UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Aplicación web para la gestión de metas personales y colaborativas

PROYECTO FIN DE GRADO

Daniel Muñoz García
Grado en Ingeniería de Software

Madrid, 2025

UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Grado en Ingeniería de Software

Aplicación web para la gestión de metas personales y colaborativas

PROYECTO FIN DE GRADO

Daniel Muñoz García
Grado en Ingeniería de Software

Bajo la dirección de:
Dr. Borja Bordel Sánchez

Madrid, 2025

Título: Aplicación web para la gestión de metas personales y colaborativas.

Autor: Daniel Muñoz García

Grado en Ingeniería de Software

Dirección:

Dr. Borja Bordel Sánchez

Agradecimientos

Ha sido un largo viaje hasta llegar aquí. No solo hablo del tiempo que he dedicado a este trabajo, sino a todos los años de esta gran etapa de mi vida que, por fortuna o por desgracia han llegado a su fin, y va a suponer, sin ninguna duda, un antes y un después. Una experiencia la cual me ha hecho madurar, crecer, aprender y disfrutar. Un camino lleno de agradecimientos a muchas personas, las cuales sería imposible de nombrar a todas ellas en estos pocos párrafos.

En primer lugar, agradecer a todos los profesores que han pasado por mi etapa universitaria, los cuales me han hecho aprender y madurar.

En segundo lugar, agradecer a los amigos que me acompañaron durante todas las clases, exámenes y ratos en la cafetería durante estos años que, sin duda, hicieron de esta etapa más completa, divertida y lucrativa. En especial nombrar a Alejandro Caro, David Peñas, Diego Casero, Gabriel Alexander, Vladimir Karapetian, Ángel Briones y Silvia Bespín. Nunca olvidaré todos los momentos vividos.

Por otro lado, me gustaría agradecer especialmente a mi novia y a toda su familia. Gracias Alejandra, por tu apoyo incondicional, por hacer que cada día quiera sacar la mejor versión de mí y por compartir conmigo tanto los momentos buenos, como los no tan buenos durante estos años.

Hacer especial mención a mi hermano Pedro, que me ayudó a elegir mi camino siguiendo sus mismos pasos, me apoyó y ayudó en cada momento, y siempre me deseó lo mejor. Gracias por ser el mejor hermano mayor que alguien pueda imaginar.

Y cómo no, a mis padres, Susana y José. Gracias por darme esta gran oportunidad de realizar mis estudios, por el constante apoyo y por no dejarme nunca tirar la toalla. Es imposible expresar todo el agradecimiento que siento y lo mucho que os quiero.

Además, una muy especial mención y agradecimiento a mi yaya, Carmen, por darme todo el amor del mundo, los mejores consejos, el máximo apoyo y por sentirte siempre tan orgullosa de mí.

Por último, expresar mi sincero agradecimiento a mi tutor Borja Bordel Sánchez, por su dedicación y orientación durante el desarrollo del proyecto.

Abstract

This project aims to design and develop a web application for planning and tracking personal and collaborative goals. The idea comes from the difficulty many people have when trying to organise their medium- and long-term objectives, break them down into achievable tasks, and keep a clear and up-to-date record of their progress. In practice, this management is usually spread across several generic tools (spreadsheets, notes applications, mobile reminders), which makes it hard to have an overall view and to coordinate when goals are shared with other people. The proposed application puts all this information in one place and offers a simple interface that allows users to create goals, set deadlines and units of measure, register contributions or tasks, and clearly see the level of completion.

The solution has been implemented as a web application using technologies that are widely used in software development. The back-end has been developed in PHP, structuring the business logic into different modules responsible for managing users, goals and collaborations. The user interface has been built with HTML5, CSS3 and JavaScript, with a focus on a clean and consistent design that adapts to different screen sizes. For data storage, a MySQL relational database has been used, modelling entities such as users, individual and collaborative goals, tasks, progress records and participation relationships between users. Basic security measures have been included, such as session management and the use of CSRF tokens in forms, in order to reduce the risk of common attacks on web applications.

The system offers two main types of interaction: management of personal goals and collaboration on shared goals. Each user can register and log in, create new goals by defining a title, description, due date and unit of measure (for example hours, deliveries or percentages), update their progress step by step, mark goals as completed and review their history. In both individual and collaborative goals, it is possible to work with two types of goals: numeric goals, focused on reaching a quantitative objective through the sum or subtraction of contributions (for example, reaching a certain number of study hours), and task-based goals, where the objective is to complete a list of partial tasks that contribute to the global goal. In collaborative goals, the owner of the goal can invite other participants, who can

register their own numeric contributions, assign tasks to themselves or to others, and mark these tasks as completed. The detailed view of each goal shows the total of all contributions, the overall progress percentage (calculated either from the numeric target or from the number of completed tasks) and a history of updates. This improves transparency and helps to share responsibilities. In addition, the main page offers a summary panel with goals that are close to their deadline and goals that are still incomplete, helping users decide what to work on first.

The development of the project has followed an iterative and incremental approach, adding and refining features as the requirements were validated. A Git-based version control system has been used to manage the source code and record the evolution of the application. During the process, tests have been designed and executed for the most critical flows, such as registration and authentication, creation and editing of goals, recording progress and managing collaborations. When problems appeared, they were fixed and the database design and user interface were adjusted when necessary. For now, the application has been deployed and tested in a local development environment, which has made it easier to verify that the implemented logic works correctly.

Possible future improvements include adding automatic reminders (for example, via email or notifications), integrating with external calendars to show goal deadlines, extending the statistics module with charts showing progress over time, and adding mechanisms to increase user motivation. Another planned step is to deploy the application on a server accessible from the internet, configuring a web server (such as Apache or Nginx), enabling remote access to the database and strengthening security with HTTPS. In the medium term, it would also be interesting to develop a mobile application that reuses the system's logic and improves accessibility for daily use. Overall, the developed application meets the objectives defined for this Final Degree Project in Software Engineering. It provides a functional tool for structured management of personal and collaborative goals and a solid foundation for future evolution of the project.

Resumen

Este proyecto tiene como objetivo diseñar y desarrollar una aplicación web para la planificación y el seguimiento de metas personales y colaborativas. La idea surge de la dificultad que encuentran muchas personas a la hora de organizar sus objetivos a medio y largo plazo, dividirlos en tareas alcanzables y mantener un registro claro y actualizado de su progreso. En la práctica, esta gestión suele repartirse entre diferentes herramientas (hojas de cálculo, aplicaciones de notas, recordatorios del móvil), lo que dificulta tener una visión general y coordinarse cuando las metas se comparten con otras personas. La aplicación propuesta centraliza esta información y ofrece una interfaz sencilla que permite crear metas, asociarles fechas límite y unidades de medida, registrar aportaciones o tareas, y visualizar de forma clara el progreso real.

La solución se ha implementado como una aplicación web basada en tecnologías muy utilizadas en el desarrollo software. El back-end se ha desarrollado en PHP, estructurando la lógica de negocio en distintos módulos responsables de la gestión de usuarios, metas y colaboraciones. La interfaz de usuario se ha construido con HTML5, CSS3 y JavaScript, priorizando un diseño limpio, coherente y adaptable a diferentes tamaños de pantalla. Para la persistencia de datos se ha utilizado una base de datos relacional MySQL, en la que se modelan entidades como usuarios, metas individuales y colaborativas, tareas, registros de progreso y relaciones de participación entre usuarios. Se han incorporado medidas básicas de seguridad, como la gestión de sesiones y el uso de tokens CSRF en los formularios, con el fin de mitigar ataques comunes en aplicaciones web.

El sistema contempla principalmente dos tipos de interacción: la gestión de metas personales y la colaboración en metas compartidas. Cada usuario puede registrarse e iniciar sesión, crear nuevas metas definiendo un título, una descripción, una fecha de vencimiento y una unidad de medida (por ejemplo, horas, entregas o porcentajes), actualizar su progreso de manera incremental, marcar metas como completadas y consultar su historial. Tanto en metas individuales como colaborativas, es posible trabajar con dos tipos de metas: metas numéricas, orientadas a alcanzar un objetivo cuantitativo mediante la suma o resta de aportaciones (por ejemplo, llegar a cierto número de horas de estudio), y metas de

tareas, basadas en completar una lista de tareas parciales que contribuyen al objetivo global. En el caso de las metas colaborativas, el usuario propietario puede invitar a otros participantes, que pueden registrar sus propias aportaciones numéricas, asignarles tareas y marcar estas como completadas. La vista detallada de cada meta muestra la suma de las contribuciones, el porcentaje total de avance calculado en función del objetivo numérico o del número de tareas completadas y un histórico de actualizaciones, lo que facilita la transparencia y el reparto de responsabilidades. Además, la página principal ofrece un panel de resumen con las metas próximas a vencer y aquellas aún no completadas, ayudando al usuario a priorizar en qué trabajar.

El desarrollo del proyecto se ha realizado siguiendo un enfoque iterativo e incremental, incorporando y refinando funcionalidades a medida que se iban validando los requisitos. Se ha utilizado un sistema de control de versiones basado en Git para gestionar el código fuente y registrar la evolución de la aplicación. A lo largo del proceso se han definido y ejecutado pruebas sobre los flujos más críticos como registro y autenticación, creación y edición de metas, registro de progreso y gestión de colaboraciones, corrigiendo errores y ajustando el diseño de la base de datos y de la interfaz cuando ha sido necesario. Por el momento, la aplicación se ha desplegado y probado en un entorno local de desarrollo, lo que ha facilitado la verificación de la lógica implementada.

Entre las posibles mejoras futuras se encuentran la incorporación de recordatorios automáticos, la integración con calendarios externos para reflejar las fechas límite de las metas, la ampliación del módulo de estadísticas mediante gráficos de evolución del progreso y la introducción de mecanismos que fomenten la motivación del usuario. También se plantea el despliegue de la aplicación en un servidor accesible desde internet, configurando un servidor web, habilitando el acceso remoto a la base de datos y reforzando la seguridad mediante HTTPS. A medio plazo, sería interesante desarrollar una aplicación móvil que reutilice la lógica del sistema y mejore la accesibilidad en el uso diario. En conjunto, la aplicación desarrollada cumple los objetivos planteados para este Trabajo de Fin de Grado en Ingeniería del Software, proporcionando una herramienta funcional para la gestión estructurada de metas personales y colaborativas y una base sólida sobre la que seguir evolucionando el proyecto.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Abreviaturas y Acrónimos

UPM	Universidad Politécnica de Madrid
SQL	Structured Query Language
HTTP	HyperText Transfer Protocol. Protocolo de comunicación web
HTTPS	HyperText Transfer Protocol Secure
HTML	HyperText Markup Language
URL	Uniform Resource Locator. Dirección única de un recurso en la web
XML	eXtensible Markup Language
CRUD	Create, Read, Update, Delete.
CSS	Cascading Style Sheets. Lenguaje de estilos
CSRF	Cross-Site Request Forgery. Tipo de ataque web
ODS	Objetivos de Desarrollo Sostenible
ONU	Organización de las Naciones Unidas
RGPD	Reglamento General de Protección de Datos
SVG	Scalable Vector Graphics
DOM	Document Object Model
UTF	UTF-8 – Unicode Transformation Format
RF	Requisito Funcional
RNF	Requisito No Funcional
CU	Casos de Uso

1. Introducción

El proyecto tiene como objetivo la creación de una aplicación web para la planificación y el seguimiento de metas personales y colaborativas. La motivación principal es ofrecer una herramienta específica que permita a los usuarios definir objetivos, desglosarlos en metas alcanzables y registrar su progreso de forma estructurada, evitando la dispersión de información entre múltiples aplicaciones genéricas. Este documento describe las distintas fases del proyecto, desde el análisis de la necesidad y la definición de requisitos hasta el diseño, implementación y validación del sistema, así como una serie de posibles mejoras y ampliaciones futuras.

Una parte importante del trabajo se ha centrado en el diseño de una interfaz atractiva, moderna y coherente, con el objetivo de fomentar el uso continuo de la aplicación. No solo se busca que el sistema sea funcional, sino también que resulte agradable de utilizar en el día a día: una organización clara de las pantallas, el uso de colores y tipografías consistentes y una presentación visual cuidada que invite al usuario a volver para actualizar sus metas y consultar su progreso.

Las principales características del proyecto son:

Interfaz centrada en el usuario y visualmente atractiva: El sistema está diseñado para proporcionar una experiencia intuitiva y agradable, permitiendo a los usuarios crear, gestionar y monitorizar sus metas personales y colaborativas a través de una interfaz web moderna, responsive y estéticamente cuidada. La atención al diseño visual busca aumentar la motivación del usuario y favorecer el uso continuado de la herramienta.

Gestión integral de datos: La aplicación procesa y almacena información relacionada con usuarios, metas individuales y compartidas, tareas asociadas, fechas límite y registros de progreso, garantizando la persistencia y consistencia de los datos mediante una base de datos relacional MySQL.

Soporte para distintos tipos de metas: El sistema permite trabajar con metas numéricas, orientadas a alcanzar un objetivo cuantitativo mediante la suma o resta de aportaciones, y con metas basadas en tareas, estructuradas en listas de subtareas que deben completarse para alcanzar el objetivo global. Ambos tipos de metas pueden utilizarse tanto de forma individual como en metas colaborativas.

Gestión de colaboraciones entre usuarios: La aplicación facilita la creación de metas compartidas en las que un usuario actúa como propietario e invita a otros participantes, que pueden registrar sus propias aportaciones numéricas, gestionar tareas y marcarlas como completadas, favoreciendo la transparencia y el reparto de responsabilidades.

Integración entre front-end y back-end: La solución integra una interfaz de usuario desarrollada con HTML5, CSS3 y JavaScript con un back-end en PHP y una base de datos MySQL, de manera que las interacciones del usuario se traducen en operaciones de negocio y almacenamiento de datos de forma eficiente y segura.

Arquitectura cliente-servidor y preparada para el despliegue: Aunque el sistema se ha desarrollado y probado inicialmente en un entorno local, su organización y el uso de tecnologías estándar facilitan su despliegue futuro en un servidor web accesible a través de internet.

El desarrollo del proyecto se estructuró siguiendo un modelo en cascada, avanzando de manera secuencial a través de etapas bien definidas: análisis de requisitos, diseño, implementación, validación y una fase final de cierre y mantenimiento básico. Esta metodología ha permitido asegurar que cada fase se completara y revisara antes de iniciar la siguiente, reduciendo el riesgo de inconsistencias y facilitando la trazabilidad de las decisiones tomadas durante el proceso de desarrollo.

1.1. Objetivos del proyecto

El objetivo general de este proyecto es diseñar y desarrollar una aplicación web que permita planificar y hacer seguimiento de metas personales y colaborativas, ofreciendo una interfaz atractiva y fácil de usar que incentive a los usuarios a consultar y actualizar sus objetivos de forma continuada.

A partir de este objetivo general, se definen los siguientes objetivos específicos:

- **Analizar las necesidades y requisitos del sistema**

Identificar qué problemas tienen los usuarios al gestionar sus metas personales y compartidas, qué información necesitan registrar (fechas límite, tareas, progresos, colaboradores, etc.) y qué funcionalidades mínimas debe ofrecer la aplicación para resultar útil en el día a día.

- **Diseñar el modelo de datos y la arquitectura de la aplicación**

Definir un modelo de datos relacional en MySQL que incluya usuarios, metas, tipos de meta, tareas, registros de progreso y participantes, así como una arquitectura web implementada con PHP, HTML, CSS y JavaScript.

- **Implementar tipos de metas diferenciadas**

Diseñar e implementar un sistema de metas que contemple dos modalidades de uso diferentes: metas individuales y metas colaborativas.

Dos tipos de metas: metas numéricas, orientadas a alcanzar un valor cuantitativo mediante la suma o resta de aportaciones, y metas de tareas, basadas en completar una lista de tareas.

En el caso de las metas colaborativas, se deberá registrar las aportaciones de cada integrante y calcular el porcentaje de contribución individual respecto al progreso total de la meta. Además, se deberá permitir que las tareas se asignen a un miembro concreto del equipo, de forma que sea posible saber quién es responsable de cada una.

- **Desarrollar la gestión de metas personales**

Implementar las funciones necesarias para que un usuario pueda crear, editar y eliminar metas individuales, establecer fechas límite, registrar o actualizar su progreso, marcar metas como completadas y consultar un historial de metas creadas y finalizadas.

- **Desarrollar la gestión de metas colaborativas**

Implementar un módulo específico que permita a un usuario crear metas colaborativas, invitar a otros participantes, registrar las aportaciones de cada uno (numéricas y mediante tareas), visualizar el porcentaje de contribución individual y mostrar el progreso global de la meta de forma agregada.

- **Diseñar una interfaz de usuario atractiva, coherente y responsive**

Construir una interfaz web visualmente cuidada y consistente, con una organización clara de las pantallas, uso de colores y tipografías coherentes y diseño adaptable a distintos tamaños de pantalla, con el fin de mejorar la experiencia de uso y favorecer la utilización continua de la aplicación.

- **Incorporar mecanismos básicos de seguridad y asegurar la calidad**

Integrar gestión de sesiones, protección frente a ataques comunes mediante tokens CSRF y validaciones de datos en el servidor. Además, definir y ejecutar pruebas sobre los flujos principales (registro, inicio de sesión, creación y edición de metas, registro de progreso y gestión de colaboraciones) para comprobar el correcto funcionamiento del sistema.

- **Documentar la solución y las decisiones de diseño**

Elaborar la documentación técnica y de usuario, describiendo el modelo de datos, la arquitectura, las funcionalidades implementadas, las pruebas realizadas y las posibles líneas de mejora, de forma que el sistema pueda mantenerse y ampliarse en el futuro.

2. Marco Teórico

En este apartado se presentan los conceptos teóricos, las tecnologías y las herramientas que sirven de base al desarrollo de esta aplicación. Se introducirán las ideas clave relacionadas con la planificación y el seguimiento de metas, la motivación del usuario y la importancia de una interfaz cuidada para favorecer el uso continuado de una herramienta de este tipo. También se tratarán algunos conceptos generales sobre aplicaciones web y arquitectura en capas, que componen la solución propuesta.

A continuación, se describirá el modelo de desarrollo seleccionado y se justificarán las razones por las que resulta adecuado para la organización del trabajo en este proyecto. Finalmente, se detallarán las principales tecnologías y herramientas empleadas (lenguajes, gestor de base de datos, entorno de desarrollo y sistema de control de versiones), explicando el papel que desempeña cada una de ellas dentro de la aplicación y los motivos que han llevado a su elección frente a otras alternativas posibles.

