subtarea se indica si está finalizada (Fin) o en desarrollo (Desarr).

Sprint	Historia	Subtarea	Estado	Quién
0	Pantallas y bocetos	Creación bocetos	Fin	T
1	HU014-Base de datos	Creación tablas y relaciones	Fin	V
		Definir roles	Fin	V
2	HU02-Creación y navegación de cursos	Diseño pantalla principal	Fin	T
		Estructura 3 capas	Fin	T
		Navegación cursos	Fin	T
3	HU03-Carga y representación de documentos HU04-Subida de documentos HU05-Visualización contenido documentos	Carga documentos	Fin	T
		Representación de documentos	Fin	T
		Subir nuevos documentos	Fin	T
		Visualización del contenido	Fin	T
4	HU06-Modificación características documentos	Opción ocultar/mostrar	Fin	T
		Opción estadísticas	Fin	T
		Configuración estadísticas acceso	Fin	V
		Opción fecha baja	Fin	T
		Opción etiquetas	Fin	T
		Opción renombrar	Fin	T
		Opción eliminar	Fin	T
5	HU07-Login y registro de usuarios	Diseño pantalla login	Fin	T
		Lógica login	Fin	V

		Diseño pantalla registro	Fin	T
		Lógica registro	Fin	V
		Menú de perfil	Fin	T
6	HU08-Organización documentos en carpetas	Subir/Crear carpeta	Fin	T
		Drag&Drop documentos a carpetas	Fin	T
		Menú contextual carpetas	Fin	T
7	HU09-Búsqueda de documentos	Diseño sidebar búsqueda	Fin	T
		Lógica búsqueda	Fin	V
		Eliminar búsqueda	Fin	T
	HU10-Organización y filtro por etiquetas	Filtros búsqueda	Fin	V
		Filtros y búsqueda en carpetas	Fin	T
		Eliminar filtros	Fin	T
8	HU11-Retención documental HU13-Notificaciones	Retención automática según fecha de baja de los documentos	Fin	T
		Notificaciones automáticas mail	Fin	V
9	HU12-Gestión de grupos HU01- Navegación cursos (alumno)	Creación grupos	Fin	V
		Organización por grupos	Fin	V
		Gestión de roles	Fin	V

Tabla 4. Sprints, historias de usuario y tareas

4.1 Sprint 0 – Fase inicial del proyecto

De todas las funcionalidades identificadas en el apartado 2 se describen como la base de este trabajo las siguientes: subida y compartición de documentos, organización de documentos en carpetas, roles y usuarios para manejo de servicios y búsqueda y filtros con etiquetas.

Para mejorar la comprensión de las necesidades del usuario se ha usado un prototipado con el fin de identificar también los diferentes comportamientos o acciones que realizará el usuario y que componen las historias de usuario que se presentan en las siguientes secciones.

4.1.1 Prototipo

Basándonos en las diferentes funcionalidades que especificamos tras la investigación y la obtención de los diferentes objetivos, empezamos a diseñar los primeros bocetos de la aplicación.

En este primer momento nos centramos en el Rol de docente ya que es el que puede realizar más cambios y trabajar más dentro de la aplicación. Y nos basamos en las principales operaciones que puede realizar: iniciar sesión o registrarse, subir documentos, crear o subir carpetas y las diferentes acciones sobre un archivo.

De esta manera obtuvimos los primeros bocetos se detallan en las siguientes subsecciones y se muestran en las figuras 5 a 11.

I. Pantalla inicio de sesión y registro

Las dos pantallas de inicio de sesión y registro, mostradas en la Figura 5, cuentan con un diseño simple y muy similar basado en un formulario con distintos campos y un botón para iniciar sesión o registrarse.

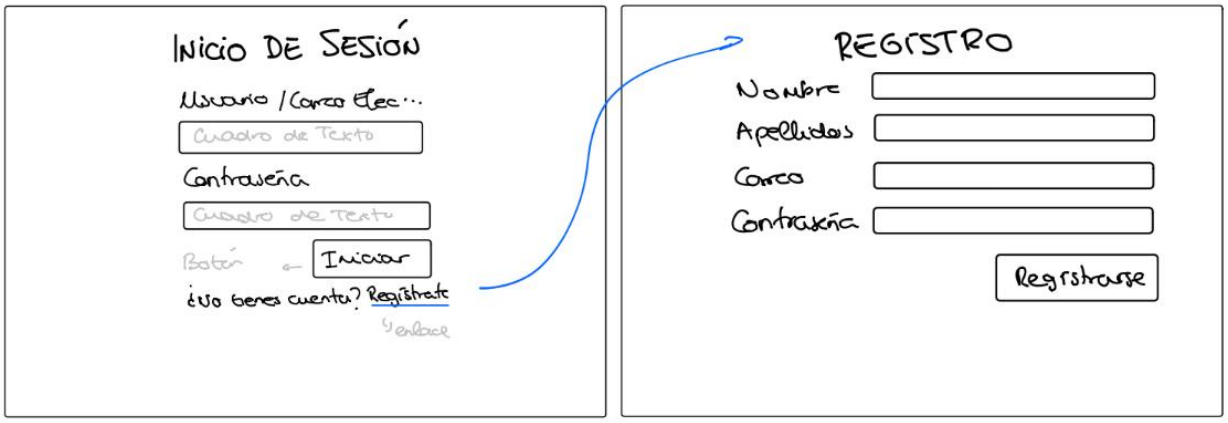

Figura 5. Boceto login y registro

La pantalla de inicio de sesión presenta dos campos para que el usuario introduzca su correo electrónico y la contraseña. Además, se añade un enlace que direcciona al usuario a la pantalla de registro en caso de que sea un nuevo usuario.

Por su lado, la pantalla de registro cuenta con los campos de nombre, apellidos, correo y contraseña.

J. Pantalla de inicio

El boceto que se muestra en la Figura 6, representa la pantalla principal de la aplicación: es la página a la que la aplicación redirecciona al usuario una vez ha realizado el login.

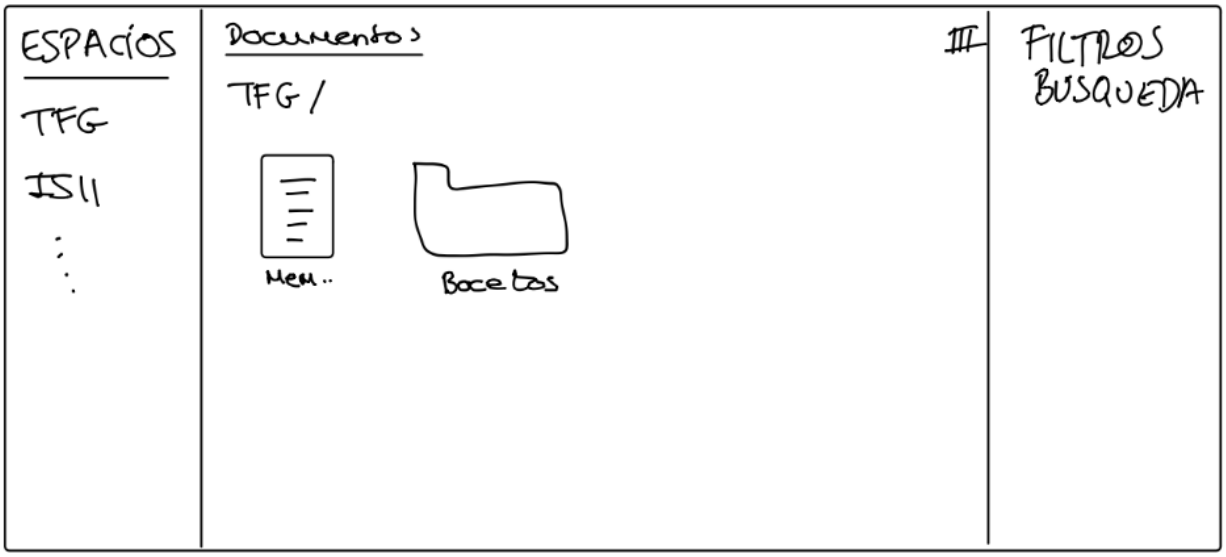

Figura 6. Boceto pantalla inicio

En este primer boceto se divide la pantalla en tres sectores, el primero y más a la izquierda representa el lugar donde aparecen los diferentes cursos. La parte del centro muestra la zona de visualización de los diferentes documentos y carpetas relacionados con el curso seleccionado. Y, por último, la tercera parte más a la derecha sería una zona donde aparecen los filtros y la búsqueda de archivos que se quiere implementar.

K. Pantalla Subir Documento

Para la funcionalidad de subir documento se añade un botón en la pantalla principal (Figura 7). Este botón abre una primera pantalla donde se elige el archivo que se quiere guardar en la aplicación (Figura 8) y tras seleccionarlo y darle al botón de aceptar, se abre una segunda pantalla, que se muestra en la Figura 9.

Figura 7. Boceto subir documento I

Figura 8. Boceto subir documento II

En la pantalla mostrada en la Figura 9, el usuario puede añadir diferentes características del documento como son: las diferentes etiquetas, una fecha de vencimiento, si es visible o no y si permite la subida de archivos por parte de los alumnos.

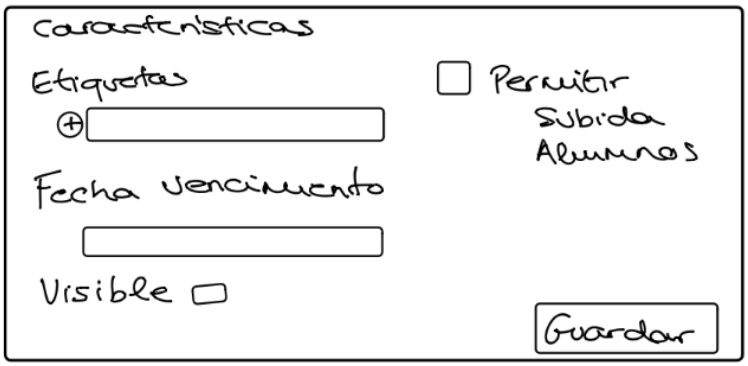

Figura 9. Boceto subir documento III

L. Pantalla añadir carpetas

La propuesta inicial para añadir una carpeta (Figura 10) consiste en incorporar un menú contextual dentro del área de documentos de la pantalla principal. Al hacer clic derecho, dicho menú desplegaría opciones como "crear carpeta", "subir carpeta" y "subir archivo", facilitando así la gestión de archivos por parte del usuario.

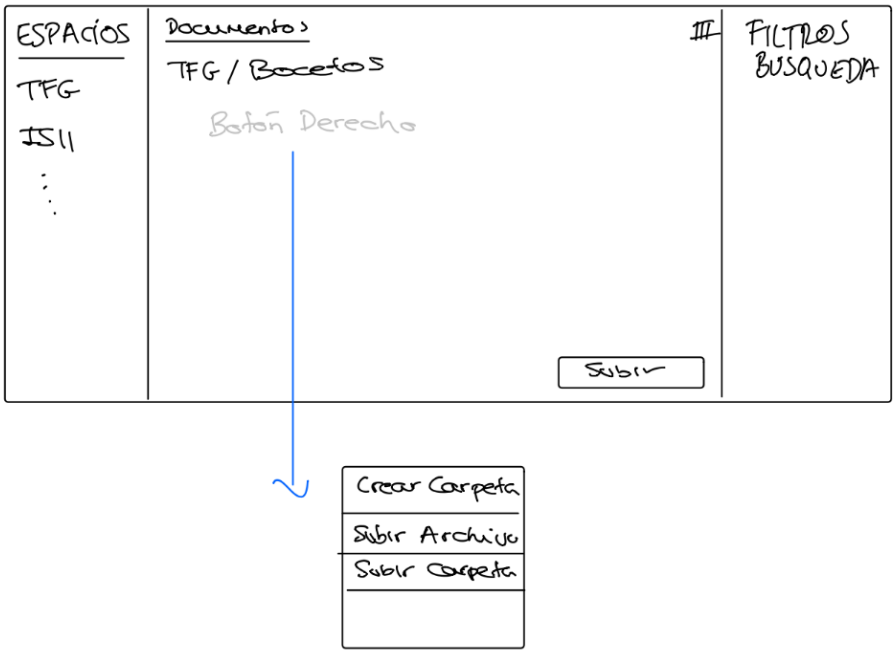

Figura 10. Boceto añadir carpeta

M. Pantalla opciones archivo

Y para las diferentes funcionalidades de modificar características de los documentos la idea consiste nuevamente en añadir un menú contextual a los documentos.

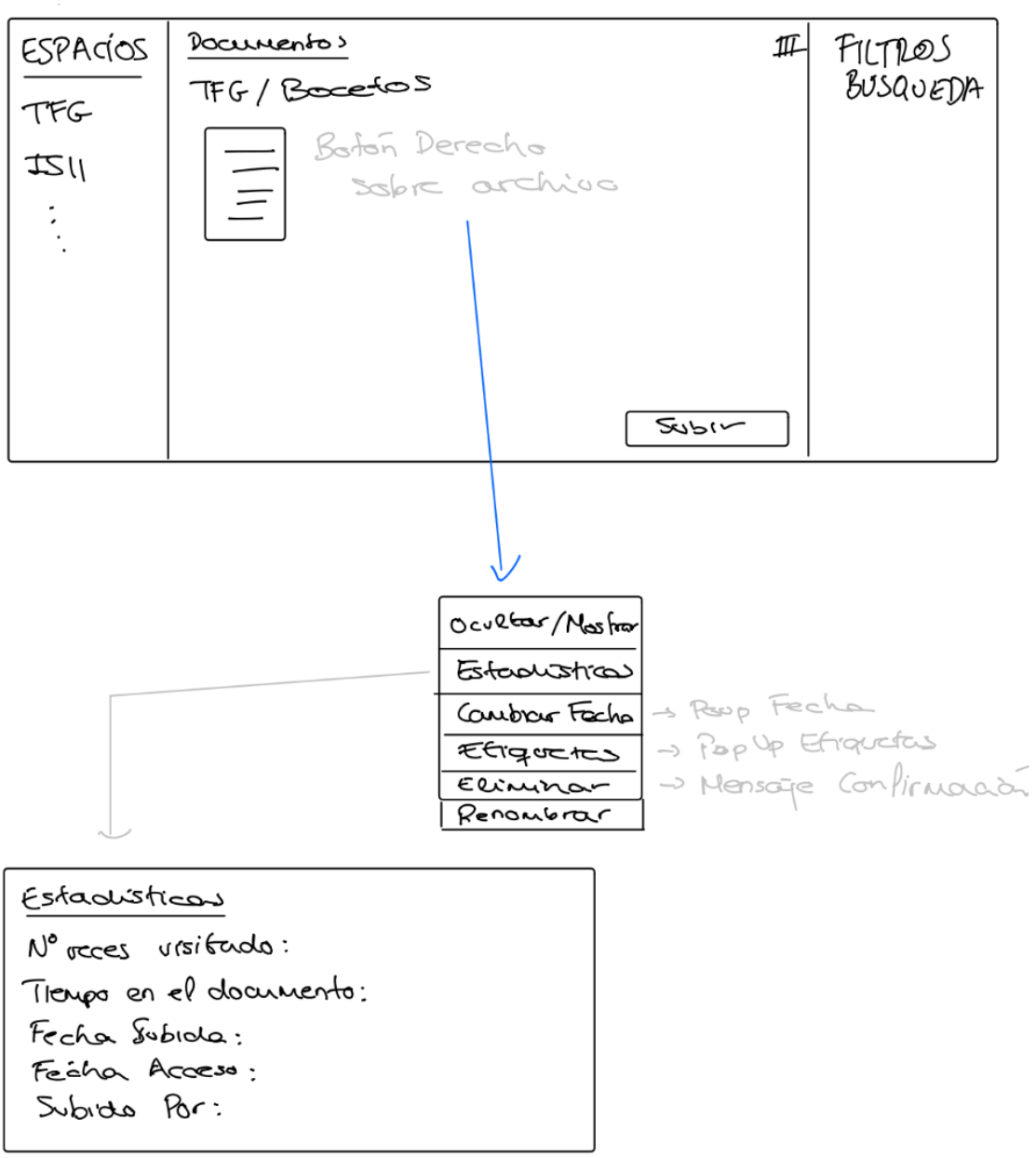

Figura 11. Boceto opciones archivo

Este menú aparecería tras hacer clic derecho sobre un documento y se desplegarían diferentes opciones. Estas opciones son: ocultar/mostrar, para modificar la visibilidad de un archivo, estadísticas donde se abriría un pop up tal y como se ve en la imagen, cambiar fecha que abriría también un popup, etiquetas para mostrar y editar las etiquetas del documento en un nuevo popup y por último las opciones de eliminar y renombrar.

4.1.2 Product Backlog

A partir de las funcionalidades fundamentales y el prototipo se obtienen y desarrollan las siguientes historias de usuario.

Historia de usuario	
ID	HU01
Nombre	Navegación de cursos (alumno)
Prioridad	Media
Riesgo	Media
Descripción	Como usuario quiero poder ver los cursos en los que estoy matriculado para poder acceder a sus documentos.
Validación	- Deben aparecer todos los cursos a los que pertenezca - Deben de estar relacionados con un grupo.

Tabla 5. Historia de usuario 1

Historia de usuario	
ID	HU02
Nombre	Creación y navegación de cursos

Prioridad	Alta
Riesgo	Medio
Descripción	Como docente quiero poder ver los cursos que tengo y crear uno nuevo indicando el grupo.
Validación	- Deben aparecer todos los cursos a los que pertenezca. - Deben de estar relacionados con un grupo. - El curso debe de registrarse correctamente en la base de datos.

Tabla 6. Historia de usuario 2

Historia de usuario	
ID	HU03
Nombre	Carga y representación de documentos
Prioridad	Alta
Riesgo	Medio
Descripción	Como usuario quiero ver los documentos disponibles en un curso.
Validación	- Cuando se selecciona un curso se deben mostrar sus documentos. - Solo se ven documentos visibles para ese curso y grupo. - Solo se ven documentos visibles para el usuario.

Tabla 7. Historia de usuario 3

Historia de usuario

ID	HU04
Nombre	Subida de documentos
Prioridad	Alta
Riesgo	Alto
Descripción	Como docente quiero subir un documento a un curso indicando características como visibilidad, etiquetas y fecha de baja.
Validación	- El documento debe registrarse correctamente en la base de datos. - El formulario debe de validar los datos. - Se deben de guardar correctamente la relación entre el documento y las etiquetas. - Se debe de guardar correctamente si el documento es visible o no.

Tabla 8. Historia de usuario 4

Historia de usuario	
ID	HU05
Nombre	Visualización del contenido de los documentos
Prioridad	Alta
Riesgo	Medio
Descripción	Como usuario quiero ver el contenido de los documentos de un curso.

Validación	- Cuando se selecciona un documento debe abrirse para poder consultar su contenido.

Tabla 9. Historia de usuario 5

Historia de usuario	
ID	HU06
Nombre	Modificación de las características de los documentos
Prioridad	Media
Riesgo	Media
Descripción	Como docente quiero poder modificar características de un documento ya subido.
Validación	- Debe mostrarse un menú contextual con las diferentes opciones. - Todos los cambios deben de realizarse tras la confirmación del docente. - Los cambios se tienen que registrar correctamente en la base de datos.

Tabla 10. Historia de usuario 6

Historia de usuario	
ID	HU07
Nombre	Login y registro de usuarios
Prioridad	Alta

Riesgo	Medio
Descripción	Como nuevo usuario quiero registrarme e iniciar sesión para acceder a mis cursos y documentos.
Validación	- Deben validarse los datos de acceso. - Al registrarse se debe de guardar correctamente el usuario en la base de datos. - Tras el inicio de sesión debe de redirigir a la página principal.

Tabla 11. Historia de usuario 7

Historia de usuario	
ID	HU08
Nombre	Organización de documentos en carpetas
Prioridad	Alta
Riesgo	Medio
Descripción	Como docente quiero poder organizar mis documentos utilizando carpetas
Validación	- Deben poder crearse nuevas carpetas. - Debe poder subirse una carpeta desde el PC del usuario. - Debe de quedar registrada correctamente la carpeta en la base de datos. - Debe de quedar registrada correctamente la relación entre los documentos de la carpeta y la carpeta.

Tabla 12. Historia de usuario 8

Historia de usuario	
ID	HU09
Nombre	Búsqueda de documentos
Prioridad	Alta
Riesgo	Medio
Descripción	Como usuario quiero poder buscar y encontrar documentos por nombre.
Validación	- Deben de mostrarse los resultados buscados. - Deben de manejarse errores o búsquedas sin resultados.

Tabla 13. Historia de usuario 9

Historia de usuario	
ID	HU010
Nombre	Organización y filtro por etiquetas
Prioridad	Alta
Riesgo	Medio
Descripción	Como usuario quiero poder filtrar y encontrar documentos mediante las etiquetas.
Validación	- Deben de mostrarse los resultados filtrados.

	- Deben de manejarse errores o filtrados sin resultados.

Tabla 14. Historia de usuario 10

Historia de usuario	
ID	HU11
Nombre	Retención documental
Prioridad	Medio
Riesgo	Medio
Descripción	Debe haber un control automático que compruebe la fecha de baja de documentos y los elimine si procede.
Validación	- Deben de eliminarse los documentos con fecha de baja de ese día o anterior.