2.1. Gestión de metas personales y colaborativas

La gestión efectiva de metas personales y colaborativas es un aspecto fundamental del desarrollo individual y del trabajo en equipo, y ha cobrado una relevancia creciente en la era digital. Plantear objetivos claros, desglosarlos en pasos alcanzables y revisar periódicamente el progreso son prácticas que mejoran la motivación y aumentan la probabilidad de cumplir lo que se propone. Sin embargo, en la práctica diaria, muchas de estas acciones se realizan de forma informal o desorganizada, lo que reduce su eficacia.

- **Problemática actual:**

En la actualidad, la gestión de metas presenta varios problemas importantes. La mayoría de las personas utilizan métodos dispersos y poco integrados para organizar sus objetivos: listas escritas a mano, notas dispersas en el teléfono, hojas

de cálculo o aplicaciones muy básicas que solo permiten registrar tareas sueltas. Esta dispersión de la información provoca diversas dificultades:

Falta de visión global del progreso: Sin un sistema centralizado, resulta complicado saber cuánto se ha avanzado realmente hacia una meta, qué queda pendiente o qué objetivos se están descuidando.

Pérdida de motivación a medio y largo plazo: La ausencia de indicadores visuales de progreso y de un seguimiento estructurado hace que, con el tiempo, las metas pierdan prioridad frente a otras tareas más inmediatas.

Desorganización temporal: Cuando las metas no están vinculadas a fechas límite ni se relacionan con una planificación mínima, la asignación de tiempo se vuelve imprecisa y las tareas importantes tienden a posponerse.

En el caso de metas compartidas, estos problemas se agravan. No siempre queda claro qué ha aportado cada miembro del grupo, ni en qué medida ha contribuido al objetivo global. Además, las responsabilidades sobre tareas concretas suelen difuminarse, lo que puede generar solapamientos, tareas sin responsable o malentendidos en la coordinación.

- **Tipos de metas:**

Para afrontar estos retos, es necesario distinguir entre diferentes tipos de metas y formas de seguimiento:

Metas numéricas, centradas en alcanzar un valor cuantitativo (por ejemplo, horas dedicadas, entregas realizadas o unidades completadas).

Metas basadas en tareas, en las que el avance se mide por la cantidad de subtareas realizadas respecto al total planeado.

En un entorno colaborativo, esta distinción debe combinarse con la capacidad de:

Registrar las aportaciones de cada integrante, de forma que pueda conocerse su porcentaje de participación respecto al avance total.

Asignar tareas a miembros del equipo, lo que permite establecer responsabilidades y mejorar la coordinación.

La aplicación desarrollada en este proyecto toma estos conceptos como base para ofrecer un sistema en el que metas individuales y colaborativas, numéricas y de tareas, puedan gestionarse de forma unificada y estructurada.

- **Beneficios de la digitalización en la gestión de metas:**

La digitalización de la gestión de metas aporta una serie de ventajas frente a los métodos tradicionales:

Centralización de la información:

Todos los datos relacionados con las metas (descripciones, fechas límite, tareas, participantes y registros de progreso) se almacenan en un único sistema, reduciendo la dispersión y facilitando la consulta.

Visualización clara del progreso:

Barras de progreso, porcentajes y listados de tareas completadas ofrecen una representación inmediata del estado de cada meta, lo que ayuda al usuario a evaluar su situación y a tomar decisiones sobre qué priorizar.

Seguimiento de la colaboración:

En metas compartidas, la posibilidad de ver la contribución de cada integrante y las tareas asignadas a cada uno, mejora la transparencia, el reparto de trabajo y la sensación de responsabilidad compartida.

Persistencia y organización a largo plazo:

El almacenamiento estructurado permite consultar metas antiguas, revisar cómo se han alcanzado ciertos objetivos y utilizar esta información como referencia para metas futuras.

En el contexto de este proyecto, estos beneficios se combinan con un énfasis especial en una interfaz visualmente atractiva y fácil de usar, con el objetivo de que el usuario no solo disponga de un sistema estructurado, sino que además se sienta motivado a utilizarlo de forma continua para planificar, registrar y revisar sus metas.

2.2. Modelo de desarrollo en cascada

Para el desarrollo de este proyecto se adoptó un enfoque basado en las mejores prácticas de desarrollo web moderno, en el que se ha priorizado la experiencia del usuario, la escalabilidad y el mantenimiento a largo plazo. A nivel de proceso, se siguió un modelo de desarrollo en cascada, organizando el trabajo en fases secuenciales y bien delimitadas, con el objetivo de mantener el proyecto bajo control y asegurar que cada etapa quedara razonablemente cerrada antes de avanzar a la siguiente.

2.2.1. Modelo en cascada: visión general

El modelo de desarrollo en cascada es una metodología clásica de ingeniería del software que organiza el ciclo de vida de un proyecto en una serie de fases lineales. Cada fase tiene unos objetivos y entregables específicos, y su salida sirve de entrada para la siguiente. De forma general, las etapas suelen ser:

Análisis de requisitos: recogida y definición detallada de lo que el sistema debe hacer (requisitos funcionales y no funcionales).

Diseño: definición de la arquitectura del sistema, el modelo de datos, la estructura de la interfaz de usuario y las decisiones técnicas principales.

Implementación: construcción del sistema a partir del diseño, desarrollando el código fuente necesario.

Validación y pruebas: verificación de que el sistema cumple los requisitos establecidos, mediante pruebas funcionales y, en su caso, ajustes y correcciones.

Mantenimiento: corrección de errores encontrados tras la entrega y posibles pequeñas mejoras o refactorizaciones.

La característica principal de este modelo es su secuencialidad: idealmente, no se avanza a una fase nueva hasta haber cerrado la anterior. Esto proporciona una alta trazabilidad y un control claro del avance, lo que resulta especialmente adecuado en proyectos acotados, con un único desarrollador y un alcance relativamente estable, como es el caso de esta aplicación.

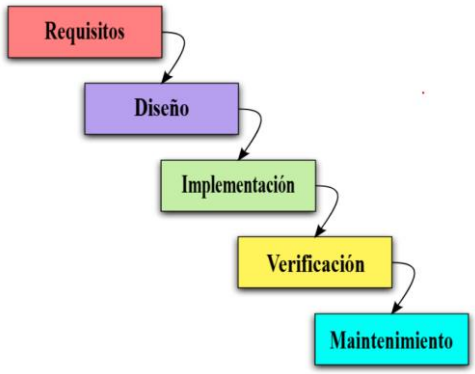

Figura 1: Modelo de desarrollo en cascada.

2.2.2. Aplicación del modelo en este proyecto

En este proyecto, el modelo en cascada se ha concretado en las siguientes fases:

- **Fase de análisis de requisitos**

En primer lugar, se identificó el problema a resolver: la dificultad para gestionar metas personales y colaborativas de forma centralizada y estructurada.

A partir de ahí, se definieron los requisitos principales del sistema, entre los que destacan:

- Gestión de usuarios.
- Creación y seguimiento de metas individuales y colaborativas.
- Diferenciación entre metas numéricas y metas basadas en tareas.
- Posibilidad de registrar aportaciones de progreso y asignar tareas a miembros concretos en metas compartidas.
- Necesidad de una interfaz atractiva y fácil de usar, que fomentara el uso continuo.

- **Fase de diseño**

A partir de los requisitos, se llevó a cabo el diseño del sistema en varios niveles:

- Diseño del modelo de datos, definiendo las tablas y relaciones necesarias en MySQL: usuarios, metas, tipos de meta, tareas, registros de progreso y participantes en metas colaborativas.
- Diseño de la arquitectura, separando la aplicación en capas de presentación (HTML, CSS, JavaScript), lógica de negocio (PHP) y persistencia (MySQL).
- Diseño de la interfaz de usuario, bocetando las principales pantallas: listado de metas, detalle de cada meta, gestión de tareas, vista de progreso y formularios de autenticación.

En esta fase se definieron también criterios de diseño visual (estructura de tarjetas, distribución de elementos, estilo de botones, etc.) para lograr una apariencia moderna y coherente.

- **Fase de implementación**

Una vez definido el diseño, se procedió a la construcción de la aplicación:

- Implementación del back-end en PHP, incluyendo la gestión de sesiones, control de acceso, controladores para operaciones sobre metas, tareas y colaboraciones, y la integración con la base de datos MySQL.

- Implementación del front-end con HTML5, CSS3 y JavaScript, dando forma a la interfaz visual definida en la fase anterior y conectándola con la lógica del servidor mediante formularios y peticiones.

- Desarrollo de las funcionalidades específicas para las metas individuales (numéricas y de tareas), metas colaborativas, con registro de aportaciones y asignación de tareas a miembros concretos, cálculo del porcentaje de progreso global y del porcentaje aportado por cada integrante en metas colaborativas.

- **Fase de validación y pruebas**

Una vez implementadas las funcionalidades principales, se llevaron a cabo pruebas centradas en los flujos más críticos:

- Registro e inicio de sesión de usuarios.
- Creación, edición y eliminación de metas individuales y colaborativas.
- Registro de avances numéricos y actualización de tareas.
- Cálculo y visualización del progreso total y de las contribuciones individuales.

Las pruebas se realizaron en un entorno local de desarrollo, lo que permitió detectar y corregir errores, ajustar algunos aspectos del diseño de la base de datos (por ejemplo, tipos de campos o relaciones) y refinar detalles de la interfaz para mejorar la experiencia de usuario.

- **Fase de cierre y mantenimiento básico**

En la última fase, se llevó a cabo una fase de cierre en la que se revisó y limpió el código, eliminando dependencias innecesarias y comentando las partes más relevantes.

Finalmente se identificaron posibles mejoras futuras (recordatorios automáticos, integración con calendarios externos, despliegue en servidor público, etc.).

Aunque no se ha realizado un mantenimiento prolongado en un entorno de producción, esta fase sirvió para dejar el proyecto en un estado estable, preparado para su despliegue y evolución posterior.

En conjunto, la aplicación del modelo en cascada ha proporcionado una estructura clara para el desarrollo del proyecto, permitiendo avanzar de forma ordenada desde el análisis de requisitos hasta la validación final, con un control razonable sobre el alcance y la calidad del resultado obtenido.

2.3. Tecnologías y Herramientas Utilizadas

Para el desarrollo del proyecto, se utilizaron las siguientes tecnologías y herramientas con el objetivo de obtener un sistema mantenible, fácil de desplegar y con una interfaz atractiva para el usuario:

2.3.1. Front-end

HTML5: HTML5 es el estándar actual para la estructuración de contenido en la web. Proporciona etiquetas semánticas que permiten organizar la página en bloques lógicos (cabeceras, navegación, contenido principal, secciones, pie de página, etc.). En la aplicación, HTML5 se ha utilizado para definir la estructura de todas las vistas: formularios de registro e inicio de sesión, listados de metas, vistas de detalle, paneles de progreso y secciones de gestión de tareas, sirviendo de base para la capa de estilos.

Figura 2: HTML5.

CSS3: CSS3 es el lenguaje utilizado para definir la presentación visual de los documentos HTML. Permite controlar propiedades como colores, tipografías, márgenes, tamaños, distribución de elementos y efectos visuales. En el proyecto se ha empleado CSS3 para construir una interfaz moderna, atractiva y responsive, de forma que la aplicación se adapte razonablemente a distintos tamaños de pantalla. Se ha prestado especial atención en ofrecer una interfaz agradable que fomente el uso continuado.

Figura 3: CSS3.

JavaScript: JavaScript es el lenguaje de programación estándar para proporcionar de interactividad a las páginas web en el lado del cliente. Permite gestionar eventos y realizar operaciones lógicas. En la aplicación se ha utilizado para validar ciertos formularios en el navegador, mejorar la respuesta de la interfaz y aportar mejoras de usabilidad que complementan la lógica implementada en PHP.

Figura 4: JavaScript.

SVG: SVG (Scalable Vector Graphics) es un formato de gráficos vectoriales basado en XML que permite definir imágenes escalables sin pérdida de calidad. A diferencia de formatos rasterizados como JPEG o PNG, los SVG se adaptan bien a diferentes resoluciones y densidades de pantalla. En este proyecto se han empleado SVG para algunos elementos gráficos y decorativos de la interfaz, como iconos o pequeños detalles visuales, contribuyendo a un acabado más nítido y profesional.

Figura 5: SVG.

2.3.2. Back-end

PHP: PHP se ha utilizado como lenguaje de programación en el lado del servidor. Al ejecutarse integrado con el servidor Apache, permite generar contenido HTML dinámico, procesar peticiones HTTP, gestionar sesiones de usuario y comunicarse con la base de datos mediante extensiones específicas. En este proyecto, PHP implementa la lógica de negocio de la aplicación: autenticación de usuarios, gestión de metas individuales y colaborativas, registro de avances, asignación de tareas y cálculo de porcentajes de contribución, además de incluir medidas básicas de seguridad como el uso de tokens CSRF y la validación de entradas.

Figura 6: PHP.

2.3.3. Sistema gestor de base de datos

MySQL: MySQL se ha utilizado como sistema gestor de base de datos relacional. Permite definir tablas, relaciones y restricciones de integridad, así como realizar consultas SQL para insertar, actualizar, eliminar y recuperar información. En la aplicación se ha diseñado un esquema de base de datos que incluye tablas para usuarios, metas, tipos de meta (numéricas y de tareas), tareas asociadas, registros de progreso y participantes en metas colaborativas. MySQL ha permitido realizar operaciones como el cálculo del progreso global de una meta, el porcentaje de aportación de cada integrante o la obtención de listas de metas pendientes y completadas.

Figura 7: MySQL.

phpMyAdmin: phpMyAdmin es una herramienta web que facilita la administración de bases de datos MySQL mediante una interfaz gráfica. Permite crear y modificar tablas, ejecutar consultas SQL, revisar registros y exportar o importar datos. Durante el desarrollo del proyecto se ha utilizado phpMyAdmin para definir el esquema de la base de datos, ajustar tipos de datos y relaciones, y comprobar que las operaciones realizadas desde la aplicación se reflejaban correctamente en las tablas correspondientes.

Figura 8: phpMyAdmin.

2.3.4. Entorno de servidor y ejecución

XAMPP: Como entorno de desarrollo local se ha utilizado XAMPP, un paquete que integra el servidor web Apache, el intérprete de PHP y el servidor de bases de datos MySQL, entre otros componentes. XAMPP permite levantar fácilmente un entorno completo de servidor en la máquina local, de modo que la aplicación web puede ejecutarse y probarse accediendo desde el navegador a través de http://localhost. Esta configuración ha facilitado el desarrollo y la depuración, simulando un entorno similar al de despliegue real.

Figura 9: XAMPP.

2.3.5. Entorno de desarrollo

Visual Studio Code: Visual Studio Code es un editor de código fuente ligero y extensible, que ofrece la integración con Git. En este proyecto se ha utilizado como entorno principal de desarrollo para editar los archivos PHP, HTML, CSS y JavaScript, organizar la estructura del proyecto y realizar tareas habituales de programación desde un único entorno. Su integración con Git y GitHub ha permitido, además, visualizar cambios y gestionar el repositorio sin salir del editor.

Figura 10: Visual Studio Code.

2.3.6. Control de versiones y repositorio

Git: Se ha utilizado Git como sistema de control de versiones distribuido. Esta herramienta permite llevar un historial detallado de los cambios realizados en el código, organizar el trabajo en commits pequeños y coherentes, crear ramas para nuevas funcionalidades y volver a versiones anteriores en caso de errores. Gracias a Git, ha sido posible mantener un desarrollo ordenado y reducir el riesgo de pérdida de información durante la evolución del proyecto.

Figura 11: Git.

GitHub: GitHub se ha empleado como repositorio remoto para alojar el código del proyecto. Además de actuar como copia de seguridad del repositorio local, su interfaz web facilita la visualización del historial de commits, la navegación por los archivos y la gestión de ramas. GitHub ha sido clave para sincronizar el código entre distintas sesiones de trabajo y conservar un registro centralizado del proyecto.

Figura 12: GitHub.

GitHub Desktop: Para interactuar con Git y GitHub de forma más cómoda se ha utilizado GitHub Desktop. Esta aplicación de escritorio proporciona una interfaz gráfica para realizar operaciones como clonar repositorios, crear commits, cambiar de rama o sincronizar cambios con el repositorio remoto, sin necesidad de recurrir a la línea de comandos. Esto ha simplificado la gestión del control de versiones y ha hecho más accesible el flujo de trabajo con Git.

Figura 13: GitHub Desktop.

3. Análisis y Diseño del Sistema

En esta parte se planteará el análisis detallado de los requisitos del sistema y el diseño arquitectónico de la aplicación web para la gestión de metas personales y colaborativas. El proceso de análisis y diseño se puede considerar una de las fases más importantes del desarrollo, ya que se establecen las bases técnicas y funcionales sobre las que se construye el resultado final.

Se comenzará con el análisis de los requisitos funcionales y no funcionales que debe cumplir el sistema, seguido del diseño de la arquitectura. Después, se describirá el modelo de datos propuesto, el diseño de las principales funcionalidades y, por último, los diferentes aspectos del diseño de la interfaz.

Esta documentación sirve como guía técnica para la implementación del sistema y como referencia para futuras ampliaciones o modificaciones de la aplicación, tales como nuevas funcionalidades, mejoras en el rendimiento o la adaptación a otros entornos de despliegue.

Como se vio, el desarrollo del proyecto ha seguido un modelo en cascada, de modo que la fase de análisis y diseño se completó antes de proceder a la implementación, asegurando una base sólida y bien definida para el desarrollo de la aplicación.

3.1. Descripción del problema

Es poco frecuente encontrar a alguien que no tenga alguna meta u objetivo personal en mente. Vivimos en una sociedad llena de inquietudes, con el deseo constante de mejorar y evolucionar en distintos ámbitos de nuestra vida.

Es habitual que, en fechas señaladas, como Nochevieja, aparezcan declaraciones como: “Este año mi objetivo es…”. Sin embargo, resulta igual de común llegar al final de ese año y comprobar que muchas de esas metas no se han cumplido.

Más allá de la falta de tiempo o de motivación, estas metas suelen gestionarse de forma dispersa. Se apuntan en notas sueltas, hojas de cálculo o aplicaciones genéricas que no están pensadas para hacer un seguimiento real del avance. Esto hace difícil saber cuánto se ha progresado, qué parte del objetivo sigue pendiente y, cuando la meta es compartida, qué está aportando cada miembro del grupo y qué tareas tiene asignadas cada uno.