Tabla 15. Historia de usuario 11

Historia de usuario	
ID	HU12
Nombre	Gestión de grupos
Prioridad	Alta
Riesgo	Alta
Descripción	Como docente quiero poder crear y gestionar mis grupos.
Validación	- Deben aparecer los grupos a los que pertenezco. - Debe poder crearse un nuevo grupo.

Tabla 16. Historia de usuario 12

Historia de usuario	
ID	HU13
Nombre	Notificaciones automáticas
Prioridad	Baja
Riesgo	Baja
Descripción	Como usuario quiero recibir notificaciones de nueva subida de documentos.
Validación	- Debe mandarse un correo a todos los usuarios del grupo avisando de que hay un documento nuevo.

Tabla 17. Historia de usuario 13

Historia de usuario	
ID	HU014
Nombre	Base de datos
Prioridad	Alta

Riesgo	Alto
Descripción	Como desarrollador quiero que los datos introducidos sean persistentes.
Validación	- Quiero usar una base de datos. - Quiero que se puedan introducir nuevos datos. - Quiero que se puedan modificar datos. - Quiero que se puedan eliminar datos.

Tabla 18. Historia de usuario 14

4.1.3 Priorización de las historias de usuario.

Tras desarrollar las historias de usuario se decidió que lo más importante era empezar por todas aquellas que tuvieran que ver con las funcionalidades del docente: HU02, HU03, HU04, HU05, HU06, HU08, HU09, HU10 y HU14.

A continuación, se implementará las historias HU11, HU12 y HU13 relativas a la retención documental, la gestión de grupos y las notificaciones automáticas; y, por último, la historia HU01, correspondiente a la funcionalidad del estudiante y la gestión de roles.

4.2 Sprint 1- Creación base de datos

En este sprint se afrontan las tareas relacionadas con la historia de usuario HU14, que es la relacionada con la creación de la base de datos.

Se muestra en la Figura 12, el diagrama ER, creado de la base de datos GestorDocumental que se va a utilizar.

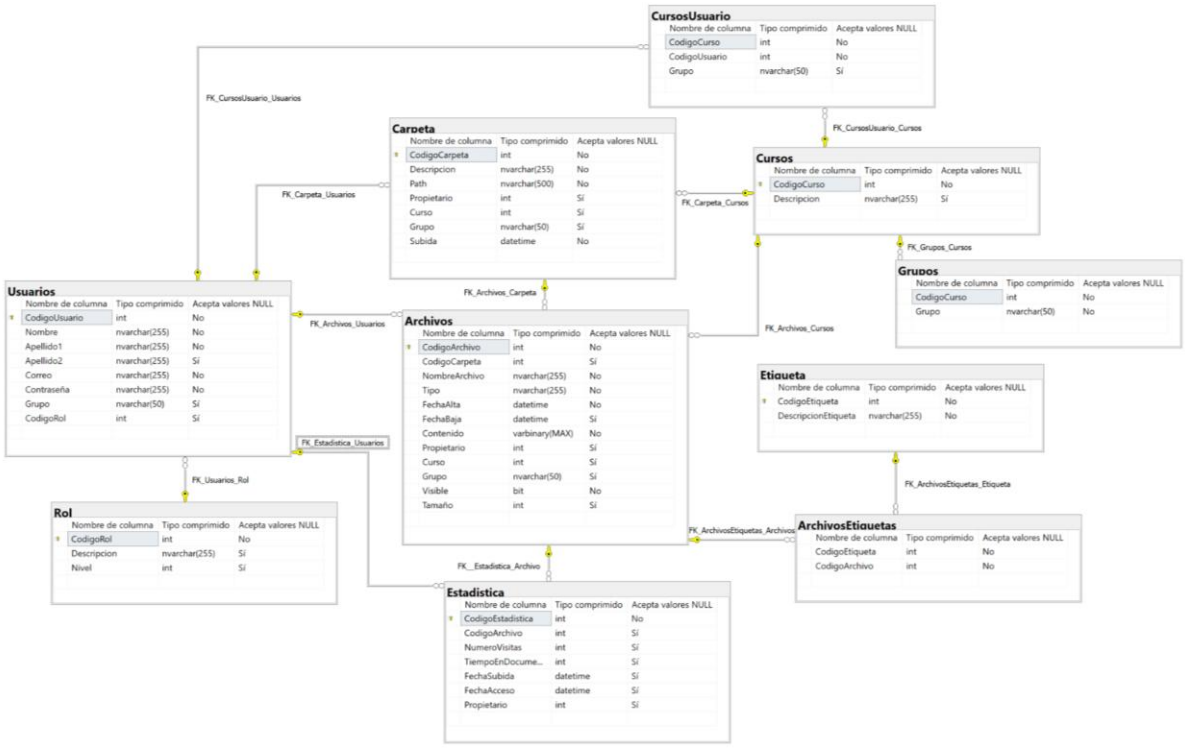

Figura 12. Diagrama ER

En base a los requerimientos de la aplicación se proporciona una estructura de tablas y columnas existentes, que permiten llevar a cabo cada funcionalidad, a continuación, se especifica cada una de ellas:

4.2.1 Tabla archivo

Campo	**Tipo**	**Acepta valores NULL**
CodigoArchivo	int	No
CodigoCarpeta	int	Sí
NombreArchivo	nvarchar(255)	No
Tipo	nvarchar(50)	No
FechaAta	datetime	No
FechaBaja	datetime	Sí
Contenido	varbinary(max)	No
Propietario	int	Si
Curso	int	Si
Grupo	nvarchar(50)	Sí
Visible	bit	No
Tamaño	int	Sí

Tabla 19. Tabla Archivo de la base de datos

Esta tabla se encarga de guardar todos los datos referentes al archivo, que es la unidad base de nuestra aplicación y alrededor de donde giran el resto de los elementos. Dentro de la tabla se gestionan los datos relativos al archivo (nombre, código, fecha alta, fecha baja, etc.) y donde se ubica el mismo (curso, grupo y propietario).

4.2.2 Tabla carpeta

Campo	**Tipo**	**Acepta valores NULL**
CodigoCarpeta	int	No
Descripcion	nvarchar(255)	No
Path	nvarchar(500)	No
Propietario	int	Si
Curso	int	Si
Grupo	nvarchar(50)	Si
Subida	datetime	No

Tabla 20. Tabla carpeta de la base de datos

Esta tabla es la encargada de guardar los datos relativos a las carpetas que contienen los archivos. Se guardan datos como su código, descripción y ubicación además del momento en el que se ha creado dicha carpeta y el grupo y curso al que pertenece.

4.2.3 Tabla cursos

Campo	**Tipo**	**Acepta valores NULL**
CodigoCurso	int	No
Descripcion	nvarchar(255)	Si

Tabla 21. Tabla Cursos de la base de datos

Esta tabla se encarga de guardar los distintos cursos existentes en la aplicación.

4.2.4 Tabla estadística

Campo	Tipo	Acepta valores null
CodigoEstadística	int	No
CodigoArchivo	int	Sí
NumeroVisitas	int	Sí
TiempoEnDocumento	int	Sí
FechaSubida	datetime	Sí
FechaAcceso	datetime	Sí
Propietario	int	Sí

Tabla 22. Tabla Estadística de la base de datos

La tabla estadística es la encargada de recopilar datos sobre un documento en concreto. Para cada uno de ellos recopila información sobre el número de accesos, el tiempo de lectura, cuando se subió, cuál fue el último acceso, etc. De esta manera, el profesor obtiene una visión clara de cómo de interesante o importante para los alumnos es dicho archivo y si estos emplean en él el tiempo necesario para su entendimiento y estudio.

4.2.5 Tabla Etiqueta

Campo	Tipo	Acepta valores NULL
CodigoEtiqueta	int	No
DescripcionEtiqueta	nvarchar(255)	No

Tabla 23. Tabla Etiqueta de la base de datos

Tabla encargada de guardar los datos relativos a las etiquetas que se han ido añadiendo a diferentes archivos.

4.2.6 Tabla ArchivosEtiquetas

Campo	Tipo	Acepta valores NULL
CodigoEtiqueta	int	No
CodigoArchivo	int	No

Tabla 24. Tabla ArchivosEtiquetas de la base de datos

Esta tabla se usa como intermediaria entre los archivos y las etiquetas. Es decir, se guardan las diferentes relaciones de un archivo con las etiquetas que tenga asociadas.

4.2.7 Tabla Usuarios

Campo	Tipo	Acepta valores NULL
CodigoUsuario	int	No
Nombre	nvarchar(255)	No
Apellido1	nvarchar(255)	No
Apellido2	nvarchar(255)	Sí
Correo	nvarchar(255)	No
Contraseña	nvarchar(255)	No
Grupo	nvarchar(50)	Sí
CodigoRol	int	Sí

Tabla 25. Tabla Usuarios de la base de datos

La tabla de usuarios guarda todas las personas registradas en el gestor documental. Guarda datos relativos a cada uno de ellos, como su nombre y correo, además del grupo al que pertenece, junto con su rol y sus datos de acceso.

4.2.8 Tabla Rol

Campo	**Tipo**	**Acepta valores NULL**
CodigoRol	int	No
Descripcion	nvarchar(255)	Sí
Nivel	int	Sí

Tabla 26. Tabla Rol de la base de datos

La tabla rol tiene todos los datos relativos a los distintos tipos de usuarios existentes en la web y su permiso para realizar determinadas acciones dentro de ella.

4.2.9 Tabla CursosUsuario

Campo	**Tipo**	**Acepta valores NULL**
CodigoCurso	int	No
CodigoUsuario	int	No
Grupo	nvarchar(50)	Sí

Tabla 27. Tabla CursosUsuario

Esta tabla se usa como intermediaria entre los usuarios y los cursos. Es decir, se guardan las diferentes relaciones de un usuario con los cursos en los que esté matriculado e indicando el grupo.

4.2.10 Tabla Grupos

Campo	Tipo	Acepta valores NULL
CodigoGrupo	int	No
DescripcionGrupo	string	No

Tabla 28. Tabla Grupos de base de datos

En esta tabla se guardan los datos relacionados con un grupo.

4.2.11 Tabla UserLogAudits

Campo	Tipo	Acepta valores NULL
Id	int	No
UserId	int	No
TargetType	tinyint	No
TargetId	int	No
ActionType	tinyint	No
Weight	float	No
OcurredAt	datetime	No

Tabla 29. Tabla UserLogAudits

Tabla que guarda la auditoría a nivel individual.

4.2.12 Tabla UserLogs

Campo	Tipo	Acepta valores Null
Id	int	No
UserId	int	No

TargetType	tinyint	No
TargetId	int	No
Count	int	No
TotalWeight	float	No
LastUpdated	datetime	No

Tabla 30. Tabla UserLogs

Tabla que guarda la auditoría a nivel acumulado.

4.3 Sprint 2 – Creación y navegación de cursos

En este sprint se crea el proyecto con las tecnologías que se han explicado en el capítulo 3 para poder empezar a desarrollar el resto de las historias de usuario.

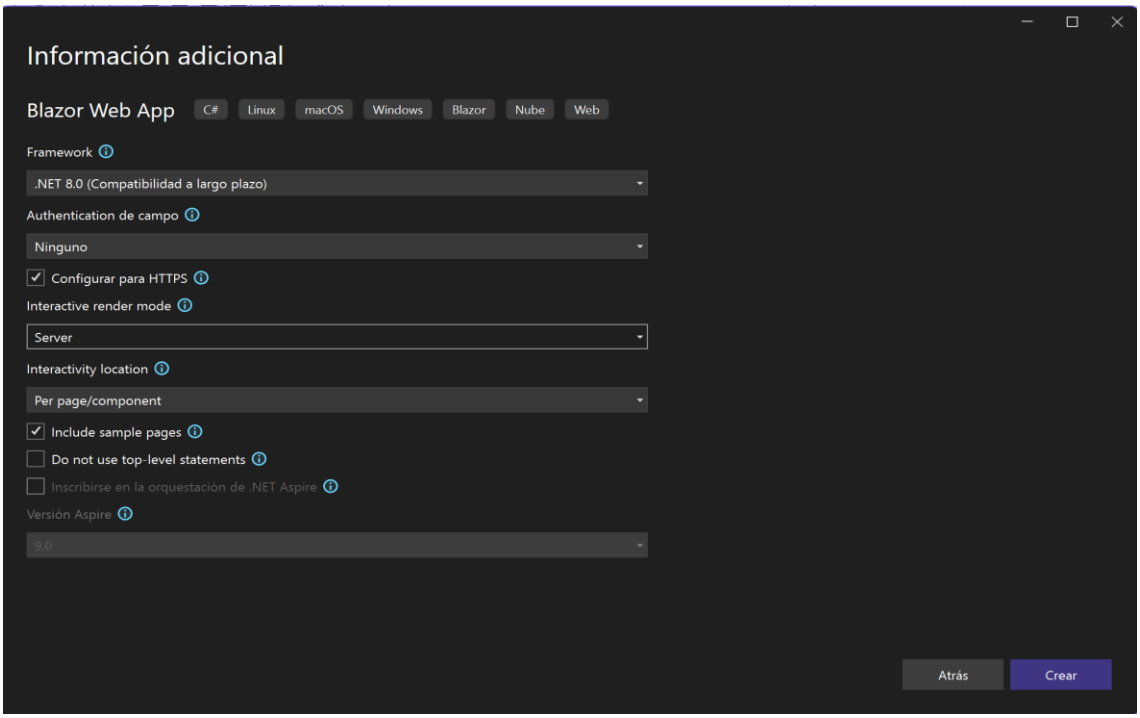

Figura 13. Configuración proyecto Blazor Server

En la Figura 13 se puede observar la configuración que fue escogida para la aplicación, empezando por el framework. Se elige net 8.0 ya que aporta compatibilidad a largo plazo para el proyecto, y sin autenticación.

Respecto a la elección de Interactive Render Mode, Blazor ofrece las siguientes opciones:

- Server: ejecuta la aplicación en el servidor y la comunicación servicio – cliente se realiza mediante una conexión a tiempo real (SignalR).
- WebAssembly: ejecuta la aplicación completamente en el cliente.
- Auto: selecciona entre los dos primeros según si hay conexión activa del servidor o no.

Se escogió la opción Server por lo indicado en el capítulo 3 apartado 3.3.1.

El resto de las opciones de configuración, vienen por defecto al crear el proyecto y se dejaron así.

- Intercative location, per page/component: solo las páginas que estén marcadas con la directiva *@rendermode InteractiveRenderMode,* serán interactivas.
- Inlcude Simple Pages: visual studio incluye en el proyecto creado paginas básicas generadas automáticamente, aportando una estructura inicial funcional del proyecto.

4.3.1 Configuración de la base de datos en el proyecto

Lo primero que se hizo fue crear el archivo *GestorDocumentalDbContext.cs* que es una clase de C# que aporta un punto de entrada a la base de datos. Esta clase tiene un constructor público y se le agregan tantos DBSet como tablas o colecciones haya en la base de datos a la que se va a acceder. Es importante definir los DBSet porque son la forma de conectar y acceder a la base de datos de manera sencilla sin necesidad de escribir las consultas SQL manualmente. Esto es porque usamos Entity Framework Core que permite conectarse e interactuar con la base de datos mediante LINQ[22][23].

Después, se registra la instancia del DBContext en el archivo *Program.cs*, necesario porque Blazor Server usa inyección de dependencias para administrar los diferentes servicios de la aplicación, y además, sirve para obtener la cadena de conexión [19].

```
// Obtener la cadena de conexión desde la configuración
var connectionString = builder.Configuration.GetConnectionString("GestorDocumentalConnection");

// Configurar DbContextFactory con SQL Server
builder.Services.AddDbContextFactory<GestorDocumentalDbContext>(options =>
    options.UseSqlServer(connectionString));
```

Figura 14. Instancia de dbcontext en program.cs

La cadena de conexión denominada en este caso "GestorDocumentalConnection" se inicializa en *appsettins.json*.

```
"ConnectionStrings": {
    "GestorDocumentalConnection": "Server=.\\SQLEXPRESS;Database=GestorDocumental;Trusted_Connection=True;MultipleActiveResultSets=true;Encrypt=False;"
},
```

Figura 15. Cadena de conexión de la base de datos

La cadena de conexión cuenta con diferentes parámetros que se explican a continuación:

- Server=.\\SQLEXPRESS: indica el servidor de la base de datos que en este caso es local (".") y la instancia de SQL Server Express que usamos.
- Database=GestorDocumental: indica el nombre de nuestra base de datos.
- Trusted_Coneecion=True: se usa para utilizar la autenticación de Windows en lugar de usar las credenciales usuario/contraseña. En nuestro caso funciona así porque tenemos permisos en el servidor SQL Server.
- MultipleActiveResultSets=true: es necesario en nuestro caso pues nos permite realizar múltiples consultas simultáneas en una misma conexión.
- Encrypt=false: no tenemos ningún certificado SSL configurado en nuestro servidor por lo tanto lo tenemos que establecer a false.

4.3.2 Configuración de la arquitectura de 3 capas

Como se ha explicado previamente vamos a usar en nuestro proyecto la arquitectura de 3 capas. Por ello se han creado diferentes componentes para cada una de las capas.

- Capa de presentación: es aquella que contiene los diferentes componentes y páginas Blazor que utiliza y ve el usuario. Dentro del proyecto creado se organiza esta parte dentro de la carpeta Components diferenciando entre:
    - Components/Layout: aquí se encuentran los diseños generales que se usan en la aplicación y el índice de navegación de los cursos.
    - Components/Pages: aquí se muestran todas las páginas Blazor que se usan en la aplicación y que están relacionadas con las diferentes funcionalidades. Las pantallas de login y registro, la pantalla principal, las pantallas de subir archivo y carpeta; y resto de las funcionalidades de la aplicación.
- Capa de negocio: aquí aparecen los archivos del servicio de los negocios y sus interfaces.
    - ArchivoService: donde aparecen las diferentes funcionalidades del manejo de archivos y carpetas.
    - CursoService: toda la gestión de los cursos.
    - UsuarioService: guarda la lógica de los usuarios y los roles.
    - AuthService: se encarga de realizar la autenticación de los usuarios y el manejo de las sesiones activas.
    - CarpetaService: encargado de la gestión de las carpetas.
    - EstadisticasService: gestión de las estadísticas de los documentos.
    - EtiquetasService: gestión de las etiquetas de los documentos.

    - EmailService: encargado de la gestión de las notificaciones por email.
    - RetencionDocumentalService: gestión de la retención documental.
    - GrupoService: gestión de los grupos
- Capa de acceso a datos: todos los repositorios y sus interfaces además de las diferentes entidades y el GestorDocumentalDbContext.

Se utilizan interfaces con el fin de no usar solo las clases concretas y así obtener modularidad para tener la opción de cambiar o actualizarse sin afectar a las otras capas del sistema. Además, las interfaces aportan flexibilidad dejando la puerta abierta a la posibilidad de cambiar de tecnologías sin necesidad de modificar la lógica completa de negocio, por ejemplo, cambiar de sql a MongoDB creándose únicamente una nueva instancia en la interfaz.

Al estar separado, los cambios, modificaciones o actualizaciones de los repositorios no producen ningún impacto al resto del código de las capas lo cual facilita el mantenimiento del código.

4.3.3 Diseño de la pantalla principal

En este *sprint* se desarrolló la base del diseño de la página principal, siguiendo el prototipo descrito en el apartado 4.1.1. Se partió del proyecto base de Visual Studio para Blazor, el cual ya incluye una estructura con una zona de navegación lateral (izquierda) y un área principal de contenido. Esta estructura está definida en *MainLayout.razor*, que incluye el panel de navegación (*NavMenu.razor*) y la zona central (*Home.razor*).

Tal como se muestra en la Figura 6 del prototipo, la página se divide en tres sectores. Como esta estructura coincidía con la que ofrece Blazor, se mantuvo *MainLayout.razor* y se modificaron *NavMenu.razor* y *Home.razor* para adaptar la visualización de cursos y documentos.

En cuanto a la navegación de cursos, al hacer *login* y acceder a la página principal, se cargan automáticamente los cursos asociados al usuario, mostrándose en

lista como enlaces para visualizar sus documentos. Los docentes disponen además de un botón en la parte superior para añadir nuevos cursos.

En la zona central dedicada a los documentos, se contemplaron dos situaciones. Si el usuario no ha seleccionado ningún curso, se muestra un mensaje de bienvenida con instrucciones. Si se ha seleccionado un curso, pueden darse dos casos: que tenga documentos (los cuales se visualizan, incluyendo carpetas si las hay), o que no tenga (mostrándose un mensaje informativo). En ambos casos, los docentes ven un botón en la esquina inferior derecha para subir documentos.

Para representar los documentos, se optó por un diseño similar a la vista de iconos de Windows, utilizando tarjetas de Radzen. Estas permiten personalizar el icono según el tipo de archivo y mostrar información relevante. Finalmente, se decidió mostrar únicamente el icono del tipo de archivo (como PDF) y su nombre justo debajo.

```
if (item is Archivo archivo && archivo.Visible)

    <RadzenCard class="rz-my-2 card-hover"
                Style="max-width: 420px;  box-shadow: none; border: none; margin:0; padding: 0;"
                @ondblclick="() => VerContenidoArchivo(archivo)"
                @oncontextmenu="args => { ShowContextMenu(args, archivo); }">
        <RadzenStack Orientation="Orientation.Vertical" JustifyContent="JustifyContent.Right" AlignItems="AlignItems.Center" Gap="0.5rem" class="rz-p-4">
            <RadzenImage Path="@ObtenerIconoArchivo(archivo.Tipo)" Style="width: 50px; height: 50px;" />
            <RadzenText TextStyle="TextStyle.Body1">
                <a href="javascript:void(0);" @onclick="() => VerContenidoArchivo(archivo)"
                   style="color: black; text-decoration: none; font-weight: bold;">
                    @archivo.NombreArchivo
                </a>
            </RadzenText>
        </RadzenStack>
    </RadzenCard>
```

Figura 16. Código para RadzenCard

En la *Figura 16* se muestra el código utilizado para construir las tarjetas que representan los documentos, junto con las configuraciones aplicadas a su diseño y funcionalidad:

- Class="rz-my-2 card-hover" aplica clases CSS que añaden margen vertical y un efecto *hover*, indicando al usuario que puede hacer doble clic.
- Style="max-width: 420px; box-shadow: none; border: none; margin:0; padding: 0;" fija un ancho uniforme para todas las tarjetas y elimina bordes y sombras, buscando una integración visual limpia con el fondo.

- Se manejan dos eventos: uno al hacer doble clic (para visualizar el documento) y otro al hacer clic derecho (para mostrar un menú contextual que permite modificar propiedades del documento).
- *RadzenStack* organiza los elementos de forma vertical, alineados al centro y a la derecha, con una separación de 0.5rem entre ellos.
- *RadzenImage* se utiliza para mostrar un icono representativo del tipo de archivo.
- *RadzenText* muestra el nombre del documento, que también actúa como enlace para su visualización.

4.3.4 Creación y navegación de cursos

Como se explicó en el sprint anterior, los cursos se cargan y visualizan en *NavMenu.razor*. Para los docentes, además, se muestra un botón que permite crear un nuevo curso. Al hacer clic en él, se despliega una ventana emergente con un formulario para introducir el nombre del curso y el grupo correspondiente.

Para implementar estas funcionalidades, se definió primero la clase *Curso.cs* (Figura 17), que contiene las propiedades reflejadas en la tabla Cursos de la base de datos. Posteriormente, se añadió un nuevo DbSet en el DbContext para permitir el acceso directo a dicha tabla.

```
using System.ComponentModel.DataAnnotations;

namespace GestorDocumental.Data.Entities
{
    23 referencias
    public class Curso
    {
        [Key]
        3 referencias
        public int CodigoCurso { get; set; }
        6 referencias
        public string? Descripcion { get; set; }
        0 referencias
        public string? Grupo { get; set; }
    }
}
```

Figura 17. Entidad curso.cs

A continuación, se desarrolló un método para obtener todos los cursos en los que el usuario está matriculado, a partir de las relaciones en la tabla CursosUsuarios,

almacenándolos en una lista. Esta lista se muestra con todos los cursos dispuestos verticalmente, cada uno con su descripción y un desplegable con los grupos asociados. Cada curso se representa mediante un *NavLink*, que actúa como enlace para cargar la pantalla de documentos del curso seleccionado (Figura 18).

```
@if (cursos != null && cursos.Any())
{
    @foreach (Curso curso in cursos)
    {
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="@($"/home/{curso.CodigoCurso}")">
                📖 @curso.Descripcion
            </NavLink>
        </div>
    }
}
```

Figura 18. Lista de cursos para la navegación

Como se mencionó, se añade un botón que abre una ventana modal para crear un nuevo curso, con dos campos: nombre y grupo. Dentro del modal, hay botones para guardar o cancelar (cerrando el modal sin cambios). La creación se implementa usando *data binding* de ASP.NET Core Blazor mediante *@bind* [24], asociando los campos de entrada con las propiedades Descripcion y Grupo de una instancia de Curso.

Al pulsar "guardar", se valida que ambos campos estén completos y, tras la confirmación del usuario, se invoca el servicio de cursos para insertar el nuevo curso en la base de datos. Al finalizar, se cierra el modal y se actualiza *NavMenu.razor* para reflejar el nuevo curso.

Asimismo, se habilita la opción de añadir un grupo a un curso mediante un menú contextual que se activa con clic derecho. Este abre otro modal con un campo para introducir el nombre del grupo, un botón para guardar (con validación previa e inserción en la base de datos mediante el servicio correspondiente), y otro botón para cancelar y cerrar la ventana.

4.3.5 Pruebas realizadas

En este sprint se han realizado pruebas de funcionamiento en las que se ha validado, fundamentalmente, el diseño inicial de la pantalla principal y la correcta adecuación a la barra lateral en la que se ha comprobado que los cursos se muestran de manera correcta y que su navegación es correcta y sin errores.

A su vez se ha comprobado que se realiza la conexión con la base de datos de una manera general y que todos estos nuevos componentes interactúan entre ellos de manera correcta.

4.4 Sprint 3 – Documentos

En este nuevo sprint se llevan a cabo las tareas relacionadas con las historias de usuario HU03, HU04 y HU05, vinculadas a la carga y representación de los documentos, a visualización del contenido de estos y a la subida de nuevos documentos.

4.4.1 Carga y representación de los documentos

Una vez seleccionado un curso, se cargan en *Home.razor*, que es la interfaz de visualización e interacción; todos los documentos asociados a dicho curso, como se explicó en el sprint anterior.

Para acceder a la información almacenada en la base de datos, se creó la entidad *Archivo.cs* (), que incluye las propiedades correspondientes a los campos de la base de datos, junto con su DbSet para facilitar el acceso directo.

```
85 referencias
public class Archivo
{
    [Key]
    19 referencias
    public int CodigoArchivo { get; set; }
    3 referencias
    public int? CodigoCarpeta { get; set; }
    31 referencias
    public string NombreArchivo { get; set; }
    8 referencias
    public string Tipo { get; set; }
    3 referencias
    public DateTime FechaAlta { get; set; }
    4 referencias
    public DateTime? FechaBaja { get; set; }
    8 referencias
    public byte[]? Contenido { get; set; }
    4 referencias
    public int Propietario { get; set; }
    4 referencias
    public int? Curso { get; set; }
    5 referencias
    public string? Grupo { get; set; }
    2 referencias
    public int? Tamaño { get; set; }
    17 referencias
    public bool Visible { get; set; }
    [NotMapped]
    3 referencias
    public List<string>? Etiquetas { get; set; }
}
```

Figura 19. Entidad Archivo.cs

La carga de los documentos en este caso depende del parámetro del código del curso recibido; por eso se realiza la carga en el método *OnParameterSetAsync* (Figura 20), que, además, se ejecuta de forma asíncrona ya que requiere una consulta a base de datos (se explica en el apartado 3.3.1).

Además de cargar los documentos, también se recupera la información completa del curso al que pertenecen, necesaria para la interfaz de usuario, específicamente para mostrar el nombre del curso en el encabezado de la página.

```
protected override async Task OnParametersSetAsync()
{

    await CargarCurso();
    await CargarArchivos();
}
```

Figura 20. Carga del curso y sus archivos

Al cargar un documento, se crea una tarjeta visual (*RadzenCard*) y, para mejorar la experiencia visual, se implementa el método *ObtenerIconoArchivo*, que asigna un icono representativo según el tipo de archivo. Por ejemplo, para archivos en formato

PDF, se utiliza un icono específico (almacenado en la carpeta wwwroot/icons), pero esta implementación ahora permite gestionar otros tipos de archivos, como Word, y asignarles su propio icono correspondiente.

Figura 21. Icono archivo pdf

4.4.2 *Visualización del contenido*

Para visualizar los documentos, cada tarjeta Radzen que representa un archivo tiene un evento *ondblclick* que, al hacer doble clic, abre el documento para ver su contenido. Este evento invoca el método *VerContenidoArchivo*, al que se le pasa como parámetro el archivo seleccionado. En dicho método, se verifica que el documento no sea nulo, se comprueba si es un archivo PDF y, luego, se llama a una función JavaScript para abrirlo y mostrar el contenido (Figura 22).

Dado que Blazor no puede enviar datos binarios directamente al cliente y JavaScript no entiende objetos byte[] de .NET, el documento se convierte en una cadena codificada en Base64. En el cliente, un script JavaScript decodifica esta cadena con *atob*, generando un array de bytes (*byteArray*). Finalmente, se crea un archivo PDF a partir de los bytes obtenidos, se asigna una URL temporal y se utiliza *window.open* para mostrar el documento en una nueva pestaña del navegador.

```
function abrirPdf(base64Data) {
            const byteCharacters = atob(base64Data);
    const byteNumbers = new Array(byteCharacters.length);
    for (let i = 0; i < byteCharacters.length; i++) {
        byteNumbers[i] = byteCharacters.charCodeAt(i);
            }
    const byteArray = new Uint8Array(byteNumbers);
    const file = new Blob([byteArray], {type: 'application/pdf' });
    const fileURL = URL.createObjectURL(file);
    window.open(fileURL, '_blank');
}
```

Figura 22. Script para visualizar el documento

Cuando se abre un documento se actualizan además el número de visitas asociado a ese documento y el tiempo que el usuario lo mantiene abierto. El servicio de estadísticas se encarga de hacer estos cambios. Para contabilizar el tiempo (en segundos) que se mantiene abierto un documento se utilizar un *StopWatch,* que se inicia cuando se hace doble clic en el documento para abrirlo y se detiene cuando se cierra el documento. Para ello se añade una nueva función en el script de *VerDocumento* (Figura 24) que monitorea esta actividad y ejecuta el método *OnVisorCerrado* (Figura 23), que detiene el stopwatch y actualiza el tiempo del documento en sus estadísticas.

```
[JSInvokable]
public async Task OnVisorCerrado()
{
    Console.WriteLine("OnVisorCerrado llamado");
    _stopwatch.Stop();

    if (codArchivoActual.HasValue)
    {
        await EstadisticaService.GuardarTiempoVisualizacion((int)codArchivoActual, _stopwatch.Elapsed);
        codArchivoActual = null;
    }

    StateHasChanged();
}
```

Figura 23. Método cierre del visor

```
function monitorCierreVentana(win, dotNetHelper) {
    console.log("monitorCierreVentana iniciado...");
    const interval = setInterval(() => {
        if (win.closed) {
            clearInterval(interval);
            console.log("Ventana cerrada. Invocando OnVisorCerrado...");
            dotNetHelper.invokeMethodAsync("OnVisorCerrado")
                .then(() => console.log("OnVisorCerrado invocado correctamente"))
                .catch(err => console.error("Error llamando a OnVisorCerrado:", err));
        }
    }, 1000);
}
```

Figura 24. Método cierre ventana

4.4.3 Añadir documentos

Una de las funciones principales del proyecto para los docentes es la subida de nuevos documentos a un curso, por lo que se priorizó la accesibilidad y simplicidad en la interfaz de usuario. Para ello, se implementó un botón en la esquina inferior derecha de la pantalla de documentos que, al pulsarlo, inicia el proceso de subida y configuración de características del documento.

Aunque inicialmente el diseño contemplaba tres etapas (clic en el botón, selección de archivo y características en modales separados), finalmente se simplificó a dos: botón de subida y un único modal que permite seleccionar el archivo y añadir sus características, con el fin de mejorar la experiencia del usuario.

Este modal, desarrollado en el componente *SubirArchivo.razor*, incluye un selector de documentos, campos para añadir etiquetas, una posible fecha de baja, visibilidad del documento, el grupo del curso correspondiente, y botones de subir y cancelar. Al hacer clic en guardar, y tras confirmar la acción, se llama al servicio encargado de gestionar documentos, que a su vez comunica con el repositorio para almacenar el archivo en la base de datos. Una vez guardado, se cierra el modal y se actualiza la interfaz. Si se pulsa cancelar, simplemente se cierra el modal y se vuelve a la pantalla principal.

Para seleccionar el archivo desde el equipo del docente se utiliza el componente *RadzenUpload* (Figura 25), configurado con *Multiple=false*, permitiendo solo un archivo por operación. Además, incluye la opción de eliminar y reemplazar el archivo seleccionado mediante un icono de eliminación ("x").

```
<label>Selecciona un archivo: (*)</label>
<RadzenUpload @ref="upload"
Auto="false"
Multiple="false"
Change="OnFileSelected"
Style="width: 100%"
ChooseText="Seleccionar archivo" />
```

Figura 25. Codigo Radzen Upload para subir un archivo

Al seleccionar un documento, se ejecuta el método vinculado al evento *Change* del componente, *OnFileSelected* (Figura 26). Este método se encarga de obtener el archivo, procesarlo y asegurarse que no es nulo, y crear una nueva instancia de la entidad *Archivo.cs* para el almacenamiento posterior en la base de datos.

Esta nueva instancia se inicializa en este momento con algunos valores como son el nombre del documento, el tipo, el tamaño, el contenido, la fecha de alta, el curso, el propietario, la carpeta si está dentro de una carpeta y la visibilidad (por defecto, visible).

```
private async Task OnFileSelected(UploadChangeEventArgs args)
{
    var archivo = args.Files.FirstOrDefault();
    if (archivo != null)
    {
        try
        {
            using (var stream = new MemoryStream())
            {
                await archivo.OpenReadStream(maxAllowedSize: 50 * 1024 * 1024).CopyToAsync(stream);
                stream.Position = 0;

                archivoSubido.NombreArchivo = archivo.Name;
                archivoSubido.Tipo = archivo.ContentType;
                archivoSubido.Tamaño = (int)archivo.Size;
                archivoSubido.Contenido = stream.ToArray();
                archivoSubido.FechaAlta = DateTime.Now;
                archivoSubido.Curso = CodigoCurso;
                archivoSubido.Propietario = CodigoUsuario;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error al procesar el archivo: {ex.Message}");
        }
    }
}
```

Figura 26. Método archivo seleccionado

Para poder guardar el contenido del archivo correctamente, se procesa el documento utilizando un flujo de memoria (*MemoryStream)* que funciona como un buffer temporal en el que se guarda el contenido binario del archivo, que se lee a través del método *OpenReadStream*, y se guarda en la propiedad contenido (*byte[]*) de la instancia nueva de archivo.

Al método de lectura se le añade el parámetro *maxAllowedSize* que permite configurar el tamaño máximo por defecto del archivo que se va a procesar. Blazor tiene un límite de 500 KB para leer archivos. Este límite lo podemos también configurar en *program.cs* configurando dos servicios [27], según se muestra en la Figura 27.

```
//limite de tamaño de los archivos
builder.Services.Configure<IISServerOptions>(options =>
{
    options.MaxRequestBodySize = 50 * 1024 * 1024;
});

builder.Services.Configure<FormOptions>(options =>
{
    options.MultipartBodyLengthLimit = 50 * 1024 * 1024;
});
```

Figura 27. Límite tamaño de los archivos

El primero sirve para modificar el tamaño máximo de una solicitud (request) que puede recibir un servidor IIS. Y el segundo, se encarga de establecer el límite de tamaño para los archivos que se envían en formularios como podría ser la subida de un archivo [28][29].

Para la asignación de etiquetas al documento que se va a subir, se implementó un campo de entrada habilitado para escritura dinámica y con un botón de acción ("+") que permite añadir todas las etiquetas que el usuario va escribiendo. Este campo de entrada ha sido configurado como un *RadzenAutoComplete* (Figura 28), que gracias su funcionamiento va proporcionado al usuario sugerencias en tiempo real basadas en coincidencias con etiquetas que ya están guardadas en la base de datos.

```
<div>
    <label><b>Etiquetas:</b></label>
    <div style="display: flex; gap: 10px;">
        <!-- AutoComplete para buscar etiquetas existentes o escribir una nueva -->
        <RadzenAutoComplete @bind-Value="nuevaEtiqueta"
        Data="@etiquetas.Select(e => e.DescripcionEtiqueta).ToList()"
        Placeholder="Escribe o selecciona una etiqueta..."
        Style="flex-grow: 1;"
        AllowClear="true" />

        <RadzenButton Icon="add" ButtonStyle="ButtonStyle.Primary" Click="AgregarEtiqueta" Disabled="@string.IsNullOrWhiteSpace(nuevaEtiqueta)" />
    </div>
</div>

@if (listaEtiquetas.Any())
{
    <div class="etiquetas" style="display: flex; flex-wrap: wrap; gap: 10px;">
        @foreach (var etiqueta in listaEtiquetas)
        {
            <span class="etiqueta" style="display: flex; align-items: center; background: ■#e0e0e0; padding: 5px 10px; border-radius: 5px;">
                @etiqueta
                <RadzenButton Icon="close" ButtonStyle="ButtonStyle.Danger" Click="@(() => EliminarEtiqueta(etiqueta))" Size="ButtonSize.Small" Style="margin-left: 5px;" />
            </span>
        }
    </div>
}
```

Figura 28. RadzenAutoComplete para etiquetas

Por tanto, durante la carga inicial del componente, se realiza una llamada el servicio de etiquetas para conseguir y guardar en una lista todas las etiquetas disponibles en el momento en la base de datos. Esta lista se vincula con la propiedad Data del componente, configurando que solo salga la descripción de la etiqueta y no el código.

```
protected override async Task OnInitializedAsync()
{
    etiquetas = await EtiquetasService.ObtenerEtiquetas();
}
```

Figura 29. Obtención de las etiquetas

Cada etiqueta añadida o seleccionada por el usuario visualizan una a una debajo del componente, cada una con una opción para poder eliminarse con el botón de acción (“x”).

El contenido del campo de texto del componente se vincula mediante *@bind* a una nueva propiedad creada como *private string nuevaEtiqueta* = "", que, si se hace clic en el botón de añadir etiqueta, el valor se guarda en una lista que va almacenando las nuevas etiquetas para posteriormente guardarlas en base de datos.

La fecha de baja se puede seleccionar, si se quiere, directamente en un calendario que está vinculado con la propiedad de *FechaBaja* de la nueva instancia del Archivo. El componente *RadzenDatePicker*, se configura para que una vez seleccionada una fecha muestre solo la fecha con el formato dd/MM/yyyy y no la hora.

```
<div>
    <label>Fecha de vencimiento:</label>
    <RadzenDatePicker Value="@archivoSubido.FechaBaja" ValueChanged="@((DateTime? value) => archivoSubido.FechaBaja = value)" ShowTime="false" DateFormat="dd/MM/yyyy"/>

</div>
```

Figura 30. Calendario selección fecha baja

Se debe seleccionar también un grupo, por defecto, si está dentro de un grupo en específico saldrá ese. Se puede elegir y cambiar el grupo por cualquiera de los grupos del curso.

Al hacer clic en "Guardar", se verifica que se haya seleccionado un documento (requisito obligatorio). Si es así, se solicita al usuario la confirmación para proceder con la subida. Tras esta confirmación, se crea un nuevo documento en la base de datos mediante el servicio *ArchivoService.cs*. Una vez creado y obtenido su identificador único, se gestionan las relaciones con las etiquetas asociadas, si las hay.

Cada etiqueta seleccionada se evalúa para comprobar si ya existe en la base de datos. Si se trata de una etiqueta nueva, se crea utilizando el servicio de etiquetas. Luego, tanto para etiquetas nuevas como existentes, se establece la relación entre el archivo y cada etiqueta mediante una llamada al mismo servicio, asociando sus respectivos identificadores.

Una vez finalizado este proceso, se cierra el modal y se actualiza la visualización principal. En caso de pulsar "Cancelar", se revierten las modificaciones temporales y se cierra el componente, regresando a la vista inicial de los documentos.

```
private async Task ConfirmarSubida()
{

    await ArchivoService.GuardarArchivoAsync(archivoSubido);


    if (listaEtiquetas.Any())
    {
        List<Etiqueta> etiquetasAgregarBD = new List<Etiqueta>();
        foreach (string etiqueta in listaEtiquetas)
        {
            Etiqueta? etiquetaBD = etiquetas.FirstOrDefault(e => e.DescripcionEtiqueta == etiqueta);
            if (etiquetaBD == null)
            {
                Etiqueta etiquetaNueva = new() { DescripcionEtiqueta = etiqueta };
                etiquetaBD = await EtiquetasService.CrearEtiqueta(etiquetaNueva);
                etiquetas.Add(etiquetaBD);
            }
            etiquetasAgregarBD.Add(etiquetaBD);
        }

        await EtiquetasService.AgregarEtiquetasArchivo(archivoSubido.CodigoArchivo, etiquetasAgregarBD);
    }


    DialogService.Close(true);
}
```

Figura 31. Guardado de nuevo archivo y sus etiquetas

4.4.4 Pruebas realizadas

En este sprint se han realizado pruebas de funcionamiento con el fin de comprobar que los documentos se suben a la plataforma y se guardan correctamente en la base de datos a pesar de cada uno de estos ser de distintos tipos.

Por otro lado, se ha verificado que tanto las etiquetas como las estadísticas se asignan de manera correcta a cada uno de los archivos además de comprobar que estos están asignados a su correcto curso.

4.5 Sprint 4 – Modificación características de los documentos

En este sprint se llevan a cabo las tareas relacionadas con la historia de usuario HU06 que es la relacionada por las opciones de modificación de características de los archivos.

En el prototipo inicial que se creó y que se puede ver en el apartado 4.1.1 del sprint 0, se definió que los documentos iban a tener un menú contextual donde aparecerían diferentes opciones para modificar el documento que se seleccionase.