Problema: Actualmente no existe una herramienta específica que centralice la gestión de metas personales y colaborativas, permitiendo combinar metas numéricas y basadas en tareas y ofreciendo una visión clara del progreso global y de la aportación de cada participante.

Objetivo: Diseñar y desarrollar una aplicación web que permita definir y gestionar metas individuales y colaborativas, tanto numéricas como de tareas, y que muestre en un panel visual e intuitivo el progreso alcanzado, el pendiente y la contribución de cada integrante en las metas colaborativas.

Solución existente	Características	Limitaciones
Uso de papel y bolígrafo	Simplicidad, accesible para cualquiera y flexible para anotar metas en cualquier formato.	No permite análisis cuantitativo, no cuenta con recordatorios y es difícil visualizar el progreso a largo plazo.
Aplicaciones de notas	Permite organizar listas de objetivos y añadir descripciones	
Hojas de cálculo	Posibilidad de registrar datos numéricos, aplicar fórmulas y generar gráficos personalizados.	Requiere conocimientos previos, poco intuitivo para usuarios no técnicos y no genera recordatorios automáticos.

Tabla 1: Características y limitaciones de las soluciones existentes.

3.1.1. Problemas y objetivos de negocio

BP: La gestión de metas personales y colaborativas se gestionan con herramientas genéricas y dispersas, que no permiten un seguimiento estructurado ni una visualización clara del progreso global y de las aportaciones individuales.

BO: Desarrollar una aplicación web que centralice la gestión de metas personales y colaborativas, permitiendo trabajar con metas numéricas y basadas en tareas, y ofreciendo un panel visual e intuitivo donde se muestre el avance global, el progreso pendiente y la contribución de cada integrante en las metas colaborativas.

BP1: Dificultad para seguir el progreso de metas personales y colaborativas.

BO1: Proporcionar un seguimiento cuantitativo y visual de las metas.

BP1.1: Los usuarios no disponen de una forma clara de medir su progreso en términos cuantitativos.

BO1.1: Implementar metas numéricas con un objetivo total y un sistema de registro de aportaciones que calcule el porcentaje de avance.

BP1.2: El avance basado en tareas suele quedar disperso en listas sin relación directa con la meta global.

BO1.2: Implementar metas de tareas, estructuradas como listas de subtareas, de manera que el porcentaje de progreso se derive del número de tareas completadas.

BP2: Falta de transparencia y coordinación en metas colaborativas.

BO2: Facilitar la gestión y el seguimiento de metas compartidas.

BP2.1: En metas de grupo no se sabe con precisión qué ha aportado cada integrante.

BO2.1: Registrar las aportaciones de cada usuario en metas colaborativas y mostrar el porcentaje de contribución individual respecto al progreso total.

BP2.2: Las tareas en objetivos compartidos no tienen responsables claramente definidos.

BO2.2: Permitir que las tareas de una meta colaborativa se asignen a un integrante concreto del equipo y mostrar su estado.

BP3: Herramientas existentes poco atractivas o complejas para el usuario.

BO3: Ofrecer una interfaz simple, coherente y visualmente atractiva que fomente el uso continuado.

BP3.1: Interfaces poco cuidadas reducen la motivación para actualizar y consultar las metas.

BO3.1: Diseñar una interfaz moderna basada en tarjetas, indicadores de progreso y una paleta de colores coherente, que resulte agradable y fácil de usar.

BP3.2: Falta de vistas resumidas que ayuden a priorizar metas.

BO3.2: Incluir una página principal a modo de panel de resumen, donde se muestren metas próximas a vencer y metas aún no completadas para facilitar la toma de decisiones del usuario.

3.1.2. Features del sistema

A partir de los objetivos de negocio anteriormente, se han identificado una serie de funcionalidades principales (features) que estructuran el comportamiento de la aplicación. Cada feature agrupa un conjunto de casos de uso y servirá para la elaboración de los diagramas de casos de uso correspondientes, y así tener claras todas las acciones que el usuario final va a poder hacer.

- **F1: Gestión de usuarios**

Esta feature recoge toda la funcionalidad relacionada con el acceso al sistema.

Permite que nuevos usuarios se registren en la aplicación y que los usuarios existentes inicien y cierren sesión de forma segura. La gestión de usuarios incluye la validación de credenciales, la creación de sesiones y el control de acceso a las distintas páginas, de modo que solo los usuarios autenticados puedan crear, consultar y modificar sus metas personales o colaborar en metas de grupo.

El correcto funcionamiento de esta feature es esencial, ya que el resto de las funcionalidades se apoyan en la identificación del usuario conectado.

- **F2: Gestión de metas personales**

Esta feature se centra en las metas que pertenecen únicamente a un usuario.

Permite crear, editar, consultar y eliminar metas individuales, distinguiendo entre metas numéricas (orientadas a alcanzar un valor cuantitativo) y metas basadas en tareas (orientadas a completar una lista de subtareas). El usuario puede definir los datos principales de la meta (título, descripción, fecha límite, tipo de meta y unidad de medida en el caso de metas numéricas) y, posteriormente, actualizar su progreso, ya sea registrando nuevas aportaciones numéricas o marcando tareas como completadas.

El objetivo de esta feature es ofrecer un mecanismo estructurado para que cada usuario pueda gestionar de forma clara sus propios objetivos.

- **F3: Gestión de metas colaborativas**

La feature de gestión de metas colaborativas permite la creación y el seguimiento de metas compartidas entre varios usuarios.

Permite que un usuario actúe como propietario de una meta colaborativa, definiendo sus datos principales y añadiendo participantes que podrán contribuir a su progreso. La aplicación ofrece vistas específicas donde se muestra el progreso global de la meta y el porcentaje de contribución de cada integrante, calculado a partir de las aportaciones registradas.

Esta feature da respuesta a la necesidad de coordinar objetivos de grupo y tener una visión de cómo contribuye cada miembro al cumplimiento de la meta.

- **F4: Gestión de tareas en metas personales**

Esta feature se centra en las tareas asociadas a metas individuales.

Permite que el usuario descomponga una meta personal de tipo “lista de tareas” en subtareas más pequeñas. Incluye la creación, edición y eliminación de tareas, así como cambiar su estado. La aplicación calcula el porcentaje de progreso de la meta en función del número de tareas completadas respecto al total definido, en base a una duración estimada que definirá el usuario en cada tarea.

De este modo, el usuario puede organizar su trabajo en pasos concretos y visualizar el avance total.

- **F5: Gestión de tareas en metas colaborativas**

Esta feature amplía el concepto de tareas al contexto de metas compartidas.

Permite crear, editar y eliminar tareas dentro de una meta colaborativa, pero añade la posibilidad de asignar cada tarea a un integrante concreto del equipo. De esta forma, cada tarea tiene un responsable claro, lo que facilita la distribución del trabajo.

La aplicación muestra el estado de las tareas y la persona asignada a cada una, lo que aporta una mejor coordinación y transparencia en el reparto de responsabilidades.

- **F6: Visualización y progreso**

La última feature agrupa las funcionalidades de visualización global de la información.

Incluye la página principal a modo de panel de resumen, donde se listan las metas y tareas del usuario que están más próximas a su fecha límite y que aún no han sido completadas. Para cada meta se muestra su porcentaje de progreso, que son calculadas a partir de las aportaciones numéricas o del número de tareas completadas.

Esta feature responde al objetivo de ofrecer una vista compacta y visualmente atractiva que facilite la toma de decisiones sobre qué meta priorizar en cada momento.

3.1.3. Diagrama de problemas de negocio

Para comprender la relación entre los problemas de negocio, los objetivos de negocio definidos y las funcionalidades principales del sistema, se ha elaborado un diagrama de trazabilidad. En él se muestra cómo, a partir de cada problema, deriva varios subproblemas de negocio con sus correspondientes objetivos, y cómo estos, a su vez, se materializan en features concretas de la aplicación.

De este modo, se garantiza que cada funcionalidad implementada responde a una necesidad identificada y contribuye de forma directa al propósito del proyecto.

La feature F1 (Gestión de usuarios) se considera una funcionalidad de soporte necesaria para el resto de features, por lo que no se vincula a un objetivo de negocio específico en el diagrama.

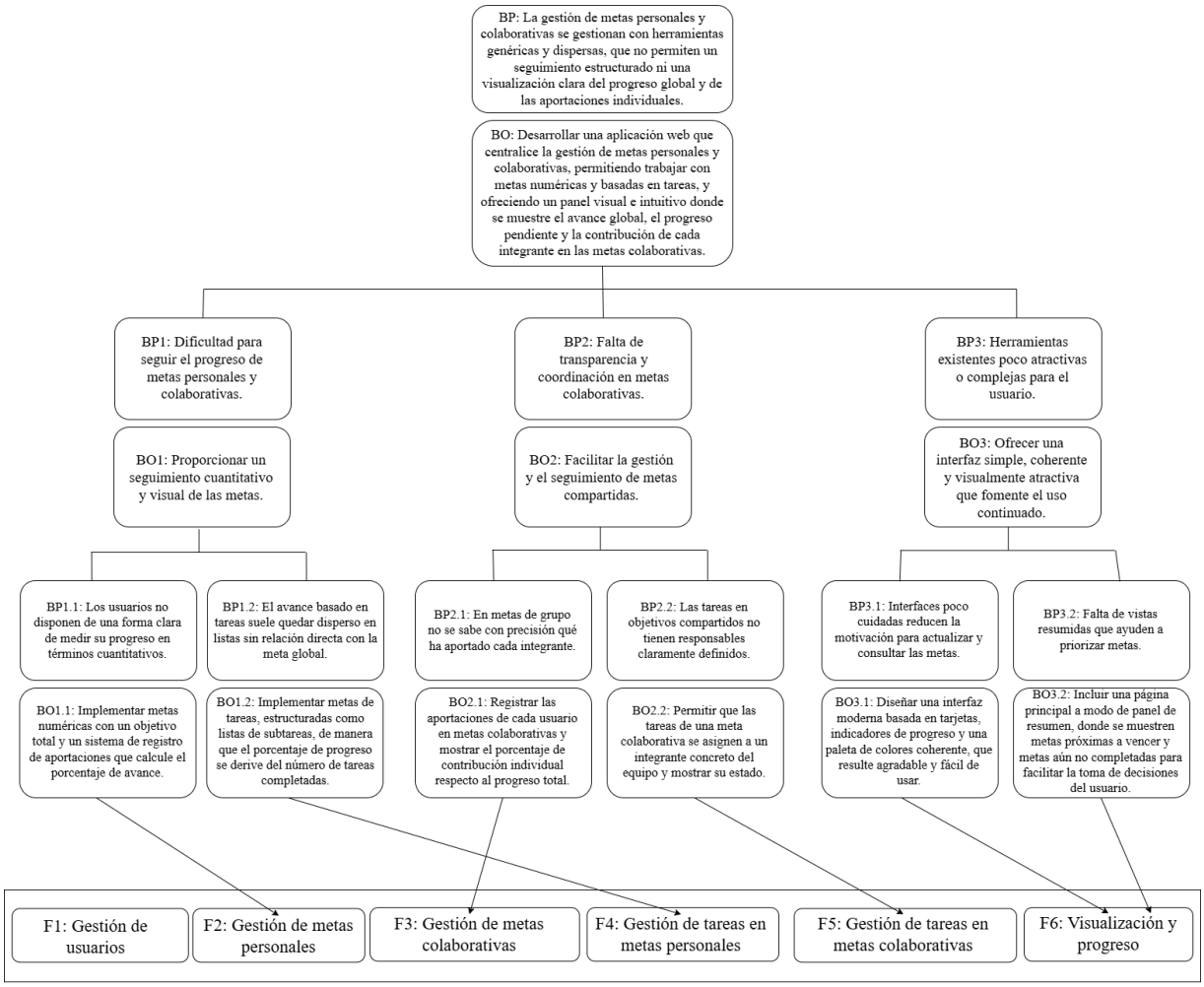

Figura 14: Diagrama de problemas de negocio.

3.2. Análisis de Requisitos

El análisis de requisitos permite definir qué debe hacer el sistema y en qué condiciones debe hacerlo. En este apartado podemos ver de forma estructurada las funcionalidades que ofrecerá la aplicación, así como las características de calidad que se esperan de ella. Para ello, los requisitos se han clasificado en requisitos funcionales y requisitos no funcionales.

3.2.1. Requisitos Funcionales

Los requisitos funcionales se han organizado por módulos, alineados con las features definidas en el apartado anterior.

RF-01. Gestión de usuarios (F1)

ID del requisito	Descripción
RF-01.1	El sistema debe permitir a los usuarios registrarse proporcionando nombre, correo electrónico y una contraseña.
RF-01.2	El sistema debe validar que el correo electrónico introducido en el registro no está ya asociado a otro usuario.
RF-01.3	El sistema debe permitir a los usuarios iniciar sesión introduciendo su correo electrónico y contraseña.
RF-01.4	El sistema debe mantener una sesión activa para el usuario autenticado mientras navega por la aplicación.
RF-01.5	El sistema debe permitir a los usuarios cerrar sesión y finalizar su sesión activa.

Tabla 2: RF-01. Gestión de usuarios (F1).

RF-02. Gestión de metas personales (F2)

ID del requisito	Descripción
RF-02.1	El sistema debe permitir al usuario crear metas personales indicando, título, descripción, y fecha límite.
RF-02.2	El sistema debe permitir seleccionar el tipo de meta personal entre meta numérica y meta de tareas.
RF-02.3	En metas numéricas personales, el sistema debe permitir definir una cantidad objetivo y una unidad de medida (horas, entregas, unidades, etc).
RF-02.4	El sistema debe permitir editar los datos de una meta personal existente.
RF-02.5	El sistema debe permitir eliminar una meta personal.
RF-02.6	El sistema debe permitir marcar una meta personal como completada.
RF-02.7	En metas numéricas personales, el sistema debe permitir registrar aportaciones incrementales al valor actual.
RF-02.8	En metas numéricas personales, el sistema debe calcular y mostrar el porcentaje de progreso en función de la cantidad alcanzada y el objetivo definido.

Tabla 3: RF-02. Gestión de metas personales (F2).

RF-03. Gestión de metas colaborativas (F3)

ID del requisito	Descripción
RF-03.1	El sistema debe permitir al usuario crear metas colaborativas indicando, título, descripción, fecha límite y tipo de meta.
RF-03.2	El sistema debe permitir al propietario de una meta colaborativa añadir participantes que estén registrados en la aplicación.

RF-03.3	En metas colaborativas numéricas, el sistema debe permitir registrar aportaciones numéricas realizadas por cualquier participante autorizado.
RF-03.4	En metas colaborativas numéricas, el sistema debe calcular y mostrar el progreso global en función de la suma de las aportaciones y el objetivo definido.
RF-03.5	En metas colaborativas numéricas, el sistema debe calcular y mostrar el porcentaje de contribución de cada participante respecto al progreso total.
RF-03.6	El sistema debe permitir al usuario propietario de una meta colaborativa eliminar participantes de dicha meta.

Tabla 4: RF-03. Gestión de metas colaborativas (F3).

RF-04. Gestión de tareas en metas personales (F4)

ID del requisito	Descripción
RF-04.1	El sistema debe permitir crear tareas asociadas a una meta personal de tipo tareas indicando nombre, descripción, fecha límite y duración aproximada.
RF-04.2	El sistema debe permitir eliminar tareas asociadas a una meta personal.
RF-04.3	El sistema debe permitir marcar una tarea de una meta personal como completada o pendiente.
RF-04.4	El sistema debe calcular y mostrar el porcentaje de progreso de una meta personal de tareas en función del número de tareas completadas respecto al total definido, usando la duración aproximada de cada tarea.

Tabla 5: RF-04. Gestión de tareas en metas personales (F4).

RF-05. Gestión de tareas en metas colaborativas (F5)

ID del requisito	Descripción
RF-05.1	El sistema debe permitir crear tareas asociadas a una meta colaborativa de tipo tareas indicando nombre, descripción, fecha límite duración aproximada y a qué participante se asigna.
RF-05.2	El sistema debe permitir eliminar tareas asociadas a una meta colaborativa.
RF-05.3	El sistema debe permitir marcar como completada o pendiente una tarea asociada a una meta colaborativa.
RF-05.4	El sistema debe calcular y mostrar el porcentaje de progreso de la meta colaborativa de tareas en función del número de tareas completadas respecto al total definido, usando la duración aproximada de cada tarea.

Tabla 6: RF-05. Gestión de tareas en metas colaborativas (F5).

RF-06. Visualización de progreso y panel de resumen (F6)

ID del requisito	Descripción
RF-06.1	El sistema debe mostrar al usuario un listado de sus metas personales y colaborativas.
RF-06.2	El sistema debe mostrar, para cada meta del listado, el porcentaje de progreso actual.
RF-06.3	El sistema debe permitir acceder desde el listado a la vista detallada de cada meta.
RF-06.4	El sistema debe resaltar en el panel las metas que estén próximas a su fecha límite.
RF-06.5	El sistema debe permitir al usuario entrar en su página de perfil donde queden reflejados sus datos (Nombre, correo y fecha de la creación del usuario), además de un listado con todas las metas que han sido completadas, tanto individuales como colaborativas.

RF-06.5	El sistema debe proporcionar al usuario un calendario únicamente visual para poder gestionar de mejor manera las fechas limite de las diferentes metas y tareas

Tabla 7: RF-06. Visualización de progreso y panel de resumen (F6).

3.2.2. Requisitos No Funcionales

Los requisitos no funcionales definen las características de calidad que debe cumplir el sistema. En este proyecto se ha prestado especial atención en la usabilidad y en el diseño visual de la interfaz, sin descuidar aspectos como el rendimiento, la seguridad o la mantenibilidad.

RNF-01. Usabilidad

ID del requisito	Descripción
RNF-01.1	La interfaz debe mantener una estructura coherente en todas las pantallas, reutilizando la misma disposición de cabecera, navegación, contenido principal y pie.
RNF-01.2	Las acciones principales del usuario (crear meta, consultar metas, actualizar progreso) no deben requerir más de tres clics desde la pantalla inicial tras el inicio de sesión.
RNF-01.3	Los formularios deben incluir etiquetas y campos alineados, de forma que el usuario identifique fácilmente la información requerida.
RNF-01.4	El sistema debe mostrar mensajes de error y validación junto a los campos, explicando la causa del fallo.
RNF-01.5	La nomenclatura utilizada en la interfaz (etiquetas, títulos, botones) debe ser consistente en todas las pantallas.