Para implementar esta funcionalidad, se integró un menú contextual a cada componente RadzenCard, que como se explica en el apartado 4.5.1 sirve para representar un documento, mediante el evento *OnContextMenu*.

```
void ShowContextMenu(MouseEventArgs args, Archivo archivo)
{
    clickArchivo = true;
    ContextMenuService.Open(args,
        new List<ContextMenuItem>
            {
            new ContextMenuItem(){ Text = "Ocultar/Mostrar", Value = "visibilidad", Icon = "visibility" },
            new ContextMenuItem(){ Text = "Estadísticas", Value = "estadisticas", Icon = "bar_chart" },
            new ContextMenuItem(){ Text = "Cambiar Fecha", Value = "cambiar_fecha", Icon = "event" },
            new ContextMenuItem(){ Text = "Etiquetas", Value = "etiquetas", Icon = "label" },
            new ContextMenuItem(){ Text = "Eliminar", Value = "eliminar", Icon = "delete" },
            new ContextMenuItem(){ Text = "Renombrar", Value = "renombrar", Icon = "edit" }
            },
        item => OnMenuItemClick(item, archivo));
    Task.Delay(300).ContinueWith(_ => clickArchivo = false);
}

void OnMenuItemClick(MenuItemEventArgs args, Archivo archivo)
{
    switch (args.Value)
    {
        case "visibilidad":
            CambiarVisibilidad(archivo);
            Console.WriteLine($"Ocultando/Mostrando archivo: {archivo.NombreArchivo}");
            break;
        case "estadisticas":
            MostrarEstadisticas(archivo);
            Console.WriteLine($"Mostrando estadísticas de: {archivo.NombreArchivo}");
            break;
        case "cambiar_fecha":
            CarmbiarFecha(archivo);
            Console.WriteLine($"Cambiando fecha de: {archivo.NombreArchivo}");
            break;
        case "etiquetas":
            ModificarEtiquetas(archivo);
            Console.WriteLine($"Gestionando etiquetas de: {archivo.NombreArchivo}");
            break;
        case "eliminar":
            EliminarArchivo(archivo);
            Console.WriteLine($"Eliminando archivo: {archivo.NombreArchivo}");
            break;
        case "renombrar":
            RenombrarArchivo(archivo);
            Console.WriteLine($"Renombrando archivo: {archivo.NombreArchivo}");
            break;
    }
    ContextMenuService.Close();
}
```

Figura 32. Menú contextual documentos

Tras hacer clic derecho sobre un documento, se capta el evento se despliega el menú contextual que contiene seis opciones: ocultar/mostrar, ver estadísticas, cambiar fecha (baja), etiquetas, eliminar archivo y renombrar. Cada una de estas opciones permite modificar una característica diferente de los documentos y se detalla en los apartados siguientes.

4.5.1 Opción ocultar/mostrar

Esta opción permite al docente modificar la visibilidad de un documento, facilitando así la posibilidad de subir archivos y decidir cuándo estarán disponibles para los alumnos. Al hacer clic en la opción, se muestra un mensaje de confirmación y, si el usuario acepta, se actualiza el atributo *Visible* del archivo en la base de datos, reflejándose el cambio en la interfaz: si el documento estaba visible, dejará de estarlo, y viceversa.

Para asegurar el acceso a documentos no visibles desde el entorno docente, se ha añadido un botón en la esquina superior derecha de la pantalla de gestión de documentos, que permite alternar entre la visualización de archivos visibles y no visibles, posibilitando su revisión, edición o reactivación.

4.5.2 Opción estadísticas

Dado que los documentos poseen una serie de características de interés se implementa esta funcionalidad para poder acceder a ellas y consultarlas. Esta interfaz no permite realizar acciones sobre el documento seleccionado, limitándose a proporcionar información clave sobre su actividad.

Se muestran el nombre del archivo, las visitas totales que ha tenido ese documento, cuanto tiempo se ha mantenido abierto y por tanto se entiende que un alumno lo ha estado usando, su fecha de subida y la última vez que alguien lo ha visualizado y por último el usuario propietario que es quien ha subido el archivo.

Para mostrar toda esta información del archivo el componente modal recibe como parámetros el identificador único del archivo y su nombre. El nombre se utiliza directamente para su visualización sin necesidad de realizar una consulta adicional a la base de datos, optimizando así el rendimiento.

Una vez recibidos los parámetros, el método *OnParametersSetAsync* se encarga de ejecutar la llamada correspondiente a la base de datos para recuperar los datos estadísticos del archivo mediante el servicio correspondiente. Esta información es

posteriormente representada en la interfaz mediante componentes visuales que en este caso con campos de texto.

4.5.3 Opción Cambiar Fecha

La fecha de baja funciona como un mecanismo de control sobre la vigencia de un documento en la aplicación, permitiendo añadir, modificar o eliminar este dato. Al seleccionar la opción desde el menú contextual, se abre una ventana modal que muestra, si existe, la fecha de baja actual y un componente de calendario para elegir una nueva.

El componente recibe como parámetro el identificador único del archivo, lo que permite recuperar su información actual y aplicar actualizaciones. Una vez seleccionada la nueva fecha, el usuario puede confirmar la acción mediante el botón Modificar y una confirmación adicional, lo que actualiza el campo correspondiente en la base de datos.

Antes de guardar, se valida que la fecha introducida no esté vacía y sea posterior al día actual. Si se pulsa Cancelar, el cuadro de diálogo se cierra sin aplicar cambios.

4.5.4 Opción Etiquetas

Esta opción permite modificar las etiquetas de un documento, ya sea añadiendo o eliminando. Al seleccionarla desde el menú contextual, se abre un modal que muestra el nombre del archivo, un componente *RadzenAutoComplete* para añadir nuevas etiquetas (sugiriendo entre las ya existentes en la base de datos), una lista de etiquetas actuales del documento con opción de eliminarlas, y dos botones: Modificar y Cancelar.

El componente recibe como parámetro el código del archivo seleccionado, que se utiliza para recuperar las etiquetas actuales desde la base de datos y, en caso de cambios, guardar las actualizaciones correctamente.

El funcionamiento se apoya en cuatro listas:

1. Todas las etiquetas de la base de datos, para el autocomplete.
2. Etiquetas actuales del documento, mostradas para posible eliminación.
3. Etiquetas eliminadas por el usuario.
4. Nuevas etiquetas que el usuario desea añadir.

Los cambios no se aplican hasta que el usuario pulsa Modificar y confirma. En ese momento, se llama al servicio de etiquetas: se eliminan las relaciones con las etiquetas quitadas, y se añaden las nuevas. Si alguna nueva etiqueta no existía, primero se crea en la base de datos y luego se asocia al documento.

Si se pulsa Cancelar, el modal se cierra sin guardar cambios y se regresa a la pantalla de documentos.

4.5.5 Opción Eliminar

Esta funcionalidad permite al usuario eliminar de forma definitiva un archivo del sistema, sin importar si tiene una fecha de baja asignada, y está pensada para casos en los que el documento ya no es relevante o debe ser retirado del curso.

Al seleccionar esta opción desde el menú contextual del archivo, se muestra un cuadro de diálogo de confirmación para que el usuario verifique su intención de eliminarlo. Tras confirmar, se ejecuta el proceso mediante una llamada al servicio correspondiente, que incluye la eliminación del documento de la base de datos y de todas sus relaciones con las etiquetas asociadas.

Una vez completado, el documento desaparece tanto de la interfaz de usuario como de la base de datos.

```
public async Task EliminarArchivo(Archivo archivo)
{
    var confirmacion = await DialogService.Confirm("¿Estás seguro de que quieres eliminar este archivo?", "Confirmar eliminación",
        new ConfirmOptions { OkButtonText = "Sí, eliminar", CancelButtonText = "Cancelar" });

    if (confirmacion == true)
    {
        await EtiquetasService.EliminarTodasEtiquetasArchivo(archivo.CodigoArchivo);
        await ArchivoService.EliminarArchivo(archivo.CodigoArchivo);

    }

    await CargarArchivos();
    StateHasChanged();
}
```

Figura 33. método eliminación archivo y sus etiquetas

4.5.6 Opción Renombrar

Por defecto, los archivos se cargan al sistema con su nombre original, pero esta funcionalidad permite modificarlo posteriormente desde el menú contextual de los documentos.

El modal recibe como parámetro el código del archivo seleccionado. Al iniciarse el componente y recibir los parámetros, se recupera la información del archivo para mostrar su nombre actual y permitir su edición.

Si el usuario pulsa Modificar, se valida que el nuevo nombre sea válido y distinto del anterior. Si se cumple esta condición, se envía la actualización a la base de datos mediante el servicio correspondiente. En caso de pulsar Cancelar, se cierra el modal sin aplicar ningún cambio y se regresa a la pantalla de documentos.

4.5.7 Pruebas realizadas

En este sprint se han llevado a cabo las distintas pruebas de funcionamiento de las acciones que puedes realizar con archivo y se ha verificado que:

- La acción ocultar/mostrar funciona en base a lo esperado.
- La opción estadísticas permite consultar los datos de estas de cada archivo.

- La opción cambiar fecha modifica los documentos de estas de manera correcta y se registra correctamente el cambio en base de datos.
- La opción etiquetas permite modificar las etiquetas y todo cambio se ve reflejado en la base datos.
- La opción eliminar borra un archivo, todas sus relaciones con el curso y lo elimina de base de datos.
- La opción renombrar cambia el nombre del archivo correctamente.

4.6 Sprint 5 – Inicio de sesión y registro de nuevos usuarios

En este sprint numero 5 se realizan las tareas correspondientes para lograr los objetivos de la historia de usuario HU07, relacionada con el inicio de sesión y el registro de los usuarios.

4.6.1 Login

Para implementar el inicio de sesión se creó un formulario que solicita correo electrónico y contraseña, con validaciones para asegurar datos válidos. Los valores se vinculan a una instancia de *Usuario.cs* mediante *@bind*.

Al hacer clic en iniciar sesión, se autentica verificando los datos en la base de datos, usando dos servicios: servicio de usuarios (consultas en la base de datos) y servicio de autenticación (autentica, crea la sesión y almacena los datos) [30][31].

Para un almacenamiento seguro de la sesión se emplea *ProtectedSessionStorage* de Blazor Server, que usa cifrado y mecanismos de integridad. Los datos solo duran la sesión, eliminándose al cerrarla manualmente o al cerrar el navegador [32][33].

Las contraseñas se almacenan cifradas con Bcrypt (explicado en punto 3.3.4). Durante la autenticación, la contraseña ingresada se compara con la versión cifrada, garantizando confidencialidad y seguridad.

La página de inicio de sesión no usa *MainLayout.razor* porque aún no se carga el contenido general. Se define un nuevo estilo con fondo y logotipo (diseño explicado más adelante) mediante las clases *login-background* y *login-card*, ajustando tamaño, opacidad y visibilidad, y añadiendo transparencia al formulario.

Para mejorar la experiencia, se añadió un botón con icono de ojo junto al campo contraseña, que alterna su visualización entre texto claro y oculto.

Tras el botón de iniciar sesión hay un enlace que redirige automáticamente a la pantalla de registro si el usuario es nuevo.

4.6.2 Register

El proceso de registro de nuevos usuarios se implementa mediante un formulario en el que se recogen los datos necesarios: nombre, apellidos, correo electrónico y contraseña. Todos los campos a excepción del segundo apellido son obligatorios y este formulario incorpora mecanismos de validación que garantizan la integridad y validez de los datos introducidos antes de proceder con la creación del usuario.

Los datos del formulario están vinculados a una nueva instancia de *Usuario.cs* mediante la propiedad *bind-value*, lo que permite su envío directo al servicio de usuarios correspondiente una vez superada la validación. Dicho servicio se encarga de almacenar el nuevo registro en la base de datos. Tras un registro exitoso, el sistema redirige automáticamente al usuario a la pantalla de inicio de sesión para que pueda autenticarse e ingresar a la aplicación.

Una vez seleccionada esta opción, el servicio de usuarios aplica el algoritmo BCrypt a la contraseña antes de almacenarla.

```
2 referencias
public async Task<bool> RegistrarUsuarioAsync(Usuario usuario)
{
    usuario.Contraseña = HashPassword(usuario.Contraseña);
    return await _usuarioRepository.RegistrarUsuarioAsync(usuario);
}
```

Figura 34. Código registro usuario

```
1 referencia
private static string HashPassword(string password)
{
    return BCrypt.Net.BCrypt.HashPassword(password);
}
```

Figura 35. Cifrado de contraseña BCrypt

Además, al registrar un nuevo usuario, este recibe por defecto el rol de estudiante, rol que podrá ser modificado posteriormente por un administrador.

4.6.3 Menú de perfil

A raíz de la implementación de las interfaces de inicio de sesión y registro, se identificó la necesidad de incorporar un menú de perfil que permitiera al usuario cerrar sesión de forma explícita, considerando que los datos de sesión ya estaban siendo guardados.

El menú de perfil se integró en la interfaz principal de la aplicación, concretamente en la esquina superior derecha. Este componente proporciona una experiencia personalizada mediante un mensaje de bienvenida dirigido al usuario autenticado, así como un menú desplegable que incluye la opción de cerrar sesión.

Para su implementación se empleó el componente *RadzenProfileMenu* al que se le añade el mensaje de bienvenida y un componente hijo que funciona como menú desplegable con la opción de cerrar sesión.

```
<RadzenProfileMenu>
    <Template>
        @if (usuarioActual != null)
        {
            <span>Bienvenido, @usuarioActual.Nombre</span>
        }
    </Template>
    <ChildContent>
        <RadzenProfileMenuItem Text="Cerrar sesión" Path="" Icon="exit_to_app" @onclick="CerrarSesion"></RadzenProfileMenuItem>
    </ChildContent>
</RadzenProfileMenu>
```

Figura 36. ProfileMenu Radzen

4.6.4 Pruebas realizadas

En este sprint se ha creado la funcionalidad del login y el registro, en base a esto, se ha realizado una prueba en la que se verifica que un usuario se puede registrar en la

aplicación sin problema. Una vez registrado se comprueba que puede acceder en la pantalla de login con los datos introducidos previamente, se comprueba que la contraseña está hasheada en base de datos para que ni si quiera los administradores del sistema tengan acceso a la misma. Se comprueba que el menú del perfil permite al usuario cerrar sesión sin problemas.

4.7 Sprint 6 – Carpetas

En este sprint se realizan las tareas para conseguir la organización de los documentos en carpetas (historia de usuario HU08).

Para el desarrollo de esta funcionalidad lo primero que se hizo fue crear la entidad *Carpeta.cs* y la definición del correspondiente DbSet para poder acceder a la tabla Carpeta de base de datos.

A continuación, en *Home.razor*, se añade al método de carga de los documentos de un curso y grupo, un método para obtener las carpetas relacionadas. Para la representación visual de las mismas se utiliza al igual que en el caso de los documentos una *RadzenCard* personalizada para que aparezca como imagen el icono de una carpeta y debajo del mismo el nombre de la carpeta.

```
if (item is Carpeta carpeta)
{
    <RadzenCard class="rz-my-2 card-hover"
                Style="max-width: 420px; box-shadow: none; border: none; margin:0; padding: 0;"
                draggable="true"
                @onclick="() => MostrarArchivosDeCarpeta(carpeta)"
                @oncontextmenu="args => { ShowContextMenuCarpeta(args, carpeta); }"
                @attributes="@(new Dictionary<string, object>() {
                    { "data-carpeta-id", carpeta.CodigoCarpeta },
                    { "ondragover", "onDragOver(event)" },
                    { "ondrop", "onDrop(event)" }
                })">
        <RadzenStack Orientation="Orientation.Vertical" JustifyContent="JustifyContent.Right" AlignItems="AlignItems.Center" Gap="0.5rem" class="rz-p-4">
            <RadzenImage Path="@ObtenerIconoCarpeta()" Style="width: 50px; height: 50px;" />
            <RadzenText TextStyle="TextStyle.Body1">
                <a href="javascript:void(0);" @onclick="() => MostrarArchivosDeCarpeta(carpeta)"
                   style="color: black; text-decoration: none; font-weight: bold;">
                    @carpeta.Descripcion
                </a>
            </RadzenText>
        </RadzenStack>
    </RadzenCard>
}
```

Figura 37. RadzenCard para las carpetas

A cada una de las tarjetas se le asigna un evento *onclick*, cuya función es mostrar el contenido de la carpeta seleccionada. Para ello, se vincula este evento con

el método *MostrarArchivosDeCarpeta*, al cual se le pasa como parámetro la instancia de la *Carpeta* seleccionada. Este método redirecciona al usuario a la pantalla *home.razor* , incluyendo en la URL el código de la carpeta. Esto permite que al renderizarse el componente nuevamente, el método *OnParameterSetAsync* recoja el código de la carpeta y cargue los documentos correspondientes a esa carpeta.

Otro de los eventos asignados a cada una de las tarjetas de las carpetas es el de *oncontextmenu* para permitir al usuario eliminar una carpeta o renombrarla. Al hacer clic derecho sobre una carpeta aparece el menú contextual con las dos opciones.

Al seleccionar la opción de eliminar se muestra un mensaje para que el usuario confirme la eliminación y en caso de que acepte se elimina la carpeta en base de datos. En caso de renombrar, se abre un modal para que el usuario introduzca el nuevo nombre y tras confirmar se modifica la carpeta en base de datos.

4.7.1 Crear y subir carpetas

Se ha desarrollado también la funcionalidad de crear una nueva carpeta vacía o subir una carpeta del pc del usuario con los documentos que contenga. Para ello se creó un menú contextual general en la pantalla de inicio de los documentos. Cuando se hace clic derecho en la pantalla sin seleccionar ningún documento o carpeta se despliega este nuevo menú contextual con tres opciones: crear carpeta, subir carpeta y subir documento. Esta última opción es el mismo funcionamiento que el explicado en el apartado 4.6.3 de añadir documentos.

Tanto para la opción de subir carpeta como la de crearla, se usa el mismo componente: *SubirCarpeta.razor*, con la diferencia de que como parámetro se envían: el código del curso en el que se encuentra el usuario, el código del usuario que realiza el cambio, la descripción del curso y un código de opción. Este último parámetro indica si el usuario va a crear una nueva carpeta o si por el contrario la va a subir.

La opción vinculada a crear una carpeta es el código de opción '1', cuando se selecciona la opción de crear carpeta se abre el modal con dos campos para que el usuario indique el nombre de la nueva carpeta y el grupo al que va a pertenecer. Por

defecto, el grupo aparece ya indicado según el curso y grupo seleccionado previamente. Aunque, este campo está configurado como un *RadzenDropDown* para poder seleccionar entre los grupos del curso ya existentes.

Si el usuario confirma la creación de esta nueva carpeta, se completan los datos necesarios de la instancia de la nueva carpeta y se hace una llamada al servicio encargado de la gestión de las carpetas para realizar la grabación en la base de datos. En caso de no confirmar, se cierra la pantalla de subida y se vuelve a la pantalla principal.

Además, se realiza una validación de los datos introducidos previo a confirmar los cambios. Se comprueba que se ha introducido un nombre de carpeta válido.

Por consiguiente, la opción para subir una carpeta es el código '2'. El funcionamiento es similar al anterior, se selecciona la opción de subir carpeta y se abre una ventana con dos campos: seleccionar carpeta y grupo.

Para la funcionalidad de seleccionar una carpeta del ordenador del usuario, el campo de seleccionar carpeta se crea con un botón que al ser presionado invoca al método *AbrirSelectorCarpeta*, que a su vez invoca una nueva función de JavaScript y crea una nueva instancia de *Carpeta.cs*, guardando el nombre de la nueva carpeta seleccionada y la creación de tantas instancias *Archivo.cs* como documentos tenga esa carpeta seleccionada. Aparece visualmente indicado el nombre de la carpeta seleccionada.

```
<div>
    <label>Selecciona una carpeta:</label>
    <RadzenButton Text="Seleccionar Carpeta" Click="AbrirSelectorCarpeta" />
    <p>@carpetaNueva.Descripcion</p>
</div>
```

Figura 38. Selección carpeta

```
private async Task AbrirSelectorCarpeta()
{
    var result = await JS.InvokeAsync<FolderResult>("abrirSelectorCarpeta");

    if (result != null)
    {
        carpetaNueva.Descripcion = result.NombreCarpeta;
        archivos = result.Archivos.Select(a => new Archivo
            {
                NombreArchivo = a.Nombre,
                Tipo = a.Tipo,
                Contenido = a.Contenido.ToArray(),
                Tamaño = a.Tamaño,
                FechaAlta = DateTime.Now,
                Visible = true
            }).ToList();
    }
}
```

Figura 39. Método AbrirSelectorCarpeta

En la función de JavaScript, para permitir la selección de una carpeta local por parte del usuario desde el navegador, se utiliza la API *showDirectoryPicker*, disponible en entornos que admiten capacidades del File System Access API.

```
window.abrirSelectorCarpeta = async () => {
    try {
        const handle = await window.showDirectoryPicker();
        const nombreCarpeta = handle.name;
        const archivos = [];

        for await (const entry of handle.values()) {
            if (entry.kind === "file") {
                const file = await entry.getFile();
                const contenido = await file.arrayBuffer();
                archivos.push({
                    nombre: file.name,
                    tipo: file.type,
                    contenido: Array.from(new Uint8Array(contenido)),
                    tamaño: file.size
                });
            }
        }

        console.log("Carpeta seleccionada:", nombreCarpeta);
        console.log("Archivos encontrados:", archivos.length);
        return { nombreCarpeta, archivos };

    } catch (error) {
        console.error("Error al seleccionar la carpeta:", error);
        return null;
    }
};
```

Figura 40. Script seleccionar carpeta

Esta función permite al usuario seleccionar una carpeta, acceder a su contenido (archivos) y extrae información relevante de cada uno, como el nombre, tipo, tamaño y contenido binario en forma de arreglo de bytes, con los que se crear las instancias que se pueden ver en la Figura 40. La función retorna un objeto que contiene el nombre de la carpeta y una lista con los archivos procesados.

Una vez seleccionada la carpeta, el usuario puede cambiar el grupo, aunque por defecto sale el grupo que se ha seleccionado dentro del curso.

Finalmente, el usuario puede optar por subir la carpeta (clic en botón guardar) o cancelar la subida. Si la decisión es guardar, aparece un mensaje que el usuario debe confirmar antes de subir la carpeta en base de datos. Si se confirma, se completa la instancia de la nueva carpeta y se llama al servicio de carpetas para poder guardar en la base de datos, tanto la información relativa a la carpeta como la de todos sus archivos.

En este caso se hace una validación para comprobar que el usuario ha seleccionado correctamente una carpeta.

```
private async Task ConfirmarSubida()
{
    var confirmacion = await DialogService.Confirm("¿Quiere subir la carpeta?", "Confirmar subir carpeta",
     new ConfirmOptions { OkButtonText = "Sí", CancelButtonText = "Cancelar" });

    if (confirmacion == true)
    {
        if (CodigoOpcion == 1)
        {
            carpetaNueva.Path = DescripcionCurso + "/" + carpetaNueva.Descripcion;
            carpetaNueva.Subida = DateTime.Now;
            carpetaNueva.Propietario = CodigoUsuario;
            carpetaNueva.Curso = CodigoCurso;
            carpetaNueva.Grupo = Grupo;
            await CarpetaService.CrearCarpeta(carpetaNueva);

            DialogService.Close(true);
        }
        else
        {
            carpetaNueva.Path = DescripcionCurso + "/" + carpetaNueva.Descripcion;
            carpetaNueva.Subida = DateTime.Now;
            carpetaNueva.Propietario = CodigoUsuario;
            carpetaNueva.Curso = CodigoCurso;
            carpetaNueva.Grupo = Grupo;
            int codigoCarpeta = await CarpetaService.CrearCarpeta(carpetaNueva);


            foreach (Archivo archivo in archivos)
            {
                archivo.Curso = CodigoCurso;
                archivo.Propietario = CodigoUsuario;
                archivo.CodigoCarpeta = codigoCarpeta;
                archivo.Grupo = carpetaNueva.Grupo;
            }
            await ArchivoService.GuardarListaArchivoAsync(archivos);
            DialogService.Close(true);
        }
    }

    DialogService.Close(false);
}
```

Figura 41. Método guardar carpeta

4.7.2 Drag & drop documentos a carpetas

Durante el desarrollo de esta funcionalidad se vio que podría ser interesante añadir la funcionalidad de arrastrar y soltar (*drag&drop*) en los documentos y las carpetas de la pantalla de inicio. Con el objetivo de mejorar y facilitar la tarea de organización de los documentos y que sea sencillo añadir un documento ya subido a una carpeta.

Para llevar a cabo esta funcionalidad se configuran las tarjetas Radzen de carpetas y documentos indicando que los elementos sean arrastrables (draggable=true) y se añaden atributos personalizados como data-carpeta-id y data-archivo-id, que permiten identificar que documento y que carpeta están interactuando.

Estos atributos se combinan con el uso de eventos JavaScript definidos externamente, como *ondragstart*, *ondragover* y *ondrop*, que se asignan dinámicamente mediante el diccionario *@attributes*. Esto permite conectar la lógica de interfaz declarada en Blazor con funciones de JavaScript que gestionan el proceso de arrastrar un archivo y soltarlo sobre una carpeta.

El funcionamiento es el siguiente: al iniciar el arrastre de un documento (*ondragstart*), se registra su identificador. Posteriormente, al posicionarse sobre una carpeta (*ondragover*), se permite la operación de soltado. Finalmente, al soltar el documento (*ondrop*), se obtiene el identificador de la carpeta destino y se invoca de forma asíncrona al método *MoverArchivoACarpeta* mediante la función *invokeMethodAsync*, que permite establecer un puente entre el entorno de JavaScript del cliente y el backend gestionado por Blazor.

```javascript
window.registrarObjetoDotNet = function (dotNetHelper) {
    window.miObjetoDotNet = dotNetHelper;
}

let archivoArrastradoId = null;

0 referencias
function onDragStart(event) {
    const archivoId = event.target.getAttribute('data-archivo-id');
    console.log("data-archivo-id:", archivoId);  // Verifica el valor aquí
    archivoArrastradoId = archivoId;
    console.log("hecho el ondragstart");
}


0 referencias
function onDragOver(event) {
    event.preventDefault(); // Necesario para permitir el drop
    console.log("hecho el ondragover");
}

0 referencias
function onDrop(event) {
    event.preventDefault();
    const carpetaId = event.currentTarget.getAttribute('data-carpeta-id');
    console.log("estamos en el ondrop");
    console.log("archivoArrastradoId:", archivoArrastradoId);
    console.log("carpetaId:", carpetaId);
    console.log("miObjetoDotNet:", window.miObjetoDotNet);
    if (archivoArrastradoId && carpetaId && window.miObjetoDotNet) {
        console.log("Vamos a llamar a la funcion moverarchivoacarpeta");
        window.miObjetoDotNet.invokeMethodAsync('MoverArchivoACarpeta', parseInt(archivoArrastradoId), parseInt(carpetaId))
            .then(() => {
                console.log('Archivo movido correctamente');
            })
            .catch(err => console.error('Error al invocar método .NET', err));
    }
}
```

Figura 42. Script drag&drop

Dicha función se declara en la página *Home.razor* y se marca como [JSInvokable] que es imprescindible para indicar que el método está disponible para poder ser accedido desde el código JavaScript.

El método *MoverArchivoACarpeta* implementa la lógica necesaria para mover un documento a una carpeta. En primer lugar, se pide confirmación al usuario, si el usuario acepta, se accede a la información del archivo y se modifica el campo relacionado con la carpeta, actualizándolo con el valor del código de esta. Finalmente, se recargan los archivos y se solicita la actualización de la interfaz de usuario para reflejar los cambios realizados.

4.7.3 Menú contextual carpetas

La gestión de carpetas permite renombrar o eliminar una carpeta mediante opciones disponibles en un menú contextual, que aparece tras hacer clic derecho sobre una carpeta.

Al seleccionar la opción de renombrar, se abre un modal que recibe el código de la carpeta como parámetro. El componente, al inicializarse, obtiene como parámetro el código de la carpeta seleccionada y carga su información de base de datos. El nuevo nombre se valida antes de enviarlo a la base de datos, comprobando que no esté vacío y que sea distinto al anterior. La operación se realiza mediante el servicio de carpetas que actualiza los datos en la base de datos.

Para la opción de eliminar, se muestra un mensaje de confirmación al usuario. Si se acepta, se invoca un servicio que elimina la carpeta y, automáticamente, todos los documentos asociados, gracias a la relación definida entre las tablas Carpetas y Archivos en la base de datos.

Ambas acciones pueden cancelarse sin realizar cambios.

4.7.4 Pruebas realizadas

En este sprint se han incluido las carpetas a la aplicación, se ha comprobado que se puede crear una carpeta de manera satisfactoria y esta se encuentra asociada al curso en el que se ha creado. También se comprueba que la aplicación permite realizar el drag & drop de archivos a carpetas de manera satisfactoria. Por último, se ha comprobado que la funcionalidad renombrar carpeta, realiza la acción de manera correcta y que la funcionalidad eliminar carpeta no solo elimina la carpeta, si no que elimina todos los documentos que existían en la misma.

4.8 Sprint 7 – Búsqueda y filtrado de documentos

En este sprint se realizan las tareas para conseguir la funcionalidad para la búsqueda de documentos (historia de usuario HU09) también comprende las funcionalidades del sprint 10, denominado filtros de documentos (Historia HU10).

En primer lugar, se ha creado la clase *navbusquedafiltros.razor*, esta clase es la encargada de dibujar el componente que aparece en la parte derecha de la pantalla, destacar que para abrir dicho componente ha sido necesario en la pantalla home añadir un botón que muestra el desplegable de la barra de búsqueda.

En cuanto al diseño en sí de la pantalla, se ha añadido en primer lugar un cuadro de texto para introducir la palabra clave por la que se quiere buscar, además de añaden dos botones, el botón de buscar que, como su nombre indica, busca archivos que contengan la palabra insertada. Además, se ha incluido un botón de limpiar búsqueda encargado de vaciar el cuadro de texto.

Justo debajo se añade un combo que permite ordenar los ficheros de un curso y grupo concretos, hay dos tipos de ordenación seleccionables, por fecha de subida más reciente y por documentos más vistos.

También existe un combo que permite seleccionar las etiquetas existentes y filtrar los documentos que poseen dichas etiquetas una vez presionado el botón buscar. Para conseguir esta funcionalidad se ha tenido que crear un nuevo método en el servicio

'EtiquetaService', esta llama al repositorio devolviendo todas las etiquetas que hay en la base de datos con relación a los distintos archivos subidos a la aplicación.

Por último, existe un botón denominado 'Limpiar filtros' que permite reiniciar todos los valores introducidos previamente, además del botón filtrar encargado de mostrar los resultados en base a todos los filtros que se han seleccionado.

Esta funcionalidad se lleva a cabo mediante la siguiente función de la Figura 43.

```
private void Filtrar()
{
    if (ordenacion == "recientes")
    {
        filtroRecientes = true;
        filtroMasVistos = false;
    }
    else if (ordenacion == "masVistos")
    {
        filtroMasVistos = true;
        filtroRecientes = false;
    }

    var query = $"?recientes={filtroRecientes}&masVistos={filtroMasVistos}";

    if (selectedTags != null && selectedTags.Any())
    {
        query += "&tags=" + Uri.EscapeDataString(string.Join(",", selectedTags));
    }
    if(CodigoCarpeta == 0){
        Navigation.NavigateTo($"/home/{CodigoCurso}{query}", forceLoad: true);
    }else{
        Navigation.NavigateTo($"/home/{CodigoCurso}/{CodigoCarpeta}{query}", forceLoad: true);
    }
```

Figura 43. Función filtrar

Esta realiza una query dependiendo de los filtros de ordenación seleccionados, además de las distintas etiquetas existentes, posteriormente se fuerza a que se recargue la pantalla home con los parámetros indicados, mostrando únicamente los archivos que se relacionen con dicha query.

4.8.1 Pruebas realizadas

En este sprint se ha llevado a cabo la funcionalidad de búsqueda y filtrado de documentos, se ha comprobado que las características existentes de filtrado de documentos funcionan en base a lo esperado y que una vez pulsado el botón se realiza la búsqueda mostrando los documentos y/o carpetas que coinciden con los datos introducidos

4.9 Sprint 8 – Retención documental y notificaciones

En el sprint número 8 se llevó a cabo la funcionalidad relacionada con la retención automática de los documentos (historia de usuario HU11) y la de notificaciones automáticas (HU13).

4.9.1 Retención documental

La retención documental se implementa mediante un servicio que se ejecuta en segundo plano automáticamente al ejecutarse la aplicación. El servicio creado se denomina *RetencionDocumentalService.cs* y se basa en un temporizador *timer* que ejecuta su lógica inmediatamente al ejecutar la aplicación y luego cada 24 horas. De esta manera se comprueba continuamente si hay documentos que tienen que ser dados de baja.

Como es un servicio que se ejecuta en segundo plano se utiliza la interfaz *IServiceScopeFactory* que permite crear un ámbito de servicios (scope) y así poder acceder a los repositorios de datos necesarios, en este caso, archivos, etiquetas y estadísticas.

Durante cada ejecución del servicio se realizan dos acciones:

- Consultar todos los archivos y determinar si hay expirados.
- Por cada uno que haya expirado, eliminarlo de la base de datos, así como, su relación con las etiquetas y su estadística asociada.

4.9.2 Notificaciones automáticas

Cuando se sube un nuevo documento y se guarda en la base de datos, se envía un correo electrónico a todos los estudiantes que pertenezcan a ese curso y grupo.

Para ello primero se obtienen de base datos la lista de los usuarios a los que hay que avisar y mediante el servicio *EmailService.cs* se envía un correo electrónico. El mensaje enviado avisa al usuario que se ha subido un nuevo archivo al curso.

```
public async Task EnviarCorreoAsync(string destinatario, string asunto, string cuerpo)
{
    string apiToken = _configuration["Postmark:ApiToken"];
    string from = _configuration["Postmark:From"];
    string fromName = _configuration["Postmark:FromName"];

    var client = new PostmarkClient(apiToken);

    var message = new PostmarkMessage
    {
        From = $"{fromName} <{from}>",
        To = destinatario,
        Subject = asunto,
        TextBody = cuerpo,
        HtmlBody = $"<p>{cuerpo}</p>"
    };

    try
    {

        var sendResult = await client.SendMessageAsync(message);
        if (sendResult.Status != PostmarkStatus.Success)
        {
            //throw new Exception($"Error al enviar correo: {sendResult.Message}. Código de estado: {sendResult.Status}");
            Console.WriteLine($"Error al enviar correo: {sendResult.Message}. Código de estado: {sendResult.Status}");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error al enviar correo:usuario invalido");
    }


}
```

Figura 44. Envío email

4.9.3 Pruebas realizadas

En este sprint se han llevado a cabo las funcionalidades de retención documental y de envío de notificaciones, en cuanto a la retención documental se ha comprobado que cuando un archivo expira se da de baja de manera correcta y se eliminan todos los datos en relación con este que pudieran existir.

Con relación a las notificaciones, se comprueba que estas se envían por correo de manera satisfactoria a todos los estudiantes que se encuentren dentro del curso y grupo en el que se ha subido un nuevo archivo.

4.10 Sprint 9 – Gestión de grupos y roles

En el sprint número 9 se llevó a cabo un cambio en cómo se gestionan los grupos y se añadieron funcionalidades pendientes respecto a ellos.

En primer lugar, se añadió un combo de selección de grupo a la hora de registrar a nuevos usuarios, esto se realizó para que los alumnos puedan visualizar todos los archivos de los distintos cursos a los que pertenece dicho grupo.

Fue necesario añadir un nuevo campo a la tabla de CursosUsuarios, dicha tabla ahora posee un campo grupo en el que se agrega la relación de dichos usuarios también con el curso correspondiente, este campo es necesario entre otras cosas para, a la hora de generar un nuevo grupo en un curso existente, agregar a todos los usuarios pertenecientes al grupo al curso en el que se ha agregado dicho grupo.

Esto también ha derivado en una modificación de la estructura de la tabla de grupo, antes tenía dos primary key, ahora la variable que contiene el nombre del grupo denominada 'grupo' ya no puede formar parte de la clave primaria ya que van a existir grupos con el mismo nombre asociados a distintos cursos con el objetivo, mencionado anteriormente de que los usuarios puedan visualizar todos los archivos asociados a su grupo.

Por otro lado, estos cambios se han visto impulsados por la implementación de la vista de ciertos factores dependiendo del rol, esto no modifica el funcionamiento original, simplemente hace más sencillo el tratamiento de datos a la hora de ver la aplicación desde la perspectiva del usuario u otra diferente. Se han limitado varias acciones que pueden hacer los alumnos, por ejemplo, estos no pueden crear un grupo ni un curso, esta labor será exclusiva de profesores y/o administradores del sistema, además tampoco tendrán la posibilidad de subir un fichero, ya que esta es otra funcionalidad exclusiva del profesor correspondiente. Se han realizado otros cambios menores orientados al rol, como mostrar ciertas opciones de menú y mostrar solo los grupos y cursos con registro en la tabla CursosUsuarios.

4.11 Adición de log de registro de acciones

La última funcionalidad añadida se trata del log de registro de acciones, este log guarda acciones tanto a nivel individual como a nivel conjunto. Esto se lleva a cabo mediante la inclusión de las nuevas clases *LogService* y *LogRepository*, al igual que las respectivas clases auxiliares como *LogTargetType*, *TipoAccion* y las entidades *UserLog* y *UserLogAudit*. Estas guardan datos de donde accede el usuario y la hora a que lo hace entre otras cosas como se puede ver en la Figura 45 y la Figura 46.

```
2 referencias
public async Task LogAsync(int userId, bool isDocumento, int targetId, TipoAccion action)
{
    using var context = _contextFactory.CreateDbContext();

    var targetType = isDocumento ? LogTargetType.Documento : LogTargetType.Carpeta;

    // 1) Calcular peso base
    double weight = _settings.DefaultWeight;
    if (_settings.Weights.TryGetValue(action.ToString(), out var w))
        weight = w;

    // 2) Aplicar multiplicador por tipo
    weight *= isDocumento
        ? _settings.DocumentoMultiplier
        : _settings.CarpetaMultiplier;

    // 3) Aplicar multiplicador por rol de usuario
    var roleFactor = _settings.RoleMultipliers
        .GetValueOrDefault(userId, 1.0);
    weight *= roleFactor;

    // 4) Grabar auditoria individual
    var audit = new UserLogAudit
    {
        UserId = userId,
        TargetType = targetType,
        TargetId = targetId,
        ActionType = action,
        Weight = weight,
        OccurredAt = DateTime.UtcNow
    };
    context.UserLogAudits.Add(audit);

    // 5) Actualizar o crear registro acumulado
    var log = await context.UserLogs
        .FirstOrDefaultAsync(l =>
            l.UserId == userId &&
            l.TargetType == targetType &&
            l.TargetId == targetId);
```

Figura 45. Log

```
        // 5) Actualizar o crear registro acumulado
        var log = await context.UserLogs
            .FirstOrDefaultAsync(l =>
                l.UserId == userId &&
                l.TargetType == targetType &&
                l.TargetId == targetId);

        if (log == null)
        {
            log = new UserLog
            {
                UserId = userId,
                TargetType = targetType,
                TargetId = targetId,
                Count = 1,
                TotalWeight = weight,
                LastUpdated = DateTime.UtcNow
            };
            context.UserLogs.Add(log);
        }
        else
        {
            log.Count++;
            log.TotalWeight += weight;
            log.LastUpdated = DateTime.UtcNow;
            context.UserLogs.Update(log);
        }

        await context.SaveChangesAsync();
    }
```

Figura 46. Log II

Se puede apreciar cómo se realiza el cálculo del peso en relación al peso base y a si se trata de carpeta y/o documento, a su vez se le asigna un peso en base al tipo de usuario que abre cada uno de los documentos y/o carpetas. Una vez calculado se guardan todos los datos relativos a la auditoría individual, estos comprenden, el id de usuario, el tipo de elemento abierto, el id de dicho elemento, el tipo de acción, el peso y el momento que sucedió. Posteriormente se agrega al registro acumulado de cada uno de los usuarios para llevar el total de todas las acciones que estos realizan y se guarda.