RNF-01.6	La aplicación debe mantener el estado del usuario durante la sesión (usuario autenticado, selección de meta, etc.) mientras no cierre sesión explícitamente.
RNF-01.7	Las acciones de eliminar deben requerir una confirmación explícita por parte del usuario antes de ejecutarse.

Tabla 8: RNF-01. Usabilidad.

RNF-02. Diseño visual

ID del requisito	Descripción
RNF-02.1	La aplicación debe utilizar una paleta de colores coherente y limitada, reutilizando los mismos colores principales, secundarios y de acento en todas las pantallas.
RNF-02.2	Las metas deben mostrarse mediante tarjetas que incluyan, al menos, el título, la fecha límite, el tipo de meta y el porcentaje de progreso.
RNF-02.3	El progreso de cada meta debe representarse visualmente mediante barras de progreso u otros indicadores gráficos equivalentes.
RNF-02.4	Deben utilizarse iconos consistentes para acciones frecuentes (editar, eliminar, completar), favoreciendo su reconocimiento rápido por parte del usuario.
RNF-02.5	La distribución de elementos en las pantallas debe seguir una estructura alineada y con espaciado suficiente, evitando la saturación visual y mejorando la legibilidad.

Tabla 9: RNF-02. Diseño visual.

RNF-03. Rendimiento

ID del requisito	Descripción
RNF-03.1	El sistema debe cargar la pantalla de inicio de sesión y el panel principal de metas en menos de 3 segundos en el entorno local de desarrollo.
RNF-03.2	Las operaciones de consulta del listado de metas de un usuario deben ejecutarse con un número acotado de consultas SQL, evitando accesos redundantes a la base de datos.
RNF-03.3	El cálculo de los porcentajes de progreso de las metas (numéricas y de tareas) no debe producir retrasos perceptibles en la carga de las vistas correspondientes.

Tabla 10: RNF-03. Rendimiento.

RNF-04. Seguridad

ID del requisito	Descripción
RNF-04.1	El sistema debe restringir el acceso a las funcionalidades de gestión de metas y tareas únicamente a usuarios autenticados mediante sesiones.
RNF-04.2	Las contraseñas de los usuarios deben almacenarse en la base de datos de forma cifrada utilizando las funciones seguras.
RNF-04.3	Todos los formularios que modifiquen datos (creación, edición o eliminación de metas y tareas) deben incluir un token CSRF que se valide en el servidor.

Tabla 11: RNF-04. Seguridad.

RNF-05. Escalabilidad

ID del requisito	Descripción
RNF-05.1	El sistema debe soportar un aumento en el número de usuarios y de metas sin requerir cambios en la arquitectura básica, siempre que el servidor disponga de recursos suficientes.
RNF-05.2	El diseño del modelo de datos debe permitir manejar un volumen creciente de registros (usuarios, metas, tareas, aportaciones) sin degradar de forma apreciable el tiempo de respuesta en las operaciones principales.

Tabla 12: RNF-05. Escalabilidad.

RNF-07. Mantenibilidad

ID del requisito	Descripción
RNF-07.1	El código debe seguir estándares de programación.
RNF-07.2	Los nombres de archivos, funciones y variables deben seguir una convención clara y descriptiva.
RNF-07.3	El modelo de datos debe permitir añadir nuevos tipos de metas o nuevos atributos sin necesidad de reestructurar completamente la base de datos existente.

Tabla 13: RNF-07. Mantenibilidad.

RNF-08. Compatibilidad

ID del requisito	Descripción
RNF-08.1	La aplicación debe funcionar en navegadores modernos (Chrome, Firefox, Safari, Edge).
RNF-08.2	El sistema debe ser compatible con diferentes tamaños de pantalla.

Tabla 14: RNF-08. Compatibilidad.

3.3. Modelado de casos de uso

Los casos de uso describen las interacciones entre los actores del sistema y las funcionalidades que ofrece la aplicación. A partir de los requisitos definidos, se han identificado los casos de uso principales y se han agrupado en módulos, alineados con las features del sistema: autenticación, gestión de metas personales, gestión de metas colaborativas, gestión de tareas y visualización del progreso.

Cada caso de uso se documenta mediante una ficha que incluye su código identificador, nombre, actor implicado, una breve descripción, las precondiciones necesarias, las postcondiciones esperadas y el flujo principal de ejecución. Esta información servirá como base tanto para el diseño detallado como para las pruebas posteriores.

3.3.1. Actores del sistema

En el sistema se ha identificado un único actor principal:

Usuario: persona registrada en la aplicación que puede iniciar sesión, crear y gestionar sus metas personales, participar en metas colaborativas creadas por otros usuarios, gestionar tareas asociadas a dichas metas y consultar el progreso global de sus objetivos.

Dentro de una meta concreta pueden distinguirse dos roles lógicos (propietario y colaborador), pero ambos siguen siendo el mismo actor "Usuario" desde el punto de vista de los casos de uso.

3.3.2. Casos de uso

- **F1: Gestión de usuarios**
    - CU-01: Iniciar sesión
    - CU-02: Registrarse
    - CU-03: Cerrar sesión

Código	CU-01
Nombre	Iniciar sesión
Actor	Usuario
Descripción	Permite al usuario acceder al sistema introduciendo su correo electrónico y contraseña válidos.
Precondiciones	El usuario debe estar registrado en la aplicación y disponer de credenciales válidas.
Postcondiciones	El usuario queda autenticado en el sistema y se redirige al panel principal de metas.
Flujo Principal	1) El usuario accede a la pantalla de inicio de sesión. 2) El usuario introduce su correo electrónico y contraseña. 3) El sistema valida las credenciales. 4) Si las credenciales son correctas, el sistema crea la sesión del usuario. 5) El sistema muestra el panel principal.

Tabla 15: CU-01: Iniciar sesión.

Código	CU-02
Nombre	Registrarse
Actor	Usuario
Descripción	Permite a un nuevo usuario crear una cuenta en la aplicación.
Precondiciones	El usuario no debe estar registrado previamente con el mismo correo electrónico.
Postcondiciones	Se crea un nuevo registro de usuario en la base de datos y el usuario puede iniciar sesión.
Flujo Principal	1) El usuario accede al formulario de registro. 2) Introduce los datos requeridos (nombre, correo y contraseña). 3) El sistema comprueba que el correo no existe. 4) El sistema crea el nuevo usuario en la base de datos. 5) El sistema muestra un mensaje de registro correcto y ofrece ir a la pantalla de inicio de sesión.

Tabla 16: CU-02: Registrarse.

Código	CU-03
Nombre	Cerrar sesión
Actor	Usuario
Descripción	Permite al usuario finalizar su sesión en la aplicación.
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	La sesión del usuario se termina.
Flujo Principal	1) El usuario pulsa el botón de cerrar sesión. 2) El sistema elimina los datos de la sesión activa. 3) El sistema redirige a la pantalla de inicio.

Tabla 17: CU-03: Cerrar sesión.

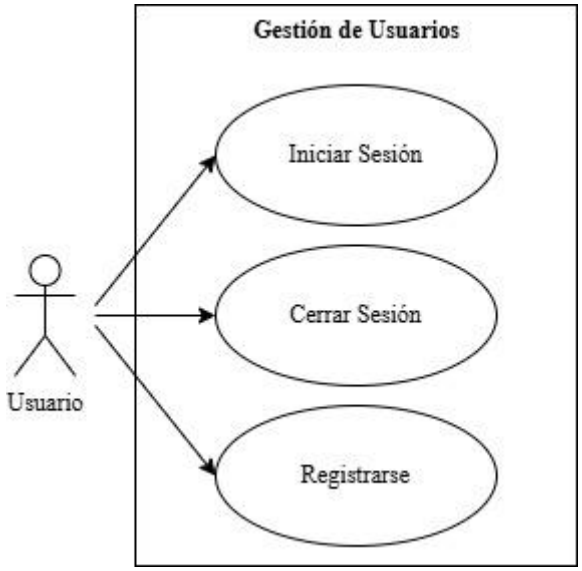

Figura 15: Diagrama de casos de uso: Gestión de Usuarios.

- **F2: Gestión de metas personales**
    - CU-04: Crear meta personal
    - CU-05: Editar meta personal
    - CU-06: Eliminar meta personal
    - CU-07: Mostrar detalles
    - CU-08: Actualizar progreso

Código	CU-04
Nombre	Crear meta personal
Actor	Usuario
Descripción	Permite al usuario crear una nueva meta personal.
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	Se registra una nueva meta personal asociada al usuario.

Flujo Principal	1) El usuario accede a la opción “Crear meta”. 2) Introduce título, descripción, fecha límite y tipo de meta (numérica o de tareas). 3) Si la meta es numérica, introduce también la cantidad objetivo, la cantidad inicial y la unidad de medida. 4) El sistema valida los datos introducidos. 5) El sistema guarda la meta en la base de datos y muestra la lista de metas actualizada.

Tabla 18: CU-04: Crear meta personal.

Código	**CU-05**
Nombre	Editar meta personal
Actor	Usuario
Descripción	Permite modificar los datos de una meta personal existente.
Precondiciones	El usuario debe estar autenticado y ser propietario de la meta personal.
Postcondiciones	Se actualizan los datos de la meta personal en la base de datos.
Flujo Principal	1) El usuario selecciona una meta personal de la lista. 2) Accede a la opción de editar. 3) Modifica los campos. 4) El sistema valida los datos. 5) El sistema guarda los cambios y muestra la meta actualizada.

Tabla 19: CU-05: Editar meta personal.

Código	CU-06
Nombre	Eliminar meta personal
Actor	Usuario
Descripción	Permite al usuario eliminar una meta personal y su información asociada.
Precondiciones	El usuario debe estar autenticado y ser propietario de la meta personal.
Postcondiciones	La meta personal y sus datos asociados dejan de estar disponibles en el sistema.
Flujo Principal	1) El usuario selecciona una meta personal de la lista. 2) Pulsa la opción de eliminar. 3) El sistema solicita confirmación. 4) El usuario confirma la eliminación. 5) El sistema elimina la meta y muestra la lista de metas actualizada.

Tabla 20: CU-06: Eliminar meta personal.

Código	CU-07
Nombre	Mostrar detalles de meta personal
Actor	Usuario
Descripción	Permite visualizar la información detallada de una meta personal.
Precondiciones	El usuario debe estar autenticado y ser propietario de la meta.
Postcondiciones	Ninguna, el usuario solo consulta información.

Flujo Principal	1) El usuario selecciona una meta personal de la lista. 2) El sistema recupera los datos de la meta, tareas y registros de progreso. 3) El sistema muestra la vista detallada con toda la información y el porcentaje de avance.

Tabla 21: CU-07: Mostrar detalles meta personal.

Código	**CU-08**
Nombre	Actualizar progreso de meta personal
Actor	Usuario
Descripción	Permite registrar una nueva aportación en una meta personal de tipo numérico.
Precondiciones	El usuario debe estar autenticado, ser propietario de la meta y la meta debe ser de tipo numérico.
Postcondiciones	Se añade un nuevo registro de aportación y se actualiza el progreso de la meta.
Flujo Principal	1) El usuario accede al detalle de una meta numérica personal. 2) Introduce la cantidad a añadir (o restar) al progreso actual. 3) El sistema valida la cantidad. 4) El sistema registra la aportación en la base de datos. 5) El sistema recalcula el porcentaje de progreso y actualiza la vista de la meta.

Tabla 22: CU-08: Actualizar progreso de meta personal.

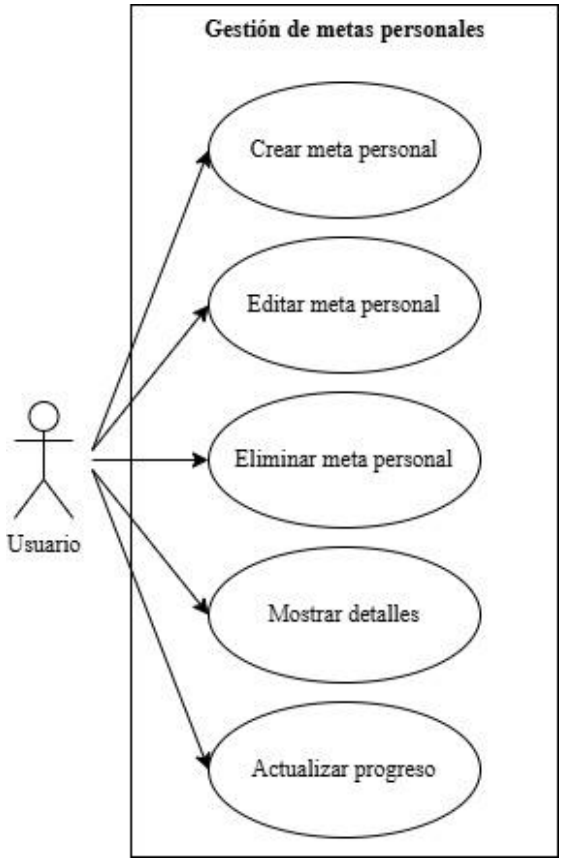

Figura 16: Diagrama de casos de uso: Gestión de metas personales.

- **F3: Gestión de metas colaborativas**
    - CU-09: Crear meta colaborativa
    - CU-10: Añadir participante
    - CU-11: Eliminar participante
    - CU-12: Cambiar rol participante
    - CU-13: Eliminar meta colaborativa
    - CU-14: Mostrar detalles
    - CU-15: Mostrar listado de miembros
    - CU-16: Actualizar progreso

Código	CU-09
Nombre	Crear meta colaborativa
Actor	Usuario
Descripción	Permite al usuario crear una meta que podrá ser compartida con otros participantes.
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	Se registra una nueva meta colaborativa con el usuario como propietario.
Flujo Principal	1) El usuario selecciona la opción de crear meta colaborativa. 2) Introduce título, descripción, fecha límite y tipo de meta (numérica o de tareas). 3) Si es numérica, define cantidad objetivo, cantidad inicial y unidad de medida. 4) El sistema valida los datos. 5) El sistema guarda la meta y la muestra en la lista de metas del propietario.

Tabla 23: CU-09: Crear meta colaborativa.

Código	CU-10
Nombre	Añadir participante a meta colaborativa
Actor	Usuario
Descripción	Permite al propietario de una meta colaborativa añadir participantes.
Precondiciones	El usuario debe estar autenticado y ser propietario de la meta colaborativa. Además de que los usuarios que desee añadir deben estar registrados en la aplicación.
Postcondiciones	Los usuarios seleccionados quedan registrados como participantes de la meta colaborativa.

Flujo Principal	1) El propietario accede al detalle de la meta colaborativa. 2) Selecciona la opción de añadir participantes. 3) Introduce los usuarios a añadir. 4) El sistema valida que existan y no estén ya añadidos. 5) El sistema registra la participación y actualiza la lista de participantes.

Tabla 24: CU-10: Añadir participante a meta colaborativa.

Código	**CU-11**
Nombre	Eliminar participante de la meta colaborativa
Actor	Usuario
Descripción	Permite al propietario eliminar un participante de la meta colaborativa.
Precondiciones	El usuario debe estar autenticado y ser propietario de la meta colaborativa.
Postcondiciones	El usuario eliminado deja de formar parte de los miembros de la meta y deja de tener acceso a los detalles ni puede participar en el progreso.
Flujo Principal	1) El propietario lista los miembros de la meta. 2) Pulsa la opción de eliminar. 3) El sistema solicita confirmación. 4) El usuario confirma. 5) El sistema elimina el miembro y actualiza el listado.

Tabla 25: CU-11: Eliminar participante de la meta colaborativa.

Código	CU-12
Nombre	Cambiar rol del participante de la meta colaborativa
Actor	Usuario
Descripción	Permite al propietario cambiar el rol de un participante de la meta colaborativa.
Precondiciones	El usuario debe estar autenticado y ser propietario de la meta colaborativa.
Postcondiciones	El usuario cambia de rol y las acciones que pueden ejercer dentro de la meta cambian.
Flujo Principal	1) El propietario lista los miembros de la meta. 2) Selecciona el nuevo rol del miembro. 3) El sistema solicita confirmación. 4) El usuario confirma. 5) El sistema actualiza el rol del miembro y actualiza el listado.

Tabla 26: CU-12: Cambiar rol del participante de la meta colaborativa.

Código	CU-13
Nombre	Eliminar meta colaborativa
Actor	Usuario
Descripción	Permite al propietario eliminar una meta colaborativa y todos sus datos asociados.
Precondiciones	El usuario debe estar autenticado y ser propietario de la meta colaborativa.
Postcondiciones	La meta colaborativa, sus tareas y registros de progreso dejan de estar disponibles para todos los participantes.

Flujo Principal	1) El propietario selecciona una meta colaborativa de la lista. 2) Pulsa la opción de eliminar. 3) El sistema solicita confirmación. 4) El usuario confirma. 5) El sistema elimina la meta y actualiza el listado de metas.

Tabla 27: CU-13: Eliminar meta colaborativa.

Código	**CU-14**
Nombre	Mostrar detalles
Actor	Usuario
Descripción	Permite visualizar la información detallada de una meta colaborativa.
Precondiciones	El usuario debe estar autenticado y ser propietario o participante de la meta.
Postcondiciones	Ninguna, el usuario solo consulta información.
Flujo Principal	1) El usuario selecciona una meta colaborativa de la lista. 2) El sistema muestra la vista detallada con toda la información y el porcentaje de avance.

Tabla 28: CU-14: Mostrar detalles meta colaborativa.

Código	**CU-15**
Nombre	Mostrar listado de miembros de meta colaborativa
Actor	Usuario
Descripción	Permite ver la lista de participantes de una meta colaborativa.

Precondiciones	El usuario debe estar autenticado y ser propietario o participante de la meta colaborativa.
Postcondiciones	Ninguna; el usuario solo consulta información.
Flujo Principal	1) El usuario accede al detalle de la meta colaborativa. 2) Selecciona la opción de ver participantes. 3) El sistema recupera la lista de participantes. 4) El sistema muestra el listado con los nombres de los miembros de la meta y sus roles.

Tabla 29: CU-15: Mostrar listado de miembros de meta colaborativa.

Código	**CU-16**
Nombre	Actualizar progreso de meta colaborativa
Actor	Usuario
Descripción	Permite registrar una nueva aportación en una meta colaborativa de tipo numérico.
Precondiciones	El usuario debe estar autenticado, ser propietario o participante de la meta y la meta debe ser de tipo numérico.
Postcondiciones	Se añade un nuevo registro de aportación asociado al usuario y se actualiza el progreso global de la meta.
Flujo Principal	1) El usuario accede al detalle de una meta colaborativa numérica. 2) Introduce la cantidad que desea aportar. 3) El sistema valida la cantidad. 4) El sistema registra la aportación asociándola al usuario. 5) El sistema recalcula el porcentaje de progreso total y el porcentaje de contribución de cada participante.

Tabla 30: CU-16: Actualizar progreso de meta colaborativa.

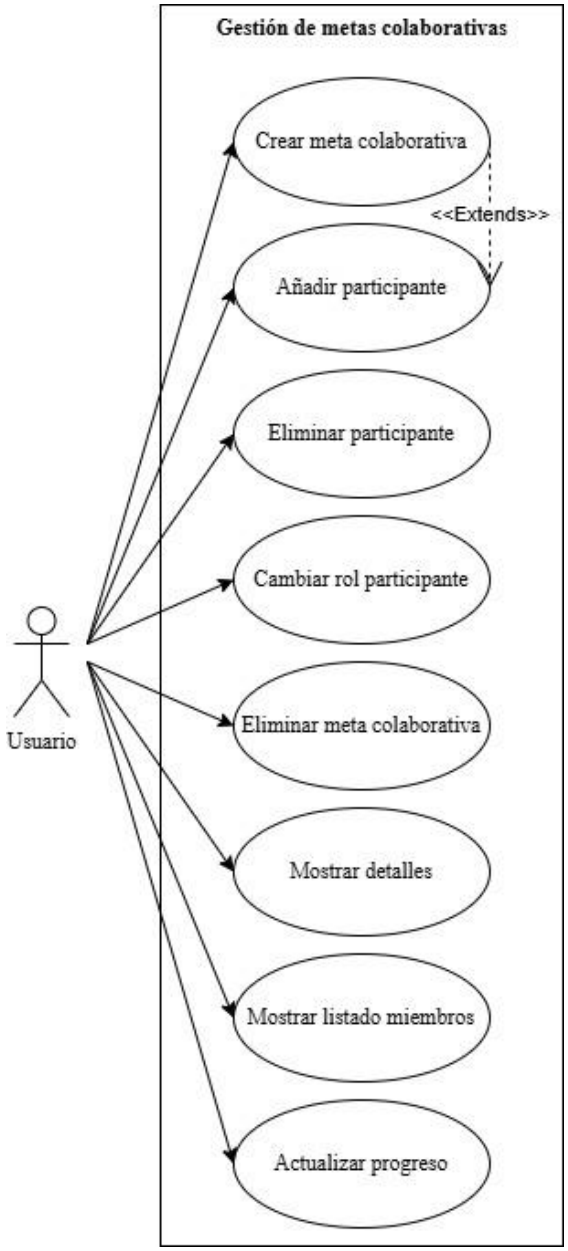

Figura 17: Diagrama de casos de uso: Gestión de metas colaborativas.

- **F4: Gestión de tareas en metas personales**
    - CU-17: Crear tarea en meta personal
    - CU-18: Eliminar tarea en meta personal
    - CU-19: Marcar tarea de meta personal como completada

Código	CU-17
Nombre	Crear tarea en meta personal
Actor	Usuario
Descripción	Permite añadir una nueva tarea a una meta personal de tipo tareas.
Precondiciones	El usuario debe estar autenticado, ser propietario de la meta y la meta debe ser de tipo tareas.
Postcondiciones	Se registra una nueva tarea asociada a la meta personal.
Flujo Principal	1) El usuario accede al detalle de una meta personal de tareas. 2) Selecciona la opción de añadir tarea. 3) Introduce el nombre, una descripción y una duración aproximada, con la que posteriormente se hará el cálculo del progreso total. 4) El sistema valida los datos. 5) El sistema guarda la tarea y actualiza la lista de tareas de la meta.

Tabla 31: CU-17: Crear tarea en meta personal..

Código	CU-18
Nombre	Eliminar tarea en meta personal
Actor	Usuario
Descripción	Permite eliminar una tarea asociada a una meta personal de tipo tareas.
Precondiciones	El usuario debe estar autenticado y ser propietario de la meta.
Postcondiciones	La tarea se elimina y actualiza el porcentaje total de la meta.

Flujo Principal	1) El usuario visualiza la lista de tareas de la meta personal. 2) Selecciona la tarea a eliminar. 3) El sistema solicita confirmación. 4) El usuario confirma. 5) El sistema elimina la tarea y actualiza la lista y el porcentaje de progreso.

Tabla 32: CU-18: Eliminar tarea en meta personal.

Código	CU-19
Nombre	Marcar tarea de meta personal como completada
Actor	Usuario
Descripción	Permite cambiar el estado de una tarea de una meta personal entre pendiente y completada.
Precondiciones	El usuario debe estar autenticado y ser propietario de la meta.
Postcondiciones	El estado de la tarea se actualiza y el porcentaje de progreso de la meta se recalcula.
Flujo Principal	1) El usuario visualiza la lista de tareas de la meta personal. 2) Marca una tarea como completada (o la desmarca). 3) El sistema actualiza el estado de la tarea. 4) El sistema recalcula el progreso de la meta y actualiza la vista.

Tabla 33: CU-19: Marcar tarea de meta personal como completada.

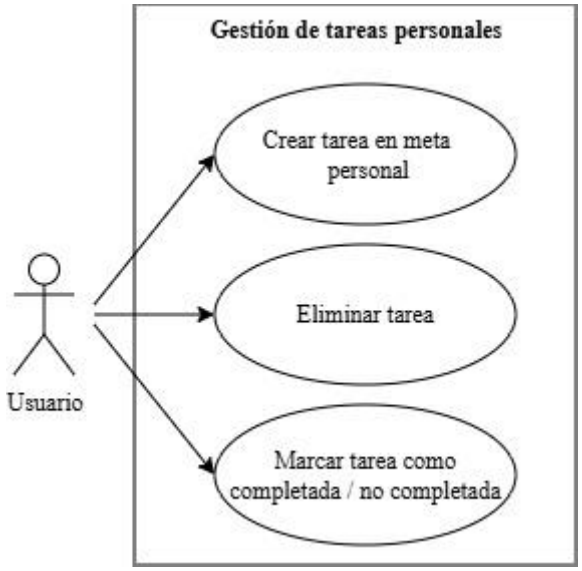

Figura 18: Diagrama de casos de uso: Gestión de tareas en metas personales.

- **F5: Gestión de tareas en metas colaborativas**
    - CU-20: Crear tarea en meta colaborativa
    - CU-21: Eliminar tarea en meta colaborativa
    - CU-22: Asignar tarea a un participante
    - CU-23: Marcar tarea de meta colaborativa como completada

Código	CU-20
Nombre	Crear tarea en meta colaborativa
Actor	Usuario
Descripción	Permite añadir una nueva tarea a una meta colaborativa de tipo tareas.
Precondiciones	El usuario debe estar autenticado y tener permisos para gestionar la meta colaborativa de tareas.
Postcondiciones	Se registra una nueva tarea asociada a la meta colaborativa.

Flujo Principal	1) El usuario accede al detalle de una meta colaborativa de tareas. 2) Selecciona la opción de añadir tarea. 3) Introduce los datos de la tarea. 4) Asigna la tarea a un miembro de la meta. 4) El sistema valida la información. 5) El sistema guarda la tarea y actualiza la lista de tareas de la meta.

Tabla 34: CU-20: Crear tarea en meta colaborativa.

Código	**CU-21**
Nombre	Eliminar tarea en meta colaborativa
Actor	Usuario
Descripción	Permite eliminar una tarea de una meta colaborativa de tipo tareas.
Precondiciones	El usuario debe estar autenticado y tener permisos para gestionar la meta colaborativa.
Postcondiciones	La tarea se elimina y deja de tenerse en cuenta para el cálculo del progreso de la meta.
Flujo Principal	1) El usuario accede al detalle de la meta colaborativa. 2) Selecciona la tarea a eliminar. 3) El sistema solicita confirmación. 4) El usuario confirma. 5) El sistema elimina la tarea y actualiza la lista de tareas y el porcentaje de progreso.

Tabla 35: CU-21: Eliminar tarea en meta colaborativa.

Código	CU-22
Nombre	Asignar tarea a un participante
Actor	Usuario
Descripción	Permite asignar una tarea de una meta colaborativa a uno de los participantes.
Precondiciones	El usuario debe estar autenticado, ser propietario de la meta colaborativa y la meta debe tener participantes registrados.
Postcondiciones	La tarea queda asociada a un participante concreto de la meta colaborativa.
Flujo Principal	1) El propietario accede a la lista de tareas de la meta colaborativa. 2) Selecciona una tarea. 3) Selecciona un participante de la lista de miembros. 4) El sistema registra la asignación. 5) El sistema muestra la tarea con el responsable asignado.

Tabla 36: CU-22: Asignar tarea a un participante.

Código	CU-23
Nombre	Marcar tarea de meta colaborativa como completada
Actor	Usuario
Descripción	Permite cambiar el estado de una tarea de una meta colaborativa entre pendiente y completada.
Precondiciones	El usuario debe estar autenticado, ser participante de la meta y tener permisos para modificar la tarea.
Postcondiciones	El estado de la tarea se actualiza y el progreso de la meta colaborativa se recalcula.

Flujo Principal	1) El usuario visualiza la lista de tareas de la meta colaborativa. 2) Marca una tarea como completada (o la desmarca). 3) El sistema actualiza el estado de la tarea. 4) El sistema recalcula el progreso de la meta y actualiza la vista.

Tabla 37: CU-23: Marcar tarea de meta colaborativa como completada.

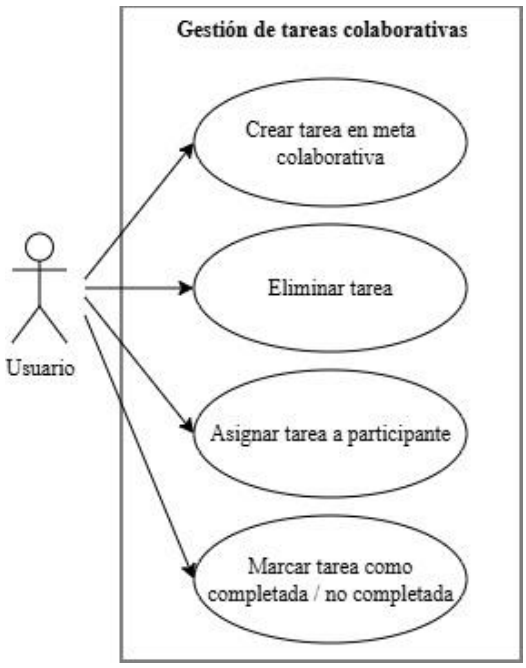

Figura 19: Diagrama de casos de uso: Gestión de tareas en metas colaborativas.

- **F6: Visualización de progreso**
    - CU-24: Mostrar calendario
    - CU-25: Mostrar panel de próximas metas
    - CU-26: Mostrar panel de próximas tareas
    - CU-27: Mostrar página de perfil
    - CU-28: Mostrar listado completo de metas personales
    - CU-29: Mostrar listado completo de metas colaborativas

Código	CU-24
Nombre	Mostrar calendario
Actor	Usuario
Descripción	Permite al usuario visualizar un calendario con las fechas límite de sus metas y, en su caso, tareas relevantes asociadas.
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	Ninguna, el usuario solo consulta información.
Flujo Principal	1) El usuario inicia sesión. 2) El usuario visualiza la página principal, donde se encuentra el calendario para poder organizarse mejor las fechas límite de las metas.

Tabla 38: CU-24: Mostrar calendario.

Código	CU-25
Nombre	Mostrar panel de próximas metas
Actor	Usuario
Descripción	Permite al usuario visualizar un panel con las metas personales cuya fecha límite está próxima, para poder priorizarlas.
Precondiciones	El usuario debe estar autenticado y tener metas con fecha límite definida.
Postcondiciones	Ninguna; el usuario solo consulta información.
Flujo Principal	1) El usuario accede al panel principal. 2) El sistema identifica las metas con la fecha límite más próxima. 3) El sistema ordena estas metas, por fecha más cercana. 4) El sistema muestra el panel con las próximas metas.

Tabla 39: CU-25: Mostrar panel de próximas metas.

Código	CU-26
Nombre	Mostrar panel de próximas tareas
Actor	Usuario
Descripción	Permite al usuario visualizar un panel con las tareas personales cuya fecha límite está próxima, para poder priorizarlas.
Precondiciones	El usuario debe estar autenticado y tener tareas pendientes asociadas a metas.
Postcondiciones	Ninguna, el usuario solo consulta información.
Flujo Principal	1) El usuario accede al panel principal. 2) El sistema identifica las tareas con la fecha límite más próxima. 3) El sistema ordena estas tareas, por fecha más cercana. 4) El sistema muestra el panel con las próximas tareas.

Tabla 40: CU-26: Mostrar panel de próximas tareas.

Código	CU-27
Nombre	Mostrar página de perfil
Actor	Usuario
Descripción	Permite al usuario acceder a su página de perfil para consultar su información básica y, en su caso, algunos datos resumen de actividad.
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	Ninguna, el usuario solo consulta información
Flujo Principal	1) El usuario selecciona la opción de “Perfil” en la cabecera de la página principal. 2) El sistema reúne la información básica del usuario (nombre, correo y fecha de creación del usuario) y un

	listado con todas las metas completadas (tanto personales como colaborativas). 3) El sistema muestra la página de perfil con dicha información.

Tabla 41: CU-27: Mostrar página de perfil.

Código	CU-28
Nombre	Mostrar listado completo de metas personales
Actor	Usuario
Descripción	Permite al usuario consultar un listado completo de todas sus metas personales, más allá del resumen mostrado en el panel principal.
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	Ninguna, el usuario solo consulta información.
Flujo Principal	1) El usuario selecciona la opción de ver todas sus metas personales. 2) El sistema reúne todas las metas personales asociadas al usuario. 3) El sistema muestra el listado completo, incluyendo el título y el porcentaje de progreso de cada meta personal.

Tabla 42: CU-28: Mostrar listado completo de metas personales.

Código	CU-29
Nombre	Mostrar listado completo de metas colaborativas
Actor	Usuario
Descripción	Permite al usuario consultar un listado completo de todas las metas colaborativas en las que participa como propietario o miembro.

Precondiciones	El usuario debe estar autenticado y participar en al menos una meta colaborativa.
Postcondiciones	Ninguna, el usuario solo consulta información.
Flujo Principal	1) El usuario selecciona la opción de ver todas sus metas colaborativas. 2) El sistema recupera todas las metas colaborativas en las que el usuario es propietario o participante. 3) El sistema muestra el listado completo, incluyendo el título y el porcentaje de progreso de cada meta colaborativa.

Tabla 43: CU-29: Mostrar listado completo de metas colaborativas.

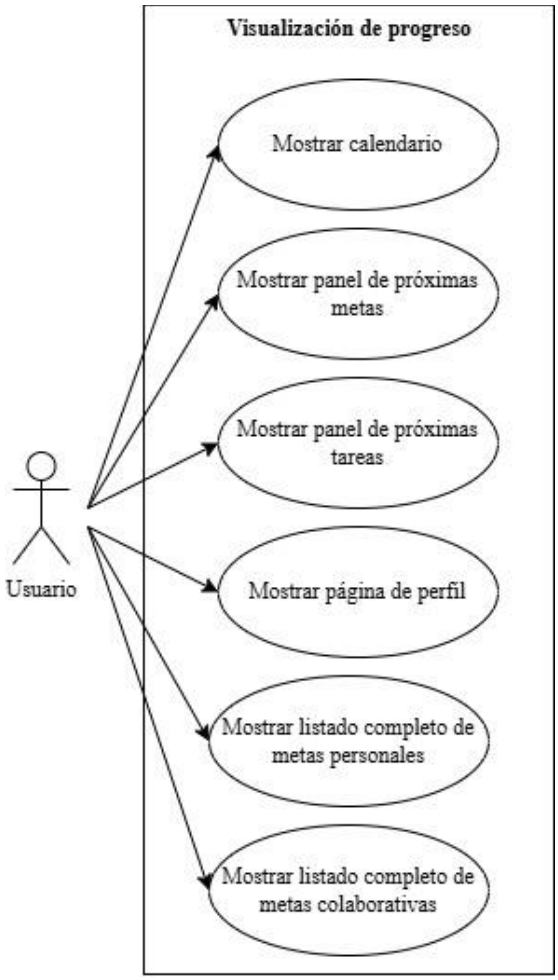

Figura 20: Diagrama de casos de uso: Visualización de progreso.

3.4. Diseño de la base de datos

La base de datos constituye el núcleo de la información del sistema, ya que en ella se almacena toda la información relacionada con los usuarios, sus metas, las tareas asociadas y las aportaciones de progreso realizadas. En este apartado se describe el diseño de la base de datos siguiendo dos niveles: el modelo conceptual, representado mediante un diagrama Entidad-Relación, y el modelo lógico, que concreta dicho diseño en un esquema relacional implementado en MySQL.

El objetivo principal del diseño es garantizar la integridad de los datos, evitar redundancias innecesarias y facilitar las operaciones habituales del sistema, como obtener el progreso global de una meta, conocer qué tareas están pendientes o identificar la aportación de cada integrante en una meta colaborativa.

3.4.1. Modelo conceptual

En el modelo conceptual se han identificado las siguientes **entidades** principales:

- Usuario

Representa a cada persona que utiliza la aplicación. Un usuario puede crear metas personales, ser propietario de metas colaborativas e intervenir como participante en metas creadas por otros usuarios.

A nivel conceptual, un usuario se caracteriza por atributos como identificador, nombre y correo electrónico.

- Meta

Representa un objetivo que el usuario quiere alcanzar. Una meta puede ser:

- Personal, cuando solo interviene su creador.
- Colaborativa, cuando tiene varios participantes.

Además, se distingue entre metas de tipo numérico (orientadas a alcanzar una cantidad objetivo) y metas basadas en tareas (orientadas a completar un conjunto de tareas).

Cada meta incluye atributos como identificador, título, descripción, fecha límite, tipo de meta y, en el caso de metas numéricas, la cantidad objetivo e inicial, y la unidad de medida.

- Tarea

Representa una unidad de trabajo dentro de una meta de tipo tareas. Cada tarea está asociada a una única meta (personal o colaborativa) y puede estar marcada como pendiente o completada.

En el caso de metas colaborativas, una tarea puede estar asignada a uno de los participantes, lo que facilita el reparto de responsabilidades dentro del equipo.

Cada tarea incluye atributos como identificador, título, descripción, fecha límite y duración aproximada, para el posterior cálculo del progreso.