Para poder visualizar los datos se ha creado un nuevo botón y tres vistas que devuelven los datos, esto se muestra en la clase *reportes.razor*. En la Figura 47 se muestran las funciones en las que se muestran mediante gráficos las top 10 carpetas y top 10 documentos consultados.

```
@if (!loaded)
{
    <p>Cargando informes…</p>
}
else
{
    <h4>Top 10 Documentos</h4>

    @if (topDocs.Any())
    {
        <RadzenChart @ref="chartDocs" Style="height:350px">
            <RadzenCategoryAxis />
            <RadzenValueAxis />
            <RadzenBarSeries Data="@topDocs"
                             CategoryProperty="Nombre"
                             ValueProperty="TotalPeso"
                             Title="Peso" />
        </RadzenChart>
    }
    else
    {
        <p>No hay datos para documentos.</p>
    }

    <hr />

    <h4>Top 10 Carpetas</h4>
    @if (topCars.Any())
    {
        <RadzenChart @ref="chartCars" Style="height:350px">
            <RadzenCategoryAxis />
            <RadzenValueAxis />
            <RadzenBarSeries Data="@topCars"
                             CategoryProperty="Nombre"
                             ValueProperty="TotalPeso"
                             Title="Peso" />
        </RadzenChart>
    }
    else
    {
        <p>No hay datos para carpetas.</p>
    }
```

Figura 47. Botón log

Por último, se muestra un mapa de calor de cada día en base al uso de los documentos, el código relacionado se puede ver en la Figura 48.

```
<h4>Heat-map Diario</h4>
@if (usage.Any())
{
    <RadzenDataGrid Data="@usage"
                    TItem="UsageDailyDto"
                    RowRender="@ColorRow"
                    Style="height:350px; overflow:auto">
        <Columns>
            <RadzenDataGridColumn TItem="UsageDailyDto" Property="UsageDate" Title="Fecha" />
            <RadzenDataGridColumn TItem="UsageDailyDto" Property="TargetType" Title="Tipo" />
            <RadzenDataGridColumn TItem="UsageDailyDto" Property="TargetId" Title="ID" />
            <RadzenDataGridColumn TItem="UsageDailyDto" Property="NumEventos" Title="Eventos" />
            <RadzenDataGridColumn TItem="UsageDailyDto" Property="TotalPeso" Title="Peso" />
        </Columns>
    </RadzenDataGrid>
}
else
{
    <p>No hay datos de uso diario.</p>
}
```

Figura 48. HeatMap

Para obtener dichos datos se usan las clases *reportingservice* y *reportingrepository*, existen 3 vistas, para cada uno de los gráficos mostrados en pantalla.

```
2 referencias
public async Task<List<TopItemDto>> GetTop10CarpetasAsync()
{
    using var context = _contextFactory.CreateDbContext();

    return await context
    .Set<UserLogAudit>()                                // UserLogAudits
    .Where(a => a.TargetType == LogTargetType.Documento)
    .GroupBy(a => a.TargetId)
    .Select(g => new TopItemDto
    {
        TargetId = g.Key,
        NumEventos = g.Count(),
        TotalPeso = g.Sum(x => x.Weight),
        Nombre = context.Archivos
                        .Where(x => x.CodigoArchivo == g.Key)
                        .Select(x => x.NombreArchivo)
                        .FirstOrDefault() ?? ""
    })
    .OrderByDescending(x => x.TotalPeso)
    .Take(10)
    .ToListAsync();
}

2 referencias
public async Task<List<TopItemDto>> GetTop10DocumentosAsync()
{
    using var context = _contextFactory.CreateDbContext();

    return await context
    .Set<UserLogAudit>()
    .Where(a => a.TargetType == LogTargetType.Carpeta)
    .GroupBy(a => a.TargetId)
    .Select(g => new TopItemDto
    {
        TargetId = g.Key,
        NumEventos = g.Count(),
        TotalPeso = g.Sum(x => x.Weight),
        Nombre = context.Carpeta
                        .Where(c => c.CodigoCarpeta == g.Key)
                        .Select(c => c.Descripcion)
                        .FirstOrDefault() ?? ""
    })
    .OrderByDescending(x => x.TotalPeso)
    .Take(10)
    .ToListAsync();
```

Figura 49. Vistas log

```
2 referencias
public async Task<List<UsageDailyDto>> GetUsageDailyAsync(DateTime fromDate, DateTime toDate)
{
    using var context = _contextFactory.CreateDbContext();

    return await context
    .Set<UserLogAudit>()
    .Where(a => a.OccurredAt.Date >= fromDate.Date
             && a.OccurredAt.Date <= toDate.Date)
    .GroupBy(a => new { a.TargetType, a.TargetId, Fecha = a.OccurredAt.Date })
    .Select(g => new UsageDailyDto
    {
        TargetType = Convert.ToInt32(g.Key.TargetType),
        TargetId = g.Key.TargetId,
        UsageDate = g.Key.Fecha,
        NumEventos = g.Count(),
        TotalPeso = g.Sum(x => x.Weight)
    })
    .OrderBy(x => x.UsageDate)
    .ThenBy(x => x.TargetType)
    .ThenBy(x => x.TargetId)
    .ToListAsync();
}
```

Figura 50. Vistas log II

Capítulo - 5 Aplicación web GestorDocumental

En este nuevo capítulo se describe el software desarrollado desde la perspectiva de los usuarios destinatarios: profesores y alumnos. La funcionalidad asociada a cada rol se describe con la correspondiente pantalla y una breve descripción.

También se menciona en el punto 5.5 el rol del administrador.

5.1 Acceso a la aplicación

Login: al ejecutar la aplicación aparece la página de inicio de sesión en la que el usuario introduce sus credenciales para poder acceder.

Figura 51. Pantalla login

Registro: en caso de que el usuario no esté registrado, puede pulsar el enlace *Regístrate aquí* que le redirigirá a la pantalla de registro.

Figura 52. Pantalla registro

5.2 El docente y el GestorDocumental

En esta sección se van a presentar las tareas que puede ejecutar el rol del docente. Todas ellas están accesibles desde la pantalla de inicio mostrada en la siguiente Figura 53, a la que accede tras autenticarse; y/o la pantalla de la Figura 54 dentro de un curso o grupo.

Figura 53. Pantalla de inicio

5.2.1 Gestión de cursos

Una vez autenticado, el docente accede a la pantalla principal en la que, por defecto, aparecen los cursos en el margen izquierdo y un mensaje de bienvenida en el margen derecho, indicando que tiene que seleccionar un curso para ver los documentos.

Navegación: si el docente selecciona un curso puede ver los grupos asociados al curso.

El docente puede desplegar cada curso para ver los grupos que tiene cada uno. Si se selecciona un curso en general sin seleccionar un grupo, se muestran los documentos de todos los grupos (Figura 54).

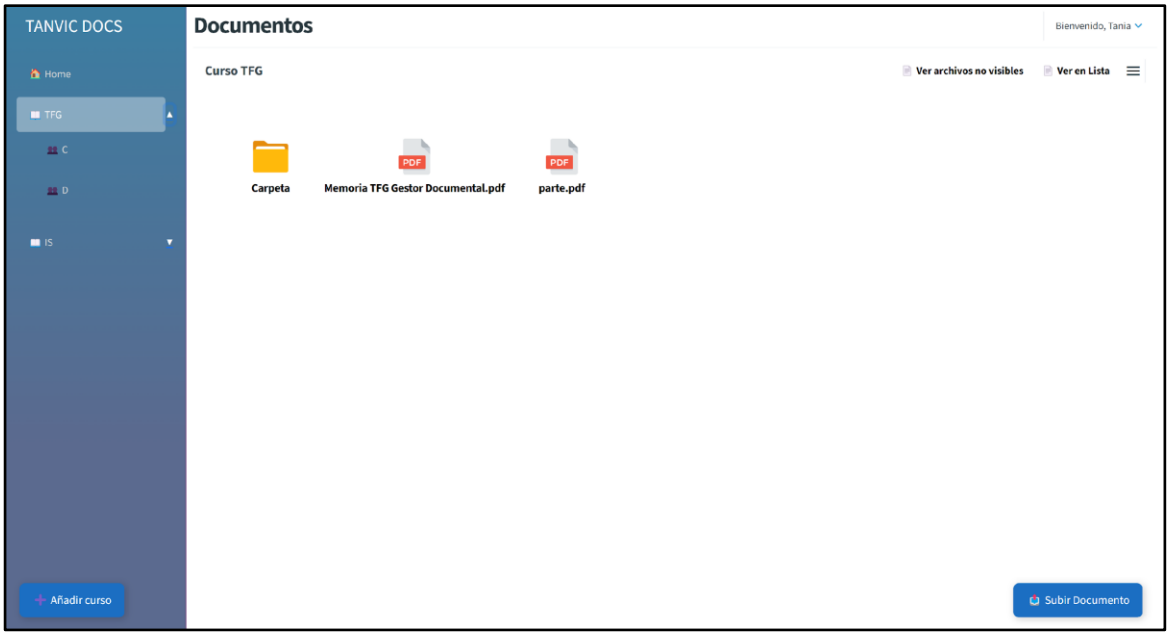

Figura 54. Pantalla inicio curso

Si dentro de un curso se selecciona un grupo, salen únicamente los documentos de ese grupo (Figura 55).

Figura 55. Pantalla inicio grupo de un curso

Crear nuevo curso: desde la pantalla de inicio, hacer clic en el botón *Añadir Curso*. Introducir los datos que se solicitan en la siguiente pantalla y guardar. Se puede abortar la creación con el botón cancelar.

Figura 56. Pantalla creación curso

Añadir un nuevo grupo a un curso: desde la pantalla de inicio, hacer clic derecho sobre el curso y seleccionar la opción de agregar grupo del menú contextual.

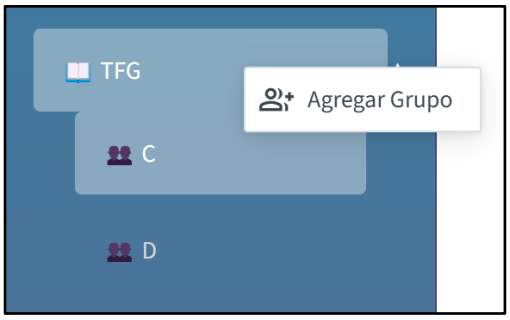

Figura 57. Menú contextual grupos

Una vez se ha seleccionado esa opción se abre una nueva pantalla para indicar el nuevo grupo a crear. Nuevamente hay dos opciones, guardar (se crea el grupo) y cancelar (no se realiza ningún cambio).

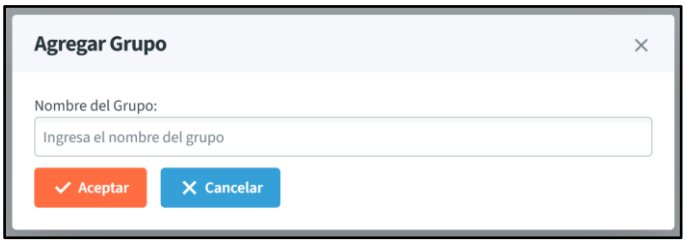

Figura 58. Crear grupo en un curso

5.2.2 Gestión de documentos

Esta gestión incluye añadir nuevos documentos a un curso y sus características; seleccionar modos de visualización de documentos; visualizar contenido; ocultar o mostrar los documentos; eliminar; renombrar; cambiar la fecha de baja automática del documento y mostrar estadísticas.

Subir documento: desde la pantalla inicio del curso (Figura 54) o la pantalla de inicio del grupo de un curso (Figura 55); hacer clic en el botón de la esquina inferior derecha (subir documento). Seleccionar el documento que se desea subir y rellenar las características correspondientes y guardar. Se puede cancelar la subida con el botón de cancelar. El procedimiento es el mismo tanto dentro del curso como dentro de un grupo específico.

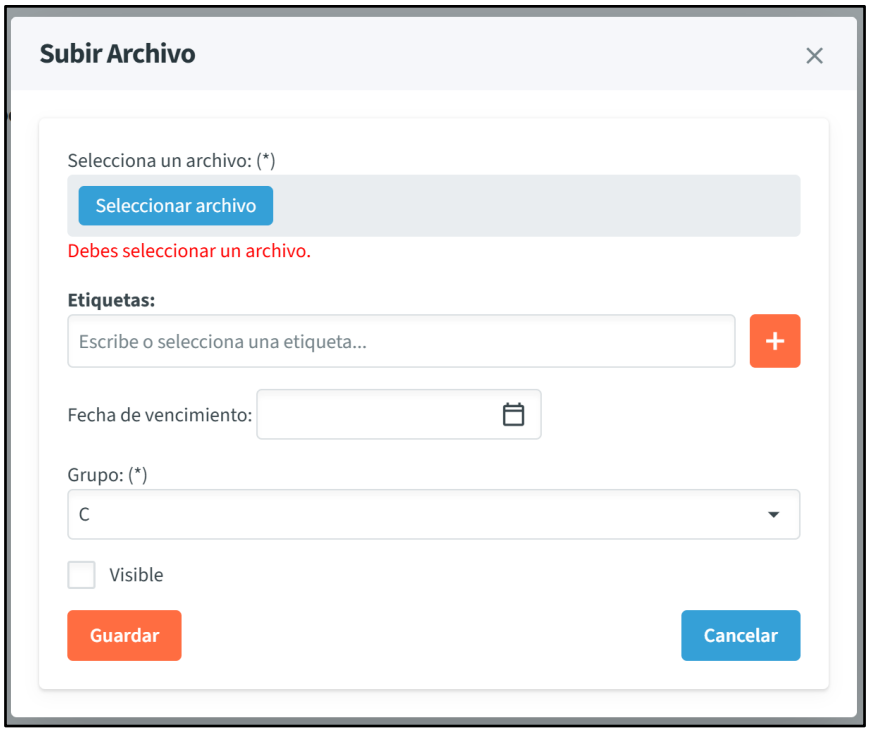

Figura 59. Model subida de un documento

La sección de etiquetas ofrece sugerencias a tiempo real de etiquetas ya existentes, según el docente va escribiendo las etiquetas que quiere añadir.

Figura 60. Sugerencias añadir etiquetas

Se puede eliminar una etiqueta añadida, presionando en la "x" que se puede ver en la siguiente Figura 61.

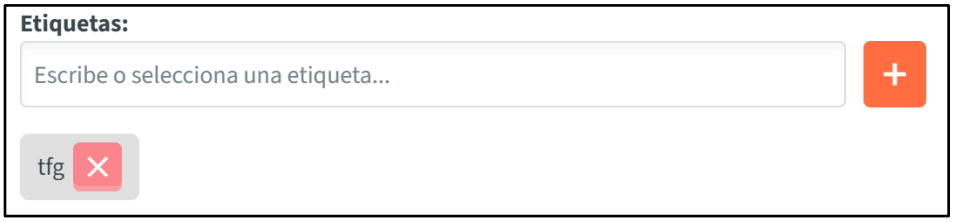

Figura 61. Eliminar etiquetas añadidas

Para la selección de la fecha de baja se hace directamente en el calendario que aparece tras presionar el símbolo.

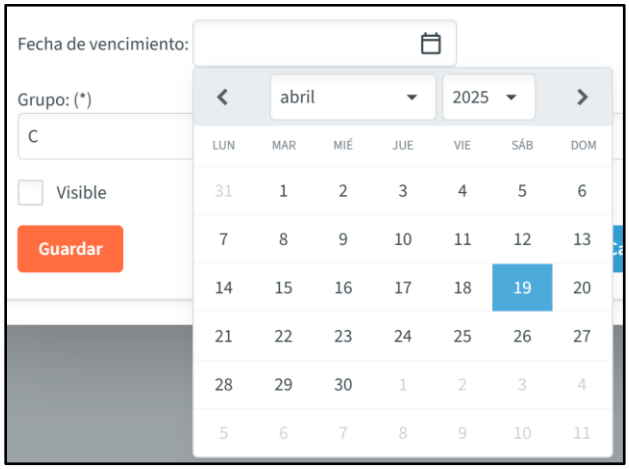

Figura 62. Calendario fecha de baja

En la selección del grupo, aparece por defecto el grupo en el que se está trabajando, pero se puede modificar por otro de los grupos que tenga el curso.

Figura 63. Selección del grupo

Seleccionar modo de visualización de los documentos: en la pantalla de inicio de curso/grupo hacer clic sobre el botón *Ver en lista.* Por defecto la visualización de los documentos es con iconos.

Figura 64. Modos visualización de documentos

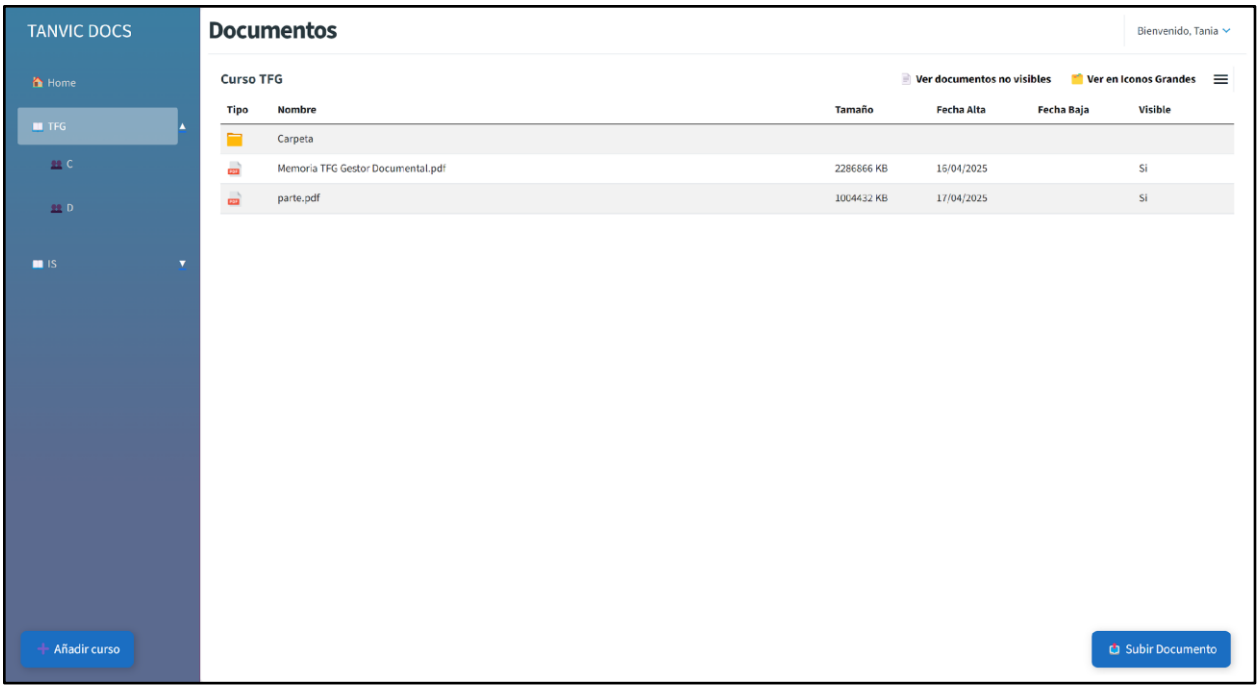

Figura 65. Visualización modo lista de los documentos

Visualizar el contenido de un documento: desde la pantalla de inicio de curso/grupo hacer doble clic sobre el documento.

En caso de los tipos de archivos PDF e imágenes se abrirá una nueva ventana en el navegador. Mientras que, para el resto de tipo de documentos, como el Word, se descargará en el ordenador del usuario.

Ocultar/Mostrar: desde la pantalla de inicio de curso/grupo de documentos hacer clic derecho sobre un documento y seleccionar la opción Ocultar/Mostrar del menú contextual de los documentos (Figura 66) y confirmar el cambio (Figura 67).

Figura 66. Menú contextual documentos

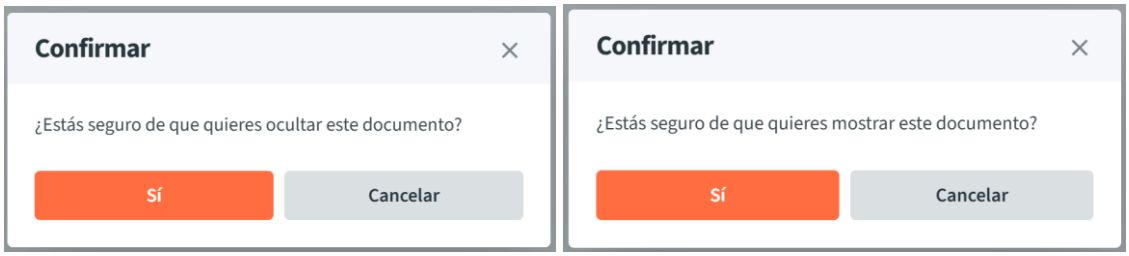

Figura 67. Mensajes confirmación ocultar/mostrar documentos

Para ver los documentos que no están visibles para los alumnos presionar el botón de *Ver Documentos No Visibles*, situado en la esquina superior derecha.

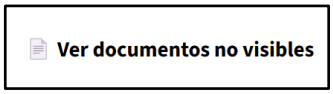

Figura 68. Botón ver documentos no visibles

Los documentos no visibles salen con una distinción para diferenciarlos de los que si son visibles.

Figura 69. Documento no visible

Eliminar: desde el menú contextual de los documentos seleccionar la opción de eliminar y confirmar.

Figura 70. Eliminación de documentos

Renombrar: desde el menú contextual de los documentos seleccionar la opción de renombrar. Rellenar el campo con el nuevo nombre, guardar y confirmar el cambio. Se puede anular el cambio con el botón cancelar.

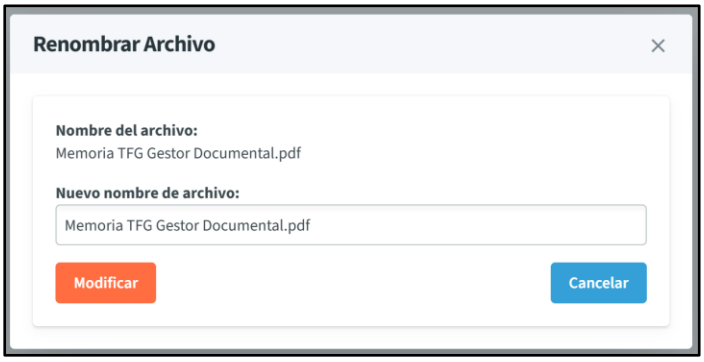

Figura 71. Renombrar documentos

Cambiar fecha de baja automática del documento: seleccionar la opción de fecha baja del menú contextual de los documentos. Seleccionar la nueva fecha y guardar. Se pueden anular los cambios con el botón cancelar.

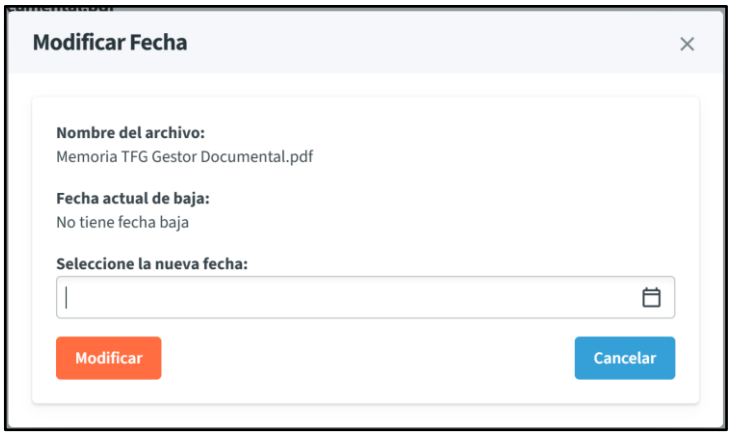

Figura 72. Modificar fecha de baja automática

Mostrar estadísticas: en el menú contextual de los documentos seleccionar la opción de Estadísticas. Para volver al menú principal presionar en el botón salir.