- Participación

Esta entidad intermedia modela la relación entre Usuario y Meta en el caso de metas colaborativas. Permite asociar varios usuarios a una misma meta colaborativa, además de su propietario, y sirve de base para registrar qué usuarios pueden aportar progreso o tener tareas asignadas dentro de dicha meta.

- Aportación

Recoge los registros de progreso en metas de tipo numérico. Cada aportación indica cuánto ha avanzado el usuario en una meta.

Además de la cantidad aportada, se registra al usuario, lo que permite calcular el porcentaje de progreso total de la meta y, en metas colaborativas, la contribución de cada integrante.

Sobre estas entidades se han definido las **relaciones** PROPIETARIO, PARTICIPA, COMPUESTA y RESPONSABLE, que permiten modelar la propiedad de las metas, la participación en metas colaborativas, la descomposición de metas en tareas y la asignación de responsables a dichas tareas:

- Relación PROPIETARIO (Usuario - Meta)

La relación PROPIETARIO indica quién es el creador y dueño de cada meta. Permite distinguir de forma clara qué usuario tiene control principal sobre una meta (por ejemplo, quién puede eliminarla o gestionar a sus participantes).

Un usuario puede no tener metas (usuario recién registrado) o ser propietario de muchas metas (0, N). Por otro lado, toda meta debe tener exactamente un propietario; no existen metas sin usuario que las haya creado (1, 1).

- Relación ternaria PARTICIPA (Usuario - Meta - Aportación)

En las metas colaborativas, existen usuarios que no son propietarios de las metas, por lo que es necesario crear otra relación entre Usuario y Meta. Además, no basta con saber qué usuarios participan en una meta, ya que también es necesario registrar cuánto ha aportado cada uno al objetivo numérico. Para modelar este comportamiento se ha definido la relación ternaria PARTICIPA, que conecta las entidades Usuario, Meta y Aportación.

La entidad Aportación actúa como el "registro" de esa participación, almacenando atributos propios como la *cantidad* y la *fecha_aportación*. Por ello, PARTICIPA se representa como una relación ternaria entre el usuario que aporta, la meta sobre la que se aporta, y la aportación concreta realizada.

Para un Usuario y una Meta dada, pueden existir 0 o N Aportaciones.

Es decir, un usuario puede no haber registrado todavía ningún avance en una meta concreta, o puede haber registrado muchas aportaciones a lo largo del tiempo (por ejemplo, varias sesiones de estudio, varios pagos parciales, etc.).

Cada Aportación está vinculada exactamente a un Usuario y a una Meta.

No tiene sentido una aportación que no esté asociada a un usuario concreto ni a una meta concreta. Por eso, desde el punto de vista de Aportación, la cardinalidad hacia Usuario y Meta es (1,1).

Por lo tanto, la relación ternaria permite modelar de forma precisa y flexible el hecho de que varios usuarios contribuyen cuantitativamente al progreso de una meta colaborativa, manteniendo un historial detallado y posibilitando el cálculo posterior de métricas como el avance global y el porcentaje de contribución de cada integrante.

- Relación COMPUESTA (Meta - Tarea)

La relación COMPUESTA enlaza las entidades Meta y Tarea y expresa que ciertas metas (aquellas cuyo *tipo_meta* es "tareas") se descomponen en un conjunto de tareas más pequeñas. Cada tarea representa un paso concreto que contribuye al cumplimiento de la meta.

Una meta puede no tener tareas (si es una meta numérica) o puede estar formada por muchas tareas. Por eso el máximo en el lado de Tarea es N y el mínimo es 0.

Por otro lado, Toda tarea está asociada obligatoriamente a una sola meta. No existen tareas "sueltas" fuera del contexto de una meta. Esto se refleja como participación total de Tarea en la relación: cada ocurrencia de Tarea debe participar en COMPUESTA.

Desde el punto de vista conceptual, Tarea se comporta como una entidad débil respecto de Meta.

Una entidad débil es aquella cuya existencia y significado dependen de otra entidad (llamada entidad fuerte) y que no puede identificarse de forma única sin la referencia a esa entidad.

En este modelo, una Tarea solo tiene sentido dentro de una Meta. No se concibe una tarea que no pertenezca a ninguna meta. Es decir, siempre respondemos a la pregunta "¿tarea de qué meta?".

Conceptualmente, una tarea podría identificarse por una clave compuesta del estilo (Meta, *número_de_tarea*): el "número de tarea" sería una clave parcial que solo es única dentro de la meta a la que pertenece. Esa combinación (*id_meta* y *número_tarea*) formaría la clave primaria completa típica de una entidad débil.

En el diagrama Entidad-Relación se ha optado por incluir un atributo *id_tarea* para facilitar la implementación posterior, pero la dependencia sigue siendo la misma: Tarea no puede existir sin una Meta asociada, y su participación en la relación COMPUESTA es (1, 1).

- Relación RESPONSABLE (Usuario - Tarea)

La relación RESPONSABLE vincula cada tarea con el usuario que tiene la responsabilidad principal de ejecutarla. Es especialmente útil en metas colaborativas, donde es necesario saber quién debe realizar cada tarea.

Un usuario puede no tener ninguna tarea asignada o ser responsable de muchas tareas (0, N). Por otro lado, una tarea puede no tener responsable (por ejemplo, en una meta personal) o tener exactamente un usuario responsable (0, 1).

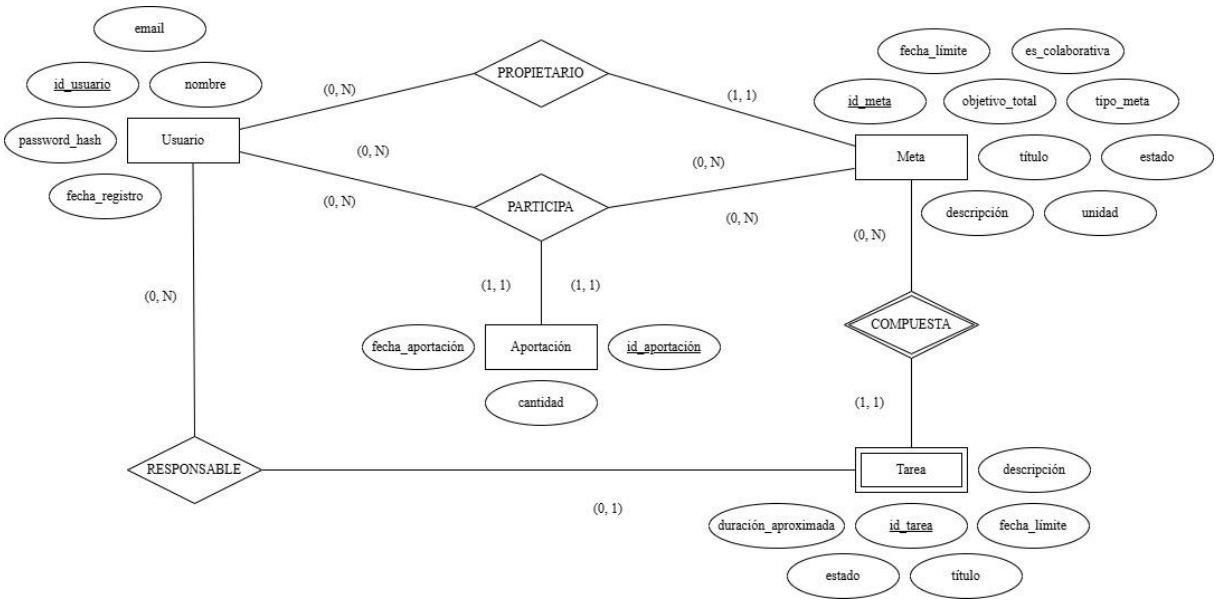

Figura 21: Diseño inicial del Diagrama Entidad-Relación.

Este modelo conceptual asegura que el sistema puede representar tanto metas individuales como colaborativas, contemplando las dos variantes de metas (numéricas y de tareas) y permitiendo registrar de forma estructurada el progreso y la participación de los usuarios.

3.4.2. Modelo lógico

A partir del modelo conceptual descrito en el apartado anterior se ha definido el esquema relacional que se ha implementado en MySQL. En esta fase se concretan las entidades y relaciones del diagrama E/R en tablas, claves primarias y claves foráneas, aplicando los principios de normalización para evitar redundancias y facilitar el mantenimiento de los datos.

Aunque en el modelo conceptual se ha utilizado una única entidad Meta y una única entidad Tarea, en la base de datos se ha optado por separar la información en tablas específicas para metas personales y metas colaborativas, así como para las tareas asociadas a cada tipo de meta. Esta decisión simplifica la lógica de consulta e implementación en el código PHP.

- Tabla *users*

La tabla *users* almacena la información de los usuarios registrados en la aplicación. Es la materialización directa de la entidad Usuario del modelo conceptual.

Función principal: representar a cada persona que puede iniciar sesión y gestionar metas en el sistema.

Relaciones:

- Un usuario puede ser propietario de varias metas personales (*goals*) y colaborativas (*collab_goals*).
- Un usuario puede aparecer como participante en metas colaborativas (*collab_goal_participants*).
- Un usuario puede tener tareas colaborativas asignadas (*collab_tasks*).
- Un usuario puede registrar aportaciones de progreso en metas colaborativas (*collab_goal_updates*).

- Tabla *goals*

La tabla *goals* almacena las metas personales de cada usuario. Corresponde a una especialización de la entidad Meta para el caso de metas no colaborativas.

Función principal: representar metas individuales, tanto numéricas como basadas en tareas, asociadas a un único usuario.

Relaciones:

- Se relaciona con *users* a través de *user_id*.
- Si la meta es de tipo tareas, se relaciona con la tabla *task* (tareas personales) mediante una clave foránea en esa tabla.

- Tabla *collab_goals*

La tabla *collab_goals* almacena las metas colaborativas, es decir, aquellas en las que participan varios usuarios. Es la otra especialización de la entidad Meta del modelo conceptual.

Relaciones:

- Se relaciona con *users* (propietario) mediante la clave foránea al usuario creador.
- Se relaciona con *collab_goal_participants*, que representa la relación PARTICIPA (qué usuarios participan en la meta).
- Se relaciona con *collab_tasks*, que almacena las tareas asociadas a esta meta cuando es de tipo tareas.
- Se relaciona con *collab_goal_updates*, que almacena las aportaciones numéricas a la meta cuando es de tipo numérico.

- Tabla *task*

La tabla *task* almacena las tareas asociadas a metas personales de tipo tareas. Corresponde a la entidad Tarea del modelo conceptual, restringida al caso de metas personales.

Relaciones:

- Cada registro de *task* está asociado a exactamente una meta personal (*goals*), reforzando el papel de Tarea como entidad dependiente de Meta.
- El porcentaje de progreso de la meta personal se calcula a partir del número de tareas completadas en esta tabla.

- Tabla *collab_tasks*

La tabla *collab_tasks* almacena las tareas asociadas a metas colaborativas de tipo tareas. Es la versión colaborativa de la entidad Tarea.

Relaciones:

- Cada tarea colaborativa pertenece a una única meta colaborativa.
- Cada tarea está asignada a un usuario participante de la meta, permitiendo el seguimiento de responsabilidades dentro del equipo.

- Tabla collab_goal_participants

La tabla *collab_goal_participants* representa la participación de los usuarios en metas colaborativas. Es la materialización de la relación PARTICIPA del modelo conceptual. Registra qué usuarios forman parte de cada meta colaborativa, además del propietario.

Relaciones:

- Un usuario puede participar en varias metas colaborativas,
- Una meta colaborativa puede tener varios participantes.

- Tabla *collab_goal_updates*

La tabla *collab_goal_updates* almacena las aportaciones de progreso realizadas por los usuarios en metas colaborativas de tipo numérico. Es la implementación de la entidad Aportación ligada a la relación ternaria PARTICIPA.

La función principal es registrar cada avance cuantitativo de un usuario sobre una meta colaborativa, manteniendo un histórico de las contribuciones.

Relaciones:

- Para cada par (usuario, meta colaborativa) puede existir un número arbitrario de registros en *collab_goal_updates*, que representan distintas aportaciones en momentos diferentes.
- El progreso global de una meta numérica se calcula sumando todas las aportaciones asociadas.
- El porcentaje de contribución de cada participante se obtiene comparando la suma de sus aportaciones con la suma total de la meta.

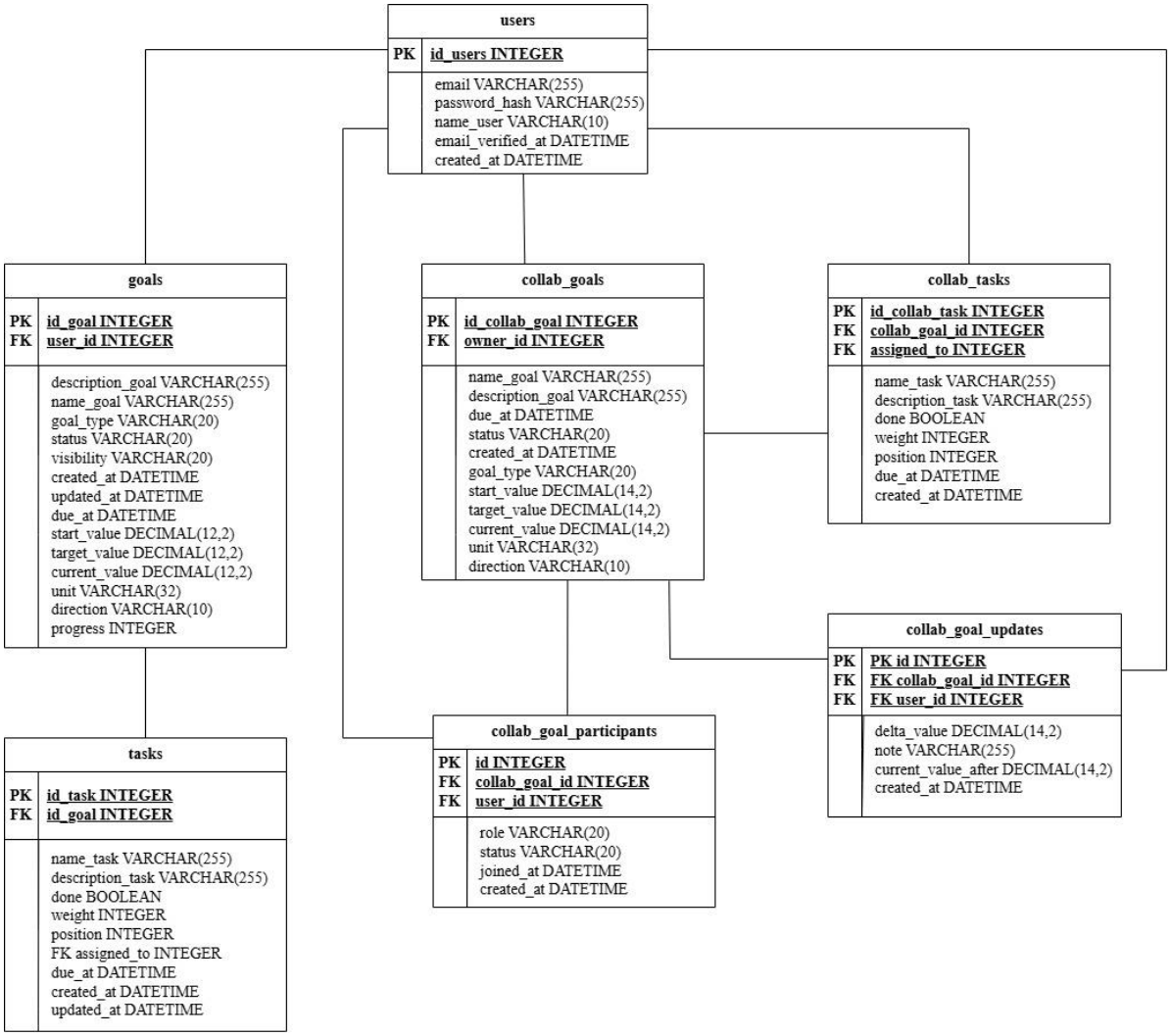

Figura 22: Diagrama del modelo lógico de la base de datos.

El modelo lógico mantiene la estructura del modelo conceptual, aunque introduce especializaciones (metas y tareas colaborativas) y tablas intermedias para adaptarse a las particularidades del sistema gestor de bases de datos y a las necesidades de implementación de la aplicación.

3.4.3. Preparación del entorno de servidor local con XAMPP

Antes de crear la base de datos, ha sido necesario preparar un entorno de servidor local que permitiese ejecutar tanto el motor de base de datos como el servidor web donde se aloja la aplicación.

Para ello se ha utilizado XAMPP, un paquete que integra en una única instalación los componentes fundamentales para el desarrollo web: servidor HTTP Apache, intérprete de PHP y servidor de bases de datos MySQL, además de la herramienta de administración phpMyAdmin.

La instalación de XAMPP se ha realizado utilizando la configuración por defecto, suficiente para un entorno de desarrollo. Una vez instalado, se han arrancado los servicios de Apache y MySQL desde el panel de control de XAMPP, lo que permite que la aplicación PHP sea accesible en http://localhost y que el gestor de bases de datos esté disponible en el puerto configurado por defecto. Para la administración gráfica de la base de datos se ha utilizado phpMyAdmin, accesible a través de http://localhost/phpmyadmin.

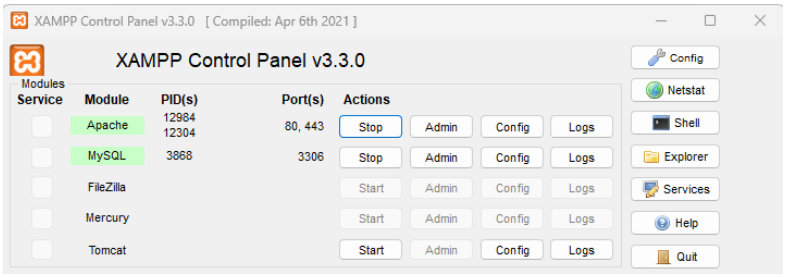

Figura 23: XAMPP Control Panel.