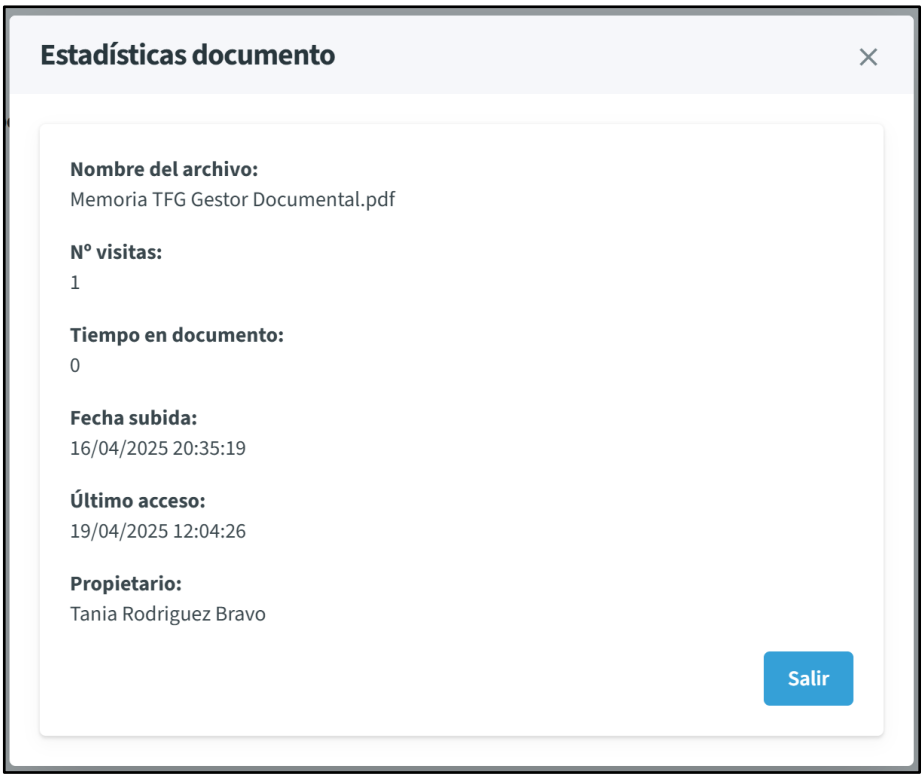

Figura 73. Pantalla de estadísticas documento

5.2.3 Gestión de etiquetas

Esta gestión permite al docente añadir o eliminar etiquetas de un documento. Desde el menú contextual de documentos (Figura 66) seleccionar la opción de etiquetas.

Desde la pantalla se puede consultar el nombre del documento seleccionado. Para eliminar las etiquetas existentes relacionadas con el documento presionar a la "x" que sale al lado de cada una de ellas. Para añadir nuevas etiquetas escribir en el campo y presionar al "+". El campo es predictivo, se mostrarán opciones de otras etiquetas ya existentes al ir escribiendo. Para guardar todos los cambios presionar al botón modificar. Se pueden cancelar las modificaciones con el botón cancelar.

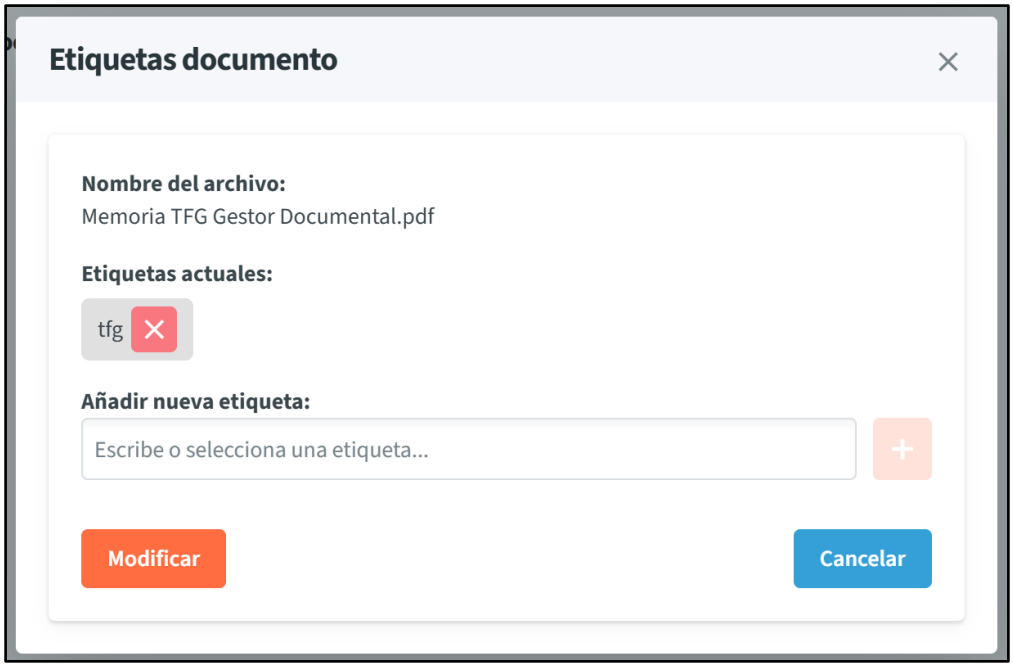

Figura 74. Pantalla modificar etiquetas

5.2.4 Gestión de carpetas

Las carpetas facilitan la organización de la documentación en los diferentes cursos. Para ello el docente puede, crear una carpeta; subir una carpeta; renombrar una carpeta existente o eliminar una carpeta y sus documentos asociados.

Crear Carpeta: desde la página de inicio de curso/grupo, hacer clic derecho en un espacio en blanco (no en otra carpeta o documento) y seleccionar la opción de Crear Carpeta del menú contextual general (Figura 75). Introducir el nombre de la carpeta y el grupo al que va a pertenecer dentro del curso y guardar. Se puede cancelar la creación presionando el botón de cancelar.

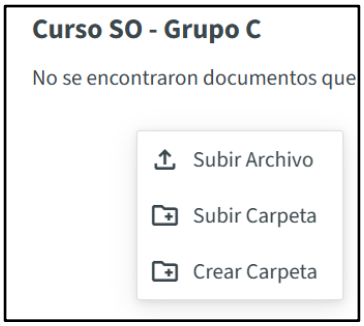

Figura 75. Menú contextual general

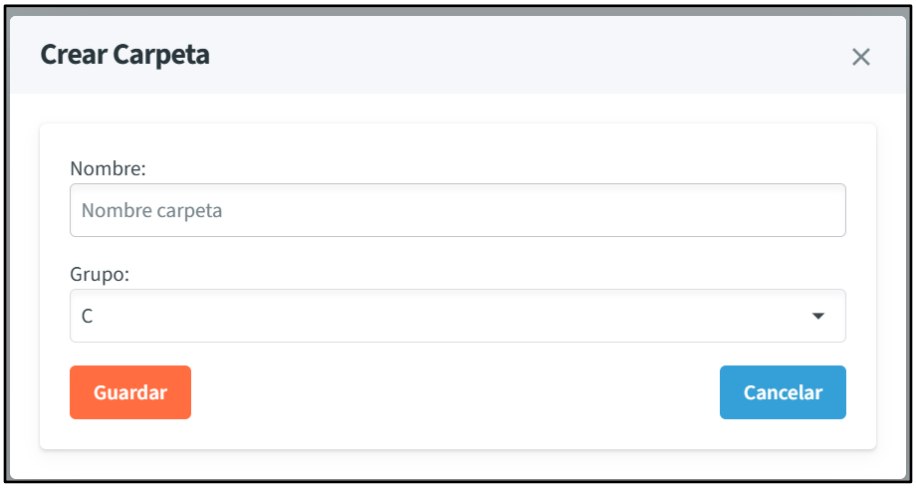

Figura 76. Crear una nueva carpeta

Subir carpeta: seleccionar la opción de subir carpeta del menú contextual general. Seleccionar la carpeta a subir, indicar el grupo al que pertenece y guardar. Se puede cancelar la subida con el botón cancelar.

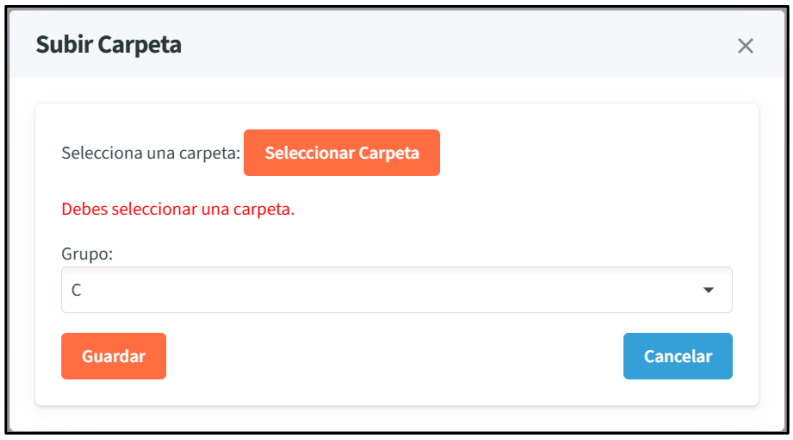

Figura 77. Subir una carpeta

Renombrar carpeta: desde la pantalla de inicio de curso/grupo, hacer clic derecho sobre la carpeta y seleccionar la opción de Renombrar del menú contextual de carpetas. Escribir el nuevo nombre y modificar. Se puede cancelar con el botón de cancelar.

Figura 78. Menú contextual carpetas

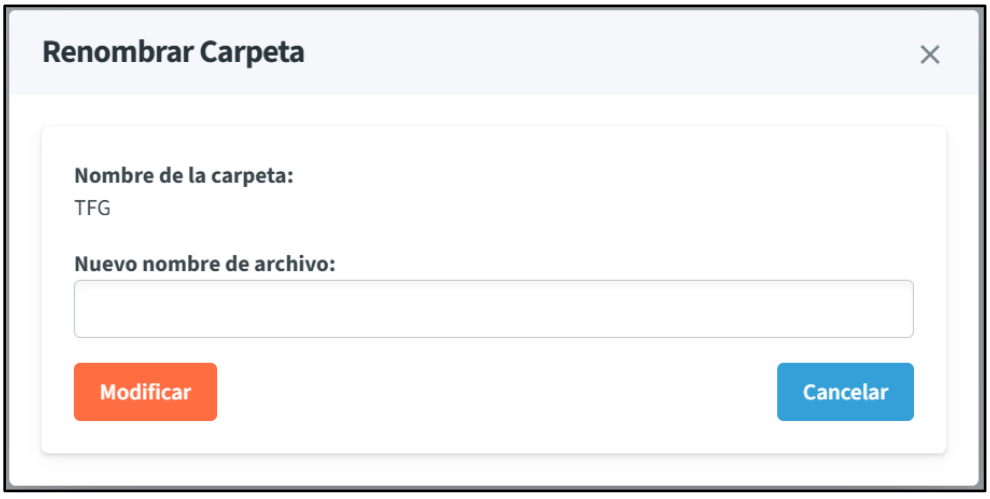

Figura 79. Renombrar carpeta

Eliminar carpeta: seleccionar la opción de eliminar del menú contextual de carpetas. Confirmar la eliminación o cancelar operación.

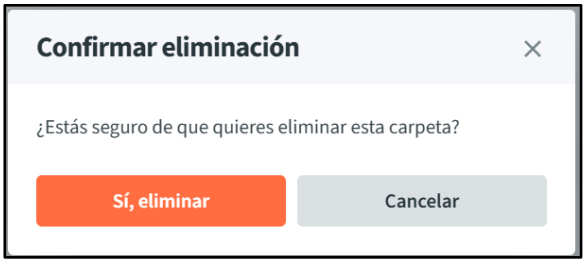

Figura 80. Eliminar carpeta

5.3 Búsqueda y filtros de documentos

Desde la página de inicio de un curso/grupo (Figura 54) los usuarios pueden hacer clic sobre el icono de la esquina derecha de las tres líneas para abrir el menú de búsqueda y filtros (Figura 81).

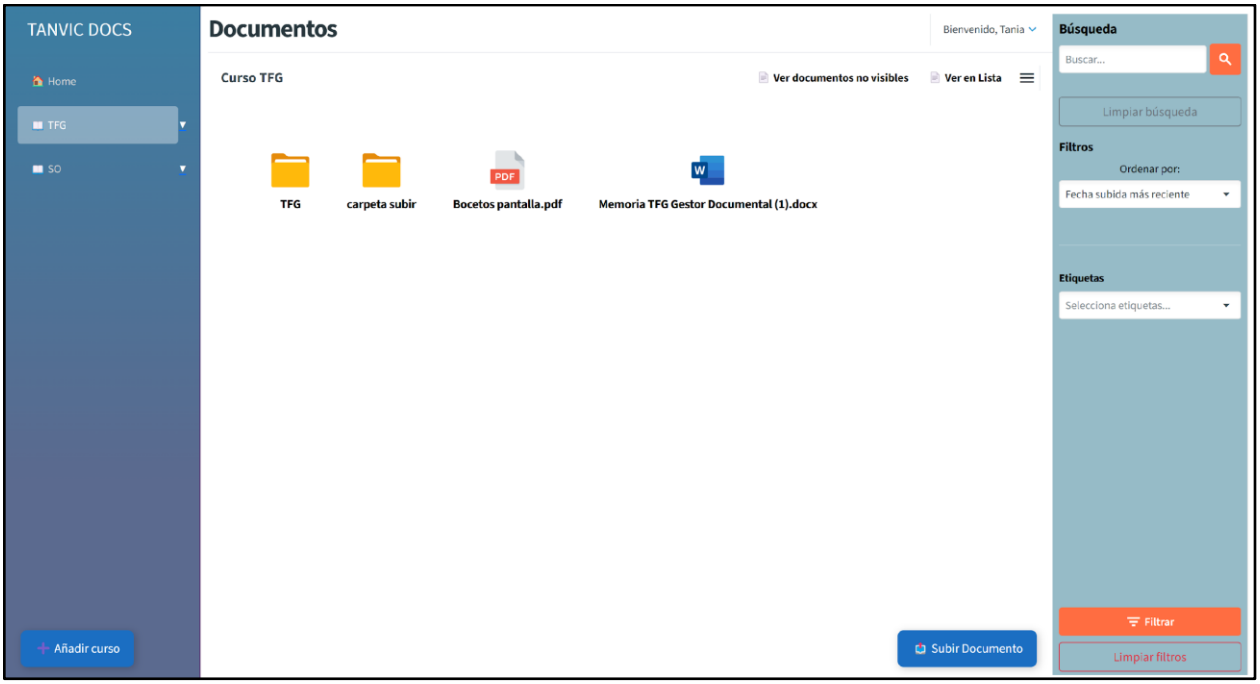

Figura 81. Menú de búsqueda y filtros

Búsqueda: desde el menú de búsqueda y filtros, introducir la palabra a buscar y hacer clic en el icono de la lupa. Se puede limpiar la búsqueda pulsando el botón limpiar búsqueda y volver al curso.

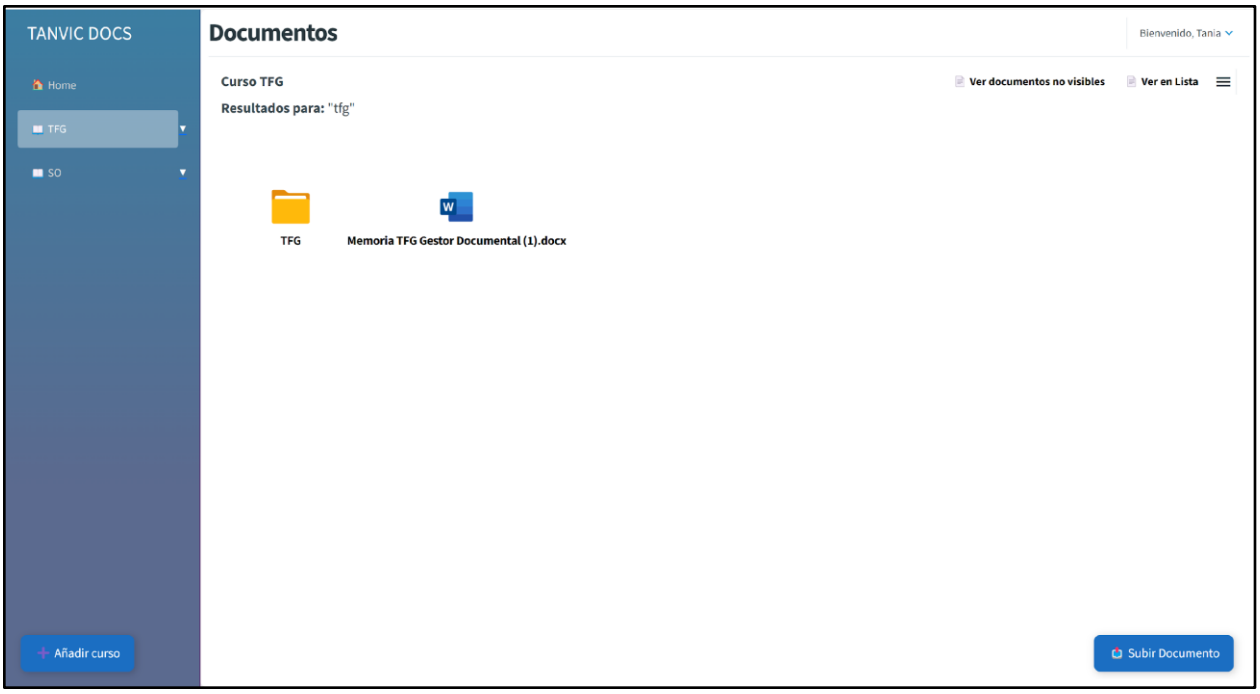

Figura 82. Resultados búsqueda

Filtros: desde el menú de búsqueda y filtros, seleccionar los filtros (ordenación y etiquetas) y filtrar. Se puede volver al curso y limpiar los filtros con el botón limpiar filtros.

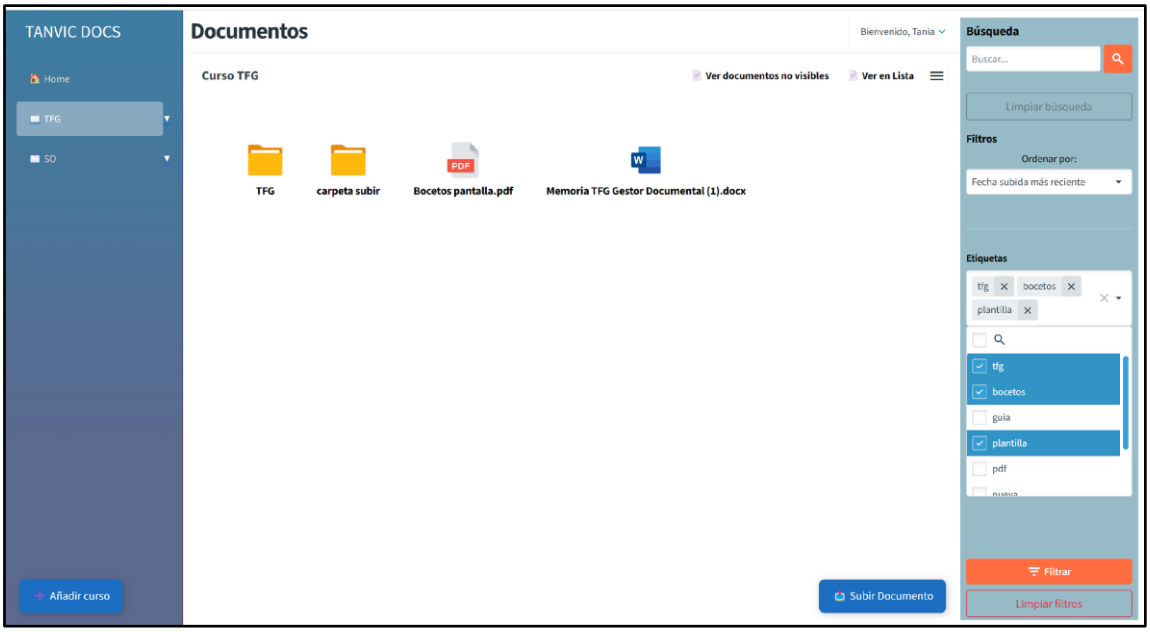

Figura 83. Seleccionar filtros búsqueda

Figura 84. Resultado filtrado

5.4 El alumno y GestorDocumental

El alumno lo único que puede hacer es acceder a la aplicación, consultar los documentos de su curso y grupo y buscar o filtrar esos documentos. La manera de hacerlo es igual que la del docente como se ve en los puntos anteriores.

5.5 El administrador y el GestorDocumental

El rol del administrador está orientada a tareas de mantenimiento y gestión interna. No cuenta con ninguna funcionalidad específica dentro de la aplicación, sino que opera principalmente a nivel de base de datos o sistema, realizando tareas como asignación de roles (por ejemplo, modificar el rol de un usuario nuevo, que por defecto es alumno, a docente si es necesario) u otras tareas técnicas para el correcto funcionamiento de la aplicación.

5.6 El log de auditoría

Se ha implementado un log que registre las acciones realizadas por los usuarios tanto a nivel individual como a nivel acumulado, además de una vista para ver cada día cuales son los elementos más visitados cada día.

Para acceder a la pantalla para revisar dichos datos se ha añadido un nuevo botón junto al botón de subir documento.

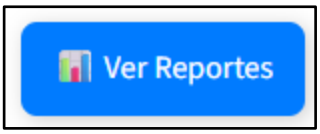

Figura 85. Botón ver reportes

Una vez pulsado el botón se muestra la pantalla con tres gráficos diferentes, el primero que muestra los top 10 documentos (Figura 86), el segundo que muestra las top 10 carpetas (Figura 87) y por último el *heatmap* (Figura 88).

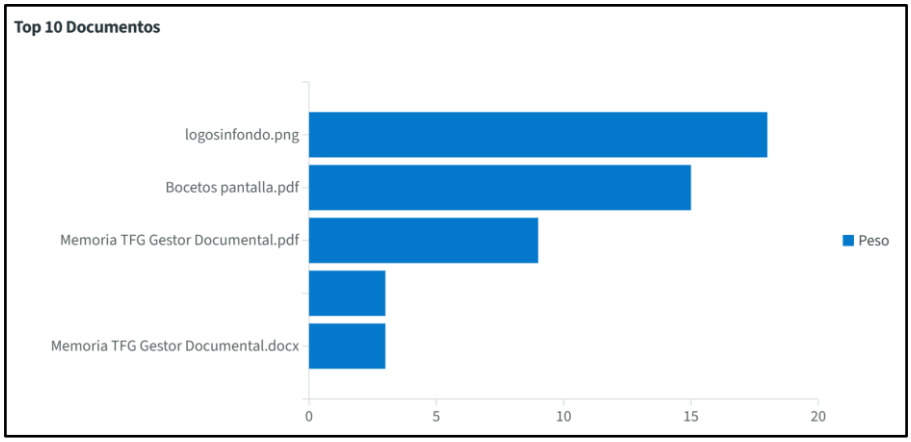

Figura 86. Top 10 documentos

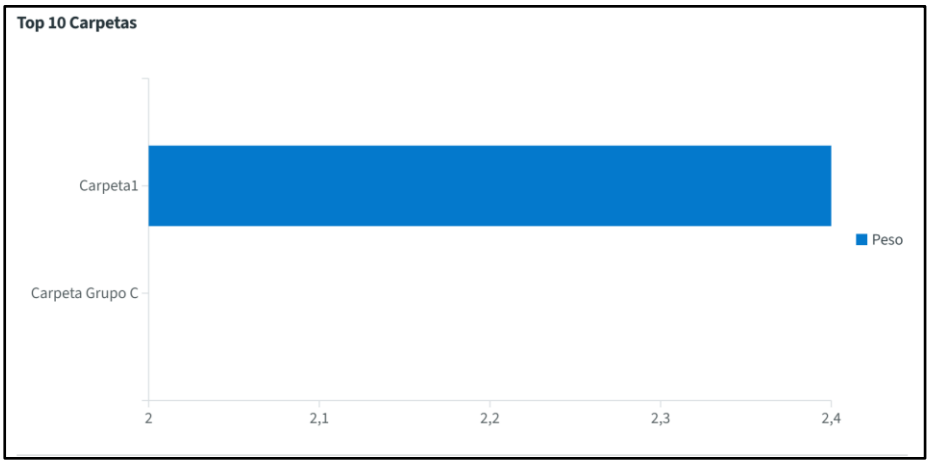

Figura 87. Top 10 carpetas

Heat-map Diario

FECHA	TIPO	ID	EVENTOS	PESO
14/05/2025 0:00:00	0	1	5	2
14/05/2025 0:00:00	1	1	2	3
14/05/2025 0:00:00	1	3	4	6
15/05/2025 0:00:00	0	1	4	1,6
15/05/2025 0:00:00	0	2	3	1,2000000000000002
15/05/2025 0:00:00	1	1	6	9
15/05/2025 0:00:00	1	3	2	3

Figura 88. Heatmap diario

5.7 Pruebas unitarias

Se han añadido pruebas unitarias básicas de las clases *Home.razor* y *CrearGrupo.razor*. Estas pruebas van orientadas a prevenir que futuros cambios de la aplicación modifiquen el funcionamiento básico requerido de la misma. De cara al futuro sería interesante realizar pruebas que comprueben la funcionalidad de manera

más exhaustiva y que cubran un mayor número de casuísticas y clases. En la Figura 89 se muestra un ejemplo de algunas pruebas creadas en el código.

```
[Theory]
[InlineData("application/pdf", "icons/pdf.png")]
[InlineData("text/plain", "icons/plain.png")]
[InlineData("application/msword", "icons/word.png")]
[InlineData("application/vnd.openxmlformats-officedocument.wordprocessingml.document", "icons/word.png")]
[InlineData("image/jpeg", "icons/imagen.png")]
[InlineData("application/zip", "icons/archivo.png")]
0 referencias
public void ObtenerIconoArchivo_ReturnsExpectedIcon(string mimeType, string expected)
{
    // Arrange
    var home = new Home();

    // Act
    var icon = home.ObtenerIconoArchivo(mimeType);

    // Assert
    Assert.Equal(expected, icon);
}

[Fact]
0 referencias
public void ObtenerIconoCarpeta_ReturnsCarpetaIcon()
{
    // Arrange
    var home = new Home();

    // Act
    var icon = home.ObtenerIconoCarpeta();

    // Assert
    Assert.Equal("icons/carpeta.png", icon);
}
```

Figura 89. Pruebas unitarias

Como se puede observar dichas pruebas son básicas pero fundamentales para verificar que no se modifica dicho funcionamiento.

Capítulo - 6 Conclusiones y trabajo a futuro

6.1 Conclusiones

Tras la finalización del trabajo, se han logrado los principales objetivos planteados inicialmente. Se ha desarrollado una aplicación software funcional, intuitiva y con una interfaz clara y profesional, orientada a facilitar la gestión y compartición de documentación académica entre docentes y alumnos. Esta herramienta permite a los profesores organizar los recursos por curso y grupo, compartir materiales de forma estructurada mediante carpetas, y ofrecer al alumnado un entorno eficaz para el acceso a dichos recursos.

El sistema incluye funcionalidades que enriquecen la experiencia del usuario, como el seguimiento de estadísticas de acceso (número de visitas, duración de visualización, propietario y fecha del último acceso), búsqueda por palabras clave, filtrado por etiquetas y ordenación según distintos criterios (más recientes, más visitados, etc.). También se ha incorporado un sistema de notificaciones por correo electrónico para alertar sobre la subida de nuevos documentos, así como un servicio de retención documental que elimina automáticamente aquellos archivos que han expirado según los criterios definidos.

6.2 Trabajo a futuro

De cara a futuras mejoras del sistema, se plantean diversas funcionalidades destinadas a enriquecer la experiencia tanto del alumnado como del profesorado. Una de las principales propuestas es la implementación de un sistema de recomendación de documentos, que sugiera automáticamente material potencialmente útil en función del historial de navegación del usuario, los documentos más consultados dentro del grupo o etiquetas asociadas a intereses previos. Esta funcionalidad puede ser clave para facilitar el acceso a contenido relevante de forma proactiva.

Relacionado con esto, también sería interesante incluir un historial de navegación personalizado, donde cada usuario pueda consultar los documentos que ha visualizado

antes y acceder rápidamente a ellos. Se podría complementar con opciones para marcar documentos como favoritos o establecer accesos rápidos.

Otra mejora relevante sería permitir que los alumnos puedan subir sus propios documentos a la plataforma, fomentando la colaboración entre compañeros. Estos documentos no serían visibles de inmediato, sino que requerirían una aprobación previa por parte del docente, lo cual mantendría el control de calidad y relevancia del contenido compartido.

Además, se podrían incluir herramientas colaborativas como comentarios en documentos o valoraciones por parte de los alumnos para que sirviera de ayuda a otros alumnos.

En cuanto al sistema de retención documental, sería interesante permitir una configuración personalizada por parte del administrador o usuario, ajustando los plazos de expiración, niveles de acceso o la posibilidad de marcar documentos como permanentes.

Por último, en relación con las pruebas unitarias, sería recomendable realizar

Estas mejoras futuras no solo ampliarían la funcionalidad del gestor documental, sino que también contribuirían a convertirlo en una plataforma más dinámica, participativa y adaptada a las necesidades reales de docentes y estudiantes.

Chapter 6 - Conclusions y future work.

6.1 (BIS) Conclusions

Following the completion of this project, the main objectives initially proposed have been successfully achieved. A functional and intuitive software application has been developed, featuring a clear and professional interface aimed at facilitating the management and sharing of academic documentation between teachers and students. This tool enables teachers to organize resources by course and group, share materials in a structured way through folders, and offer students an effective environment for accessing such resources.

The system includes features that enhance the user experience, such as access statistics tracking (number of visits, viewing duration, owner, and date of last access), keyword search, filtering by tags, and sorting based on various criteria (most recent, most viewed, etc.). An email notification system has also been integrated to alert users about newly uploaded documents, as well as a document retention service that automatically removes expired files based on predefined criteria.

6.2 (BIS) Future work

Several improvements are proposed for future development of the system, aiming to further enrich the experience for both students and teachers. One of the main proposals is the implementation of a document recommendation system, which would automatically suggest potentially useful materials based on the user's browsing history, the most consulted documents within a group, or tags related to previous interests. This feature could play a key role in proactively guiding users to relevant content.

In line with this, it would also be valuable to include a personalized browsing history, allowing each user to view previously accessed documents and quickly return to them. This could be complemented by options to mark documents as favorites or create quick-access shortcuts.

Another significant enhancement would be to allow students to upload their own documents to the platform, promoting peer-to-peer collaboration. These documents would not be made visible immediately, but would instead require prior approval by the teacher, thus maintaining quality and relevance of the shared content.

Collaborative tools such as document comments or student ratings could also be added, helping other users evaluate and make better use of available resources.

Regarding the document retention system, it would be interesting to offer personalized configuration options for administrators or users, allowing customization of expiration periods, access levels, or the ability to mark certain documents as permanent.

Finally, with respect to unit testing, it would be advisable to expand its coverage to ensure the robustness and reliability of the application under a wider range of scenarios.

These future improvements would not only expand the functionality of the document manager but would also help transform it into a more dynamic, participatory, and user-adapted platform that truly meets the evolving needs of teachers and students.

CONTRIBUCIONES PERSONALES

Estudiante 1: Tania Rodríguez Bravo

A lo largo de la realización de este Trabajo de Fin de Grado, he participado activamente en todas las fases del desarrollo. Desde las etapas iniciales, contribuyendo con ideas y participando en la toma de decisiones preliminares, hasta la fase final con el fin de conseguir los objetivos establecidos.

He tenido una implicación significativa en la etapa de codificación, asumiendo una parte considerable de esta labor, así como en la ejecución de las pruebas de funcionamiento general del sistema. Asimismo, he colaborado en la elaboración del contenido principal de la memoria, especialmente en las secciones que detallo a continuación.

- Realización de búsqueda preliminar de gestores documentales y comparación entre ellos. (Conjunta)

- Creación y gestión inicial de Jira. (Conjunta)

- Decisión de lenguaje de programación y base de datos a escoger en el proyecto. (Conjunta)

- Decisión y puesta a punto de servidor de gestión de código (GitHub) (Conjunta)

- Decisión e implementación de arquitectura de capas del proyecto. (Conjunta)

- Bocetos de las diferentes pantallas. (Conjunta)

- Configuración del proyecto inicial. (Individual)

- Diseño y creación de las pantallas de la aplicación. (Individual)

- Diseño del estilo (css) de la aplicación. (Individual)

- Realización de la funcionalidad de navegación de cursos. (Individual)

- Realización de la gestión de documentos: carga, representación y subida de nuevos documentos. (Individual)

- Realización de visualización de documentos en pdf. (Individual)

- Apoyo en corrección de visualización de documentos en el resto de los tipos de archivo.

- Realización de la funcionalidad de ocultar/mostrar documentos. (Individual)

- Realización de la funcionalidad de cambiar fecha de baja de los documentos. (Individual)

- Realización de la funcionalidad de gestión de etiquetas de los documentos. (Individual)

- Realización de la funcionalidad de renombrar documentos. (Individual)

- Realización de la funcionalidad de eliminación de documentos. (Individual)

- Apoyo en la corrección de las funcionalidades de login y registro.

- Realización del menú de perfil. (Individual)

- Realización de la gestión de carpetas. (Individual)

- Apoyo en la funcionalidad de búsqueda y filtrado.

- Apoyo en corrección de la funcionalidad gestión de grupos.

- Realización de los apartados de la memoria relacionados con las funcionalidades implementadas, y repaso de esta. (Individual)

- Realización de pruebas generales del proyecto. (Individual)

Estudiante 2: Víctor Caparrós Cruz

He estado presente en todas las fases del desarrollo del TFG, aunque más presente en las primeras etapas de este, aportando ideas y tomando decisiones acerca de la arquitectura a usar, el diseño básico y el uso de herramientas complementarias. A su vez he tenido peso en la realización de pruebas unitarias, el log y las estadísticas como se menciona debajo. He tenido un peso menor en la parte de codificación y elaboración del grueso de la memoria, aunque he contribuido en la mayoría de los apartados de manera menor.

- Realización de búsqueda preliminar de gestores documentales y comparación entre ellos. (Conjunta)

- Creación y gestión inicial de Jira junta a la definición de los sprints iniciales (Conjunta)

- Decisión de lenguaje de programación y base de datos a escoger en el proyecto. (Conjunta)

- Decisión y puesta a punto de servidor de gestión de código (GitHub) (Conjunta)

- Análisis, decisión e implementación de arquitectura de capas del proyecto. (Conjunta)

- Bocetos de las diferentes pantallas. (Conjunta)

- Realización de funcionalidad de envíos de notificaciones mediante servicio de correo externo. (Individual)

- Realización de funcionalidad de muestra de ficheros en formato docx, txt, jpg y otros diferentes a pdf. (Individual con apoyo en corrección)

- Realización de la funcionalidad gestión de grupos. (Individual con apoyo en corrección)

- Realización de login y registro. (Individual con apoyo en corrección)

- Realización de funcionalidades relacionadas con los grupos. (Individual con apoyo en corrección)

- Realización de la gestión de las estadísticas. (Individual)

- Realización de los apartados de la memoria relacionados con las funcionalidades implementadas. (Individual)

- Realización de pruebas unitarias en el proyecto. (Individual)

- Realización de funcionalidad de log. (individual)

-Realización de pantalla de auditoría y gráficos además del botón que permite acceder a esta (individual)

-Creación de tablas para almacenar los logs tanto a nivel acumulado como a nivel individual. (individual)

-Creación de 3 vistas para ver los datos de manera más ordenada en base de datos, a su vez las vistas están replicadas en código para ser mostradas mediante los gráficos de la funcionalidad de reportes (individual)

- Creación en la memoria de la documentación relacionada con las pruebas realizadas en cada uno de los sprints (individual)

- Documentación en la memoria y aportación de código de las pruebas unitarias en la memoria (individual)

Bibliografía

[1] ownCloud, «Cloud For Education» [En línea]. Available: https://owncloud.com/industry-solutions/cloud-for-educational-institution/ [Último acceso: 26 11 2012]

[2] «Cloud For Education» [En línea]. Available: https://doc.owncloud.com/#spaces [Último acceso: 26 11 2024]

[3] «Software para gestión de documentos de forma inteligente | DocuWare» [En línea]. Available: https://start.docuware.com/es/gestion-documental-solucion [Último acceso: 26 11 2024]

[4] «DocuWare | Funciones y capacidades» [En línea]. Available: https://start.docuware.com/es/funciones-capacidades [Último acceso: 26 11 2024]

[5] Moodle. [En línea] Available: https://moodle.org/ [Último acceso: 26 11 2024]

[6] «School management platform for public schools.» [En línea]. Available: https://additioapp.com/en/schools/ [Último acceso: 26 11 2024]

[7] «Assignment Management System» [En línea]., Available: https://educloud.app/lms/assignment-management-system [Último acceso: 26 11 2024]

[8] «Zoho WorkDrive | Online file management for teams» [En línea] Available: https://www.zoho.com/workdrive/ [Último acceso: 26 11 2024]

[9] «Gestión Documental: Lista de Funcionalidades | R2Docuo» [En llínea] Available:https://www.r2docuo.com/es/gestion-documental [Último acceso: 26 11 2024]

[10] «Gestión Documental Open Source | OpenKM» [En línea] Available: https://www.openkm.com/es/gestion-documental.html [Último acceso: 26 11 2024]

[11] «Qué es GitHub, por qué es tan popular y cómo empezar» [En línea] Available: https://www.hostinger.es/tutoriales/que-es-github [Último acceso: 26 11 2024]

[12] «¿Qué es Jira? Herramienta de Gestión de Proyectos» [En línea] Available: https://imaginaformacion.com/tutoriales/que-es-jira [Último acceso: 26 11 2024]

[13] «¿Qué es Visual Studio?» [En línea] Available: https://learn.microsoft.com/es-es/visualstudio/get-started/ [Último acceso 26 11 2024]
[14] «¡Adiós Javascript, hola Blazor! Qué es y cómo funciona Blazor» [En línea] Available: https://www.hiberus.com/ [Último acceso: 26 11 2024]
[15] «¿Qué es Microsoft SQL Server y para qué sirve?» [En línea] Available: https://intelequia.com/es/blog/post/ [Último acceso: 26 11 2024]
[16] «Qué es la arquitectura en capas, ventajas y ejemplos» [En línea] Available: https://blog.hubspot.es/website/que-es-arquitectura-en-capas [Último acceso: 26 11 2024]
[17] «Qué es SCRUM» [En línea] Available: https://proyectosagiles.org/que-es-scrum/ [Último acceso: 26 11 2024]
[18] «Modelos de hospedaje Blazor en ASP.NET Core» [En línea] Available: https://learn.microsoft.com/es-es/blazorowebassembly [Último acceso: 28 03 2025]
[19] «Duración, configuración e inicialización de DbContext» [En línea] Available: https://learn.microsoft.com/es-es/ef/core/dbcontext-configuration/ [Último acceso: 28 03 2025]
[20] «Radzen. Accelerated, smarter and cost-effective Blazor development» [En línea] Available: https://www.radzen.com/ [Último acceso: 28 03 2025]
[21] «Radzen Blazor Components» [En línea] Available: https://blazor.radzen.com/get-started [Último acceso: 26 04 2025]
[22] «DbSet<TEntity> Class» [En línea] Available: https://learn.microsoft.com/es-es/dotnet/api/microsoft.entityframeworkcore.dbsetnet8 [Último acceso: 01 04 2025]
[23] «Proveedor de bases de datos de MICROSOFT SQL Server EF Core» [En línea] Available: https://learn.microsoft.com/es-es/ef/core/providers/sql-server/?tabs=dotnet-core-cli [Último acceso: 01 04 20 25]
[24] «Enlace de datos de ASP.NET Core Blazor» [En línea] Available:https://learn.microsoft.com/es-es/aspnet/blazor/data-binding?view=aspnetcore-8.0 [Último acceso: 03 04 2025]
[25] «Ciclo de vida de los componentes de ASP.NET Core Razor» [En línea] Available: https://learn.microsoft.com/es-es/aspnet/blazor/components/lifecycle [Último acceso: 03 04 2025]

[26] «Ciclo de vida de componentes en Blazor» [En línea] Available:https://www.netmentor.es/entrada/ciclo-vida-componentes-blazor [Último acceso: 03 04 2025]
[27] «Cargas de archivos de ASP.NET Core Blazor» [En línea] Available: https://learn.microsoft.com/es-es/aspnet/core/blazor/file-uploads [Último acceso: 03 04 2025]
[28] «IISServerOptions.MaxRequestBodySize Propiedad» [En línea] Available: https://learn.microsoft.com/builder.iisserveroptions.maxrequestbodysize [Último acceso: 03 04 2025]
[29] «FormOptions.MultipartBodyLengthLimit Propiedad» [En línea] Available: https://learn.microsoft.com/formoptions.multipartbodylengthlimit [Último acceso: 04 04 2025]
[30] «ASP.NET Core Blazor authentication and authorization» [En línea] Available:https://learn.microsoft.com/en-us/aspnet/core/blazor/security [Último acceso: 04 04 2025]
[31] «Autenticación y autorización de ASP.NET Core Blazor» [En línea] Available: https://learn.microsoft.com/es-es/aspnet/core/blazor/security [Último acceso: 04 04 2025]
[32] «Blazor Server, use ProtectedSessionStorage» [En línea] Available: https://www.peug.net/en/blazor-server-use-protectedsessionstorage/ [Último acceso: 04 04 2025]
[33] «ProtectedSessionStorage Clase» [En línea] Available:https://learn.microsoft.com/server.protectedbrowserstorage.protectedsessionstorage [Último acceso: 04 04 2025]
[34] «¿Qué es bcrypt?» [En línea] Available: https://www.skysnag.com/es/blog/what-is-bcrypt/ [Último acceso: 04 04 2025]
[35] «Blowfish» [En línea] Available: https://aprende.academiaciberseguridad.com/books/conceptos/page/blowfish [Último acceso: 04 04 2025]

[36] «Tipos de cifrado y modalidades de cifrado» [En línea] Available: https://www.ibm.com/docs/es/informix-servers/12.10.0?topic=encryption-ciphers-modes [Último acceso: 04 04 2025]
[37] «What is the difference between bcrypt and SHA265? » [En línea] Available: https://rietta.com/blog/bcrypt-not-sha-for-passwords/ [Último acceso: 04 04 2025]
[38] «bcrypt.NET» [En línea] Available: https://github.com/BcryptNet/bcrypt.net [Último acceso: 04 04 2025