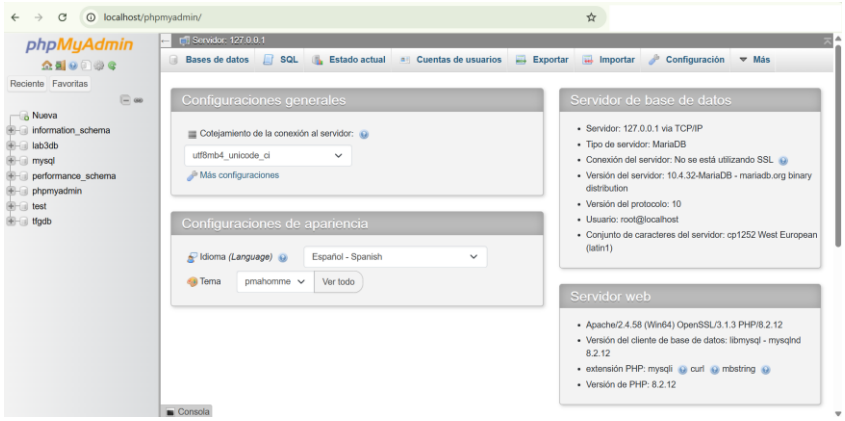

Figura 24: Página de administración gráfica de la base de datos (phpMyAdmin).

Una vez preparado el entorno de servidor local y verificado el correcto funcionamiento de Apache, MySQL y phpMyAdmin, se ha procedido a ejecutar el script de creación de la base de datos, que define tanto el esquema *tfgdb* como todas las tablas y relaciones utilizadas por la aplicación.

3.4.4. Creación de la base de datos en MySQL

Una vez definido el modelo lógico y preparado el entorno de ejecución con XAMPP, el siguiente paso ha sido materializar el esquema de datos en el servidor MySQL. Para ello se ha elaborado un script SQL que automatiza todo el proceso: creación del usuario de base de datos específico para el proyecto, generación del esquema tfgdb y definición de todas las tablas, claves primarias, claves foráneas e índices necesarios para el correcto funcionamiento de la aplicación.

El uso de un script de creación presenta varias ventajas frente a la creación manual de tablas mediante la interfaz gráfica de phpMyAdmin: garantiza que el esquema pueda reproducirse de forma idéntica en diferentes máquinas, facilita la trazabilidad de cambios sobre la base de datos y permite versionar la estructura junto con el resto del código del proyecto. Además, cualquier modificación del diseño (por ejemplo, añadir un nuevo campo o relación) puede incorporarse al script y volver a desplegarse de forma controlada.

A continuación, se mostrará los pasos seguidos para crear el usuario de MySQL, el esquema *tfgdb* y cada una de las tablas que componen la base de datos.

```
DROP DATABASE IF EXISTS tfgdb;

DROP USER IF EXISTS 'daniel'@'localhost';
CREATE USER 'daniel'@'localhost' IDENTIFIED BY 'password';
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'daniel'@'localhost';
GRANT ALL PRIVILEGES ON *.* TO 'daniel'@'localhost' REQUIRE NONE WITH GRANT OPTION
  MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0
  MAX_UPDATES_PER_HOUR 0 MAX_USER_CONNECTIONS 0;
FLUSH PRIVILEGES;
```

Figura 25: Configuración de un usuario específico de MySQL.

En primer lugar, el script elimina la base de datos *tfgdb* y el usuario *daniel* si ya existían en el servidor. Esto permite empezar siempre de un estado limpio y

reproducible. Cada vez que se ejecuta el script, el entorno de desarrollo queda exactamente igual, sin restos de pruebas anteriores ni esquemas antiguos. A continuación, se vuelve a crear el usuario *daniel* con su contraseña y se le revocan todos los privilegios que pudiera tener de configuraciones previas. Después se le asignan explícitamente los permisos que va a utilizar la aplicación. De este modo se controla de forma clara qué puede hacer este usuario y se evita depender de permisos heredados o configuraciones manuales.

Configurar un usuario específico de MySQL y asignarle privilegios forma parte de la preparación correcta del entorno de base de datos.

En lugar de utilizar el usuario administrador por defecto (por ejemplo, root en XAMPP), se crea un usuario propio del proyecto, en este caso *daniel*, que será el que utilice la aplicación para conectarse a la base de datos. Esto permite separar claramente las tareas de administración del servidor de bases de datos (que seguirían haciéndose con el usuario administrador) del acceso diario que realiza la aplicación.

De este modo, si en algún momento es necesario cambiar la contraseña, restringir permisos o revocar el acceso de la aplicación, basta con actuar sobre ese usuario concreto sin afectar al resto del servidor.

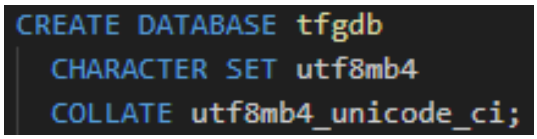

Figura 26: Creación de la base de datos tfgdb.

Una vez preparado el usuario, se crea la base de datos *tfgdb* con un conjunto de caracteres *utf8mb4* y la colación *utf8mb4_unicode_ci*. La elección de *utf8mb4* es importante desde el punto de vista técnico porque es la versión completa de UTF-8 en MySQL, que permite almacenar correctamente cualquier carácter Unicode (acentos, símbolos especiales, emojis, etc.), algo muy útil en una aplicación donde los usuarios pueden escribir textos libres en nombres y descripciones de metas y tareas. La colación *utf8mb4_unicode_ci* define cómo se comparan y ordenan las cadenas, utilizando reglas Unicode y sin distinguir entre mayúsculas y minúsculas, lo que mejora la coherencia al hacer búsquedas y ordenaciones de textos en distintos idiomas.

```sql
CREATE TABLE tfgdb.users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  name_user VARCHAR(10) NOT NULL,
  email_verified_at DATETIME NULL,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Figura 27: Creación tabla users.

Se define la tabla *users*, que almacena la información básica de cada usuario de la aplicación. Se crea un identificador numérico auto incremental como clave primaria y se exige que el correo electrónico sea único y no nulo, ya que se utilizará para el inicio de sesión. La contraseña se guarda en forma de hash y se añaden campos auxiliares, como el nombre del usuario, la fecha en la que se verificó el correo (para futuras implementaciones) y la fecha de creación del registro, que se rellena automáticamente en el momento.

```sql
CREATE TABLE tfgdb.goals (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  description_goal VARCHAR(255) NOT NULL,
  name_goal VARCHAR(255) NOT NULL,

  goal_type ENUM('NUMERIC','TASK_LIST') NOT NULL DEFAULT 'TASK_LIST',

  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,
  due_at DATETIME NOT NULL,

  -- Campos para metas numéricas
  start_value DECIMAL(12,2) NULL,
  target_value DECIMAL(12,2) NULL,
  current_value DECIMAL(12,2) NULL,
  unit VARCHAR(32) NULL,
  direction ENUM('UP','DOWN') NULL,
  progress INT NOT NULL DEFAULT 0,

  CONSTRAINT fk_goals_user FOREIGN KEY (user_id) REFERENCES tfgdb.users(id) ON DELETE CASCADE
);
```

Figura 28: Creación tabla goals.

Esta parte del código corresponde a la tabla *goals*, que representa las metas personales.

Además del identificador propio, se incluye una clave foránea *user_id* que enlaza cada meta con el usuario propietario. Se recogen el nombre y la descripción de la

meta, y mediante el campo *goal_type*, se diferencia entre metas numéricas y metas basadas en listas de tareas. También se registran las fechas de creación, actualización y vencimiento. Para las metas numéricas se añaden campos específicos como el valor inicial, el objetivo, el valor actual, la unidad de medida y la dirección del progreso, junto con un campo de progreso entero para compatibilidad. Finalmente, la clave foránea hacia *users* se define con borrado en cascada para que, si se elimina un usuario, se eliminen automáticamente sus metas personales.

```
CREATE TABLE tfgdb.tasks (
  id INT AUTO_INCREMENT PRIMARY KEY,
  goal_id INT NOT NULL,
  name_task VARCHAR(255) NOT NULL,
  description_task VARCHAR(255) NOT NULL,
  done BOOLEAN NOT NULL DEFAULT FALSE,

  weight TINYINT UNSIGNED NOT NULL DEFAULT 1,
  position INT UNSIGNED NOT NULL DEFAULT 1,
  assigned_to INT NULL,
  due_at DATETIME NOT NULL,

  created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,

  CONSTRAINT fk_task_goal FOREIGN KEY (goal_id) REFERENCES tfgdb.goals(id) ON DELETE CASCADE,
  CONSTRAINT fk_task_assignee FOREIGN KEY (assigned_to) REFERENCES tfgdb.users(id) ON DELETE SET NULL
);
```

Figura 29: Creación tabla tasks.

La tabla *tasks*, almacena las tareas asociadas a metas personales de tipo lista de tareas. Cada tarea tiene su propio identificador y una clave foránea *goal_id* que la vincula con la meta a la que pertenece, de nuevo con borrado en cascada para que las tareas desaparezcan si se elimina la meta. Se guardan el nombre y la descripción de la tarea, un indicador booleano que señala si está completada, y campos adicionales como el peso y la posición, utilizados para calcular el progreso y ordenar las tareas en la interfaz. El campo *assigned_to* permite asignar opcionalmente la tarea a un usuario concreto, y la fecha límite *due_at* sirve para priorizar el trabajo. Por último, se registran las fechas de creación y actualización de la tarea, y la clave foránea *assigned_to* se define con ON DELETE SET NULL para mantener la tarea, aunque se elimine el usuario asignado, dejando simplemente el campo sin valor.

```sql
CREATE TABLE tfgdb.collab_goals (
  id INT AUTO_INCREMENT PRIMARY KEY,
  owner_id INT NOT NULL,
  name_goal VARCHAR(255) NOT NULL,
  description_goal VARCHAR(255) NOT NULL,
  due_at DATETIME NULL,
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,

  goal_type ENUM('TASK_LIST','NUMERIC') NOT NULL DEFAULT 'TASK_LIST' AFTER description_goal,
  start_value   DECIMAL(14,2) NULL AFTER goal_type,
  target_value  DECIMAL(14,2) NULL AFTER start_value,
  current_value DECIMAL(14,2) NULL AFTER target_value,
  unit          VARCHAR(32)   NULL AFTER current_value,
  direction     ENUM('UP','DOWN') NULL AFTER unit;

  INDEX (owner_id),
  FOREIGN KEY (owner_id) REFERENCES users(id) ON DELETE CASCADE
);
```

Figura 30: Creación tabla collab_goals.

La tabla *collab_goals*, que almacena las metas colaborativas. Igual que en las metas personales, se crea un identificador auto incremental y se guarda el usuario propietario mediante el campo *owner_id*, que está vinculado a la tabla de usuarios con una clave foránea y borrado en cascada, de modo que si se elimina el propietario se eliminan también sus metas colaborativas. Se registran el nombre, la descripción, la fecha de vencimiento y la fecha de creación. Además, se incluyen los campos *goal_type, start_value, target_value, current_value, unit* y *direction* para soportar metas colaborativas tanto de tipo lista de tareas como numéricas, con la misma lógica de progreso que en las metas personales. El índice sobre *owner_id* acelera las consultas que obtienen todas las metas colaborativas de un usuario.

```sql
CREATE TABLE tfgdb.collab_goal_participants (
  id INT AUTO_INCREMENT PRIMARY KEY,
  collab_goal_id INT NOT NULL,
  user_id INT NOT NULL,

  role ENUM('OWNER','EDITOR','VIEWER') NOT NULL DEFAULT 'VIEWER',
  status ENUM('PENDING','ACCEPTED','DECLINED','REMOVED') NOT NULL DEFAULT 'PENDING',

  joined_at DATETIME NULL,
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  UNIQUE KEY uq_collab_goal_user (collab_goal_id, user_id),

  INDEX idx_goal (collab_goal_id),
  INDEX idx_user (user_id),
  FOREIGN KEY (collab_goal_id) REFERENCES collab_goals(id) ON DELETE CASCADE,
  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

Figura 31: Creación tabla collab_goal_participants.

La tabla *collab_goal_participants*, que materializa la participación de los usuarios en las metas colaborativas. Cada fila enlaza una meta (*collab_goal_id*) con un usuario (*user_id*) y permite asignarle un role (propietario, editor o solo lectura) y un status que indica si la invitación está pendiente, aceptada, rechazada o el usuario ha sido eliminado (Este atributo quedará para futuras implementaciones de la aplicación). Los índices por *collab_goal_id* y *user_id* mejoran el rendimiento al listar participantes de una meta o metas de un usuario, y la restricción de unicidad sobre el par (meta, usuario) evita que el mismo usuario se registre dos veces en la misma meta. Las claves foráneas hacia *collab_goals* y *users* usan borrado en cascada para limpiar automáticamente las participaciones cuando se elimina una meta o un usuario.

```
CREATE TABLE tfgdb.collab_tasks (
  id INT AUTO_INCREMENT PRIMARY KEY,
  collab_goal_id INT NOT NULL,
  name_task VARCHAR(255) NOT NULL,
  description_task VARCHAR(255) NOT NULL,
  done BOOLEAN NOT NULL DEFAULT FALSE,
  weight INT NOT NULL DEFAULT 1,
  position INT NOT NULL DEFAULT 1,
  due_at DATETIME NULL,
  assigned_to INT NULL,
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  INDEX (collab_goal_id),
  INDEX (assigned_to),
  FOREIGN KEY (collab_goal_id) REFERENCES collab_goals(id) ON DELETE CASCADE,
  FOREIGN KEY (assigned_to) REFERENCES users(id) ON DELETE SET NULL
);
```

Figura 32: Creación tabla collab_tasks.

La tabla *collab_tasks*, que almacena las tareas asociadas a una meta colaborativa de tipo lista de tareas. Cada tarea se vincula a una meta mediante *collab_goal_id* y puede asignarse opcionalmente a un usuario con el campo *assigned_to*. Se guardan el nombre y la descripción de la tarea, un indicador done que marca si está completada, y los campos *weight* y *position* para gestionar el peso en el cálculo del progreso y el orden de visualización. La fecha límite *due_at* permite priorizar tareas, y la fecha de creación se rellena automáticamente. Las claves foráneas aseguran que no haya tareas huérfanas: si se elimina una meta colaborativa, sus tareas se eliminan en cascada; si se elimina un usuario asignado, la tarea se mantiene, pero el campo *assigned_to* pasa a ser nulo.

```sql
CREATE TABLE tfgdb.collab_goal_updates (
  id INT AUTO_INCREMENT PRIMARY KEY,
  collab_goal_id INT NOT NULL,
  user_id INT NOT NULL,
  delta_value DECIMAL(14,2) NOT NULL,
  note VARCHAR(255) NULL,
  current_value_after DECIMAL(14,2) NULL,
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_goal_user (collab_goal_id, user_id),
  CONSTRAINT fk_cgu_goal FOREIGN KEY (collab_goal_id) REFERENCES collab_goals(id) ON DELETE CASCADE,
  CONSTRAINT fk_cgu_user FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```

Figura 33: Creación tabla collab_goal_updates.

La tabla *collab_goal_updates*, que registra las aportaciones numéricas de los usuarios a las metas colaborativas de tipo numérico. Cada actualización indica a qué meta corresponde (*collab_goal_id*), qué usuario la ha realizado (*user_id*), el cambio aplicado al valor actual mediante *delta_value*, un posible comentario (*note*) y el valor resultante tras aplicar esa aportación (*current_value_after*). La fecha de creación se guarda automáticamente, y el índice combinado por meta y usuario facilita calcular, por ejemplo, el historial de aportaciones de un usuario concreto dentro de una meta. Las claves foráneas hacia *collab_goals* y *users* con borrado en cascada garantizan la integridad referencial si desaparece una meta o un usuario, también lo hacen sus registros de actualización asociados.

Una vez que tenemos el script sql finalizado y ejecutado en nuestra página de administración gráfica de la base de datos, así es como aparecen las diferentes tablas para su correcta gestión.

Figura 34: Tablas de la base de datos en phpMyAdmin.

Posteriormente, podemos comprobar específicamente todas las tablas en particular, ver las diferentes columnas que hemos creado, sus características y posteriormente (una vez que se vayan introduciendo usuarios y metas en la aplicación) todos los datos que van componiendo la base de datos.

Y gráficamente podremos hacer una correcta y fácil gestión de la base de datos.

Figura 35: Estructura de la tabla users.

3.5. Diseño de la arquitectura de la aplicación

La arquitectura general de la aplicación responde a un modelo web de tipo cliente-servidor. El cliente está formado por el navegador del usuario, que se encarga únicamente de renderizar las páginas HTML/CSS/JavaScript y enviar peticiones HTTP al servidor. En el lado servidor, el paquete XAMPP integra el servidor web Apache, el intérprete de PHP y el gestor de bases de datos MySQL: Apache recibe las peticiones, ejecuta los scripts PHP correspondientes y estos, a su vez, consultan o actualizan la base de datos antes de generar una respuesta en formato HTML que se devuelve al navegador.

Dentro del servidor se ha seguido además una arquitectura en capas, separando la presentación, la lógica de aplicación y el acceso a datos. La estructura intenta respetar estos principios para facilitar el mantenimiento y la evolución del sistema.

3.5.1. Arquitectura en capas dentro del servidor

En el contexto del servidor, la aplicación se organiza en tres niveles principales:

- Capa de presentación

Está formada por las páginas PHP que generan el HTML que verá el usuario, junto con las hojas de estilo CSS y el JavaScript necesario para pequeños comportamientos dinámicos en el navegador. Aquí se encuentran las vistas de inicio de sesión, registro, panel principal, detalle de metas personales y colaborativas, formularios de creación y edición, panel de próximas metas y tareas, etc. El diseño se apoya en una interfaz basada en tarjetas, barras de progreso y una paleta de colores coherente, con el objetivo de ofrecer una experiencia clara y visualmente atractiva que fomente el uso continuado de la herramienta.

- Capa de lógica de aplicación

Corresponde al código PHP que procesa las peticiones recibidas desde el cliente, valida los datos de los formularios, aplica las reglas de negocio y decide qué información debe mostrarse en cada momento. En esta capa se implementan los casos de uso definidos en el análisis: creación, edición y eliminación de metas, gestión de tareas personales y colaborativas, actualización del progreso numérico, invitación y gestión de participantes, cálculo de porcentajes de contribución, etc. El código se organiza en módulos, agrupando en ficheros relacionados las operaciones sobre usuarios, metas personales, metas colaborativas y tareas.

- Capa de acceso a datos

Esta capa está formada por las funciones que se encargan de conectarse a la base de datos MySQL y ejecutar las sentencias SQL necesarias para leer o modificar la información. Desde esta capa se realizan las operaciones CRUD sobre las tablas. La conexión se configura una única vez y se reutiliza en los distintos módulos, de forma que el resto del código no depende de los detalles concretos del servidor de base de datos.

Esta separación en capas permite modificar el aspecto visual de la aplicación sin tocar la lógica de negocio, o ajustar el diseño de la base de datos sin reescribir la interfaz.

3.5.2. Flujo cliente-servidor y gestión de sesiones

Cada interacción del usuario con la aplicación sigue el patrón típico cliente-servidor. Cuando el usuario realiza una acción en el navegador (por ejemplo, enviar un formulario para crear una meta o marcar una tarea como completada), el cliente envía una petición HTTP al servidor Apache. Este servidor web determina qué script PHP debe atender la petición y lo ejecuta.

El script PHP comienza incluyendo los ficheros comunes, inicia o reanuda la sesión y comprueba si el usuario está autenticado cuando la funcionalidad lo requiere. A continuación, procesa los parámetros recibidos, llama a las funciones de acceso a datos necesarias para consultar o actualizar la información en MySQL y construye una respuesta HTML que incorpora el resultado de la operación: una lista de metas, un detalle actualizado, un mensaje de confirmación o de error, etc. Finalmente, Apache devuelve esa respuesta al navegador, que la muestra al usuario.

La sesión se gestiona mediante las capacidades estándar de PHP, almacenando en variables de sesión el identificador del usuario y algunos datos básicos que permiten personalizar la interfaz. Gracias a este mecanismo, el servidor puede saber en cada petición qué usuario está interactuando con la aplicación y garantizar que solo accede a sus metas personales y a las metas colaborativas en las que participa.

La combinación del modelo cliente-servidor con una arquitectura interna en capas proporciona una estructura clara y extensible. La lógica de la aplicación y los datos permanecen centralizados en el servidor, mientras que el cliente actúa como una capa de presentación ligera, lo que facilita futuras mejoras tanto en la interfaz como en la lógica sin necesidad de modificar la infraestructura básica.

3.6. Diseño de la interfaz de usuario

La interfaz de usuario es un elemento muy importante en esta aplicación, ya que su objetivo principal es ayudar al usuario a planificar y seguir sus metas de forma continuada. Por este motivo, el diseño se ha orientado no solo a ser funcional, sino también a resultar agradable visualmente y fácil de usar, de modo que favorezca que el usuario vuelva a la herramienta con frecuencia.

La aplicación adopta un diseño con bloques bien diferenciados de información, evitando la sobrecarga visual. La paleta de colores, la tipografía y el uso de espacios en blanco se han escogido con la intención de transmitir claridad y orden, ayudando al usuario a identificar rápidamente qué metas están avanzando, cuáles están próximas a su fecha límite y qué tareas tiene pendientes.

3.6.1. Principios de diseño y usabilidad

En el diseño de la interfaz se han tenido en cuenta los siguientes principios:

- Simplicidad y claridad

Cada pantalla se centra en una tarea concreta, reduciendo el número de elementos simultáneos en pantalla. Los formularios muestran únicamente los campos necesarios y se han agrupado visualmente para facilitar su comprensión.

- Consistencia visual

Se utilizan componentes con un estilo coherente en todas las pantallas. Esto ayuda al usuario a reconocer patrones y a crear una marca específica del producto, siendo así reconocible.

- Énfasis en el progreso

El diseño pone el foco en el estado de las metas mediante barras de progreso, indicadores visuales de completitud y resaltado de metas próximas a vencer. De este modo, el usuario puede, de un vistazo, identificar qué objetivos requieren atención prioritaria.

- **Interfaz agradable para fomentar el uso continuo**

Se ha cuidado especialmente el aspecto estético, colores suaves, separaciones entre secciones, iconos o elementos gráficos sencillos. La intención es que la aplicación resulte "amigable" y no genere rechazo visual, algo clave cuando se pretende que el usuario la utilice de forma recurrente para organizar su día a día.

3.6.2. Páginas principales

Esta será la distribución de las principales páginas que va a tener la aplicación web, necesarias para el total cumplimiento de todos los objetivos:

- Página de presentación de la aplicación
- Página de inicio de sesión y registro
- Página de portal personal
    - Calendario y paneles de próximas metas y tareas
    - Lista de metas personales en proceso
    - Lista de metas colaborativas en proceso
- Página de perfil del usuario
    - Información del usuario
    - Lista de metas personales finalizadas
    - Lista de metas colaborativas finalizadas
- Página de detalles de meta personal
- Página de detalles de meta colaborativa
- Página de editar metas personales
- Página de crear nueva meta personal
- Página de crear nueva meta colaborativa
- Página de añadir nueva tarea
- Página de Miembros de una meta colaborativa

4. Implementación

En este parte de la memoria se describe cómo se ha materializado en código el diseño presentado en los apartados anteriores. Partiendo del modelo de datos y de la arquitectura cliente-servidor definida, se ha desarrollado una aplicación web basada en PHP, HTML, CSS y MySQL, ejecutada en un entorno local proporcionado por XAMPP.

El objetivo de este capítulo no es detallar cada línea de código, sino ofrecer una visión estructurada de la solución, cómo se organiza el proyecto, cómo se gestionan las conexiones a la base de datos, qué módulos implementan las principales funcionalidades y de qué manera se manejan aspectos transversales como la sesión o la validación de datos.

4.1. Entorno de desarrollo

La implementación se ha llevado a cabo en un entorno de desarrollo local configurado con XAMPP, que integra los componentes necesarios para ejecutar aplicaciones web escritas en PHP y conectadas a MySQL. El servidor Apache se encarga de recibir las peticiones HTTP desde el navegador, el intérprete de PHP ejecuta los scripts del lado servidor y MySQL almacena de forma persistente la información de usuarios, metas, tareas y colaboraciones.

Para editar el código se ha utilizado Visual Studio Code, configurado con extensiones de soporte para PHP, HTML, CSS y SQL, lo que facilita el resaltado de sintaxis, el autocompletado y la navegación entre ficheros. El control de versiones se ha gestionado con Git, utilizando un repositorio en GitHub y la herramienta gráfica GitHub Desktop para registrar los cambios de forma incremental y poder volver a estados anteriores del proyecto si fuera necesario.

La base de datos se ha creado en el servidor MySQL local mediante el script descrito en el apartado 3.4.4, que define tanto el usuario de base de datos específico para la aplicación como el esquema *tfgdb* y todas las tablas necesarias. De este modo, cualquier desarrollador que desee reproducir el entorno puede instalar XAMPP, clonar el repositorio del proyecto, ejecutar el script SQL y configurar el archivo de conexión de PHP con las credenciales correspondientes para disponer de una copia funcional de la aplicación en su máquina.

4.2. Estructura del proyecto

La aplicación se ha desarrollado dentro de la *carpeta tfg-metas-web*, que agrupa todos los recursos necesarios: scripts PHP, hojas de estilo, imágenes, el script de creación de la base de datos y la documentación básica para su despliegue. Esta organización facilita tanto el trabajo durante el desarrollo como la posible migración del proyecto a otro equipo o servidor.

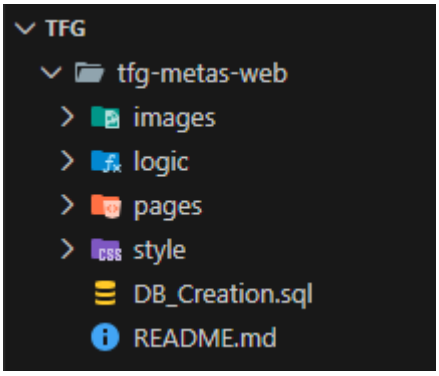

Figura 36: Estructura de directorios del proyecto.

En la raíz del se encuentra la carpeta *tfg-metas-web*, que contiene cuatro subcarpetas principales (*images, logic, pages* y *style*) y dos ficheros destacados: *DB_Creation.sql* y *README.md*. La carpeta *images* almacena los recursos gráficos utilizados en la interfaz (iconos, logotipos imágenes que se muestran en distintas páginas). La carpeta *logic* agrupa los scripts PHP de lógica de aplicación y utilidades, como la conexión a la base de datos, la gestión de la autenticación y la lógica utilizada para la cabecera de la aplicación.

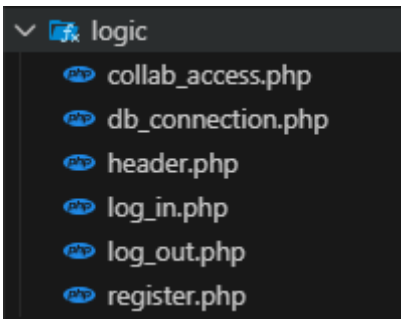

Figura 37: Carpeta logic de la aplicación.

En la carpeta *pages* se sitúan las páginas PHP que actúan como puntos de entrada desde el navegador y generan las distintas vistas de la aplicación (inicio de sesión, panel principal, detalle de metas, gestión de tareas, etc.).

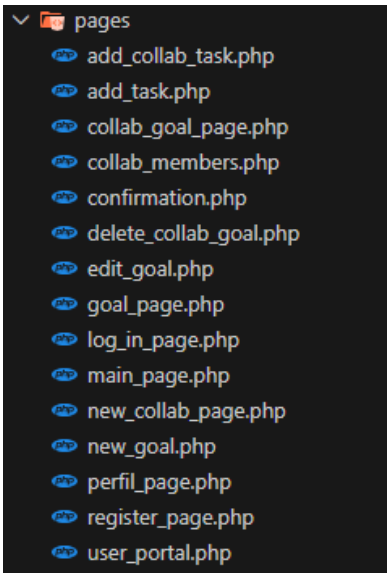

Figura 38: Carpeta pages de la aplicación.

La carpeta style contiene las hojas de estilo CSS asociadas a cada una de estas páginas, lo que refuerza la separación entre la capa de presentación y la lógica del servidor.

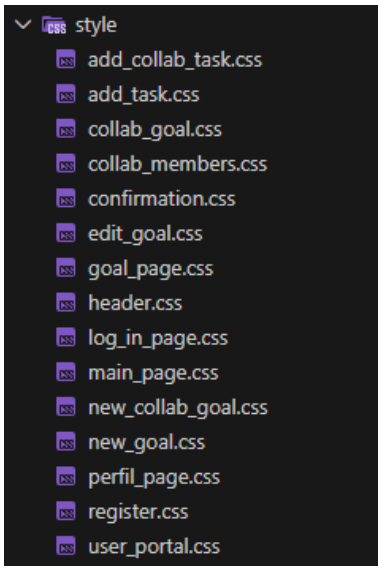

Figura 39: Carpeta style de la aplicación.

El fichero *DB_Creation.sql* incluye el script completo para crear el esquema *tfgdb* en MySQL, definiendo tablas, claves y restricciones tal y como se ha descrito en el capítulo anterior. Por su parte, *README.md* resume los pasos necesarios para poner en marcha el proyecto en un entorno local: requisitos, configuración de XAMPP, ejecución del script de base de datos y ruta de acceso a la aplicación.

Esta estructura de carpetas y ficheros refleja la arquitectura en capas adoptada y proporciona un punto de partida claro para comprender cómo se organizan los distintos componentes de la aplicación.

4.3. Módulos principales del lado servidor

Cuando digo "lado servidor" me refiero a todo el código PHP que se ejecuta en Apache. En el lado servidor existen una serie de módulos PHP que son especialmente relevantes para el funcionamiento de la aplicación. Estos ficheros tienen funcionalidades como la conexión con la base de datos, la autenticación de usuarios o la generación de cabeceras comunes, y son reutilizados desde el resto de las páginas del sistema.

Se describirá los módulos más importantes, mostrando fragmentos del código y explicando el papel que desempeña cada uno dentro de la arquitectura cliente-servidor. El primero de ellos es el código encargado de establecer la conexión con MySQL, del que dependen todas las operaciones posteriores sobre las metas, tareas y usuarios.

4.3.1. Conexión a la base de datos

tfg-metas-web > logic > db_connection.php

```php
<?php
$host = "localhost";
$username = "daniel";
$password = "password";
$dbname = "tfgdb";

// Create connection with MySQLi
$db = new mysqli(hostname: $host, username: $username, password: $password, database: $dbname);

// Check connection
if ($db->connect_error) {
    die("Database error: " . $db->connect_error);
}

```

Figura 40: Conexión con la base de datos.

En la Figura 40, se muestra el fichero responsable de crear la conexión con la base de datos del proyecto. En este script se definen, en primer lugar, los parámetros necesarios para conectarse al servidor MySQL: dirección del host (localhost en el entorno de desarrollo con XAMPP), nombre de usuario, contraseña y nombre de la base de datos (*tfgdb*). Después, se instancia un objeto de la clase *mysqli* utilizando dichos parámetros. Esta instrucción establece efectivamente la conexión con MySQL y devuelve un objeto que se utilizará más adelante para ejecutar consultas SQL. Por último, se realiza una comprobación básica de errores: si la conexión no se ha podido establecer correctamente, se muestra un mensaje y se detiene la ejecución del script.

Este fichero se incluye desde el resto de las páginas PHP que necesitan acceder a la base de datos, de forma que toda la aplicación comparte una única forma centralizada de conectarse al servidor. Esta decisión tiene varias ventajas, ya que reduce la duplicación de código, facilita el mantenimiento (cualquier cambio en las credenciales se hace en un solo lugar) y refuerza la separación entre la lógica de negocio y los detalles de infraestructura.

4.3.2. Gestión de usuarios y autenticación

Estos ficheros se ejecutan como respuesta a los formularios de la interfaz y son los encargados de validar los datos introducidos, interactuar con la base de datos *users* y mantener el estado de autenticación mediante sesiones PHP.

- Registro

tfg-metas-web > logic > register.php

```php
<?php
session_start();
require_once __DIR__ . '/db_connection.php';

if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
  http_response_code(response_code: 405);
  exit('Método no permitido');
}

$email = strtolower(string: trim(string: $_POST['email'] ?? ''));
$pass = $_POST['password'] ?? '';
$username = trim(string: $_POST['name_user'] ?? '');

if ($email === '' || $pass === '' || $username === '') {
  $_SESSION['flash'] = 'Completa todos los campos.';
  header(header: 'Location: ../pages/register_page.php');
  exit;
}
```

Figura 41: Inicio de entorno para procesar registro.

En este primer fragmento se inicializa el entorno necesario para procesar el registro. Se inicia la sesión, se incluye el fichero de conexión a la base de datos y se comprueba que la petición se haya realizado mediante el método HTTP POST. A continuación, se recuperan los valores enviados por el formulario (*email, password* y *name_user*), normalizando el correo (se pasan a minúsculas y se eliminan espacios) y eliminando espacios sobrantes en el nombre de usuario. Por último, se realiza una validación básica, si alguno de los tres campos obligatorios está vacío, se guarda en la sesión un mensaje de error para mostrarlo en la interfaz y se redirige de nuevo a la página de registro, deteniendo la ejecución del script.

```php
$hash = password_hash(password: $pass, algo: PASSWORD_DEFAULT);

$stmt = $db->prepare(query: "INSERT INTO users (email, password_hash, name_user) VALUES (?, ?, ?)");
$stmt->bind_param(types: 'sss', var: &$email, vars: &$hash, $username);

try {
  $stmt->execute();
  $stmt->close();

  $_SESSION['new_user_name'] = $username;

  header(header: 'Location: ../pages/confirmation.php');
  exit;

} catch (Throwable $e) {
  $_SESSION['flash'] = 'Ese email ya está registrado.';
  header(header: 'Location: ../pages/register_page.php');
  exit;
}
```

Figura 42: Procesar alta del usuario

En este segundo fragmento se ejecuta realmente el alta del usuario. Primero se calcula un hash seguro de la contraseña usando password_hash con el algoritmo por defecto recomendado por PHP, de manera que nunca se almacena la contraseña en texto plano. Después se prepara una sentencia INSERT sobre la tabla *users* utilizando *prepared statements*, y se asocian los parámetros (*email, password_hash, name_user*) mediante *bind_param*, lo que protege frente a inyecciones SQL. La ejecución de la sentencia se encapsula en un bloque *try/catch*, es decir, si la inserción tiene éxito, se cierra el statement, se guarda en la sesión el nombre del nuevo usuario para poder mostrar un mensaje de bienvenida y se redirige a la página de confirmación. Si se produce una excepción (por ejemplo, porque el correo ya está registrado y se viola la restricción de unicidad), se captura el error, se almacena un mensaje informativo en la sesión y se redirige de nuevo al formulario de registro, permitiendo que el usuario vuelva a intentarlo.

- Inicio de sesión

tfg-metas-web > logic > log_in.php

```php
<?php
session_start();
require_once __DIR__ . '/db_connection.php';

if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
  http_response_code(response_code: 405);
  exit('Método no permitido');
}

$email = strtolower(string: trim(string: $_POST['email'] ?? ''));
$pass  = $_POST['password'] ?? '';

if ($email === '' || $pass === '') {
  $_SESSION['flash'] = 'Completa todos los campos.';
  header(header: 'Location: ../pages/log_in_page.php'); exit;
}

$stmt = $db->prepare(query: "SELECT id, email, password_hash, name_user FROM users WHERE email = ? LIMIT 1");
$stmt->bind_param(types: 's', var: &$email);
$stmt->execute();
$res  = $stmt->get_result();
$user = $res->fetch_assoc();
$stmt->close();

if (!$user || !password_verify(password: $pass, hash: $user['password_hash'])) {
  $_SESSION['flash'] = 'Credenciales no válidas.';
  header(header: 'Location: ../pages/log_in_page.php'); exit;
}

session_regenerate_id(delete_old_session: true);
$_SESSION['uid']       = (int)$user['id'];
$_SESSION['email']     = $user['email'];
$_SESSION['name_user'] = $user['name_user'];

header(header: 'Location: ../pages/user_portal.php');
exit;
```

Figura 43: Proceso de inicio de sesión.

Este archivo *log_in.php* procesa el formulario de inicio de sesión. Al igual que en el registro, el fichero inicia la sesión, incluye la conexión a la base de datos y verifica que la petición se haya realizado mediante POST. Después normaliza los datos recibidos, es decir, se toma el correo electrónico en minúsculas y se recupera la contraseña tal como la ha introducido el usuario. Si alguno de los dos campos está vacío, se guarda un mensaje de error en la sesión y se redirige al formulario de *login* para que el usuario lo corrija.

El siguiente paso consiste en buscar el usuario en la tabla *users*. Para ello se prepara una consulta SELECT filtrando por el correo electrónico, de nuevo mediante una sentencia preparada para evitar inyecciones SQL. Tras ejecutar la consulta, se obtiene el registro asociado, si existe. El script comprueba entonces dos condiciones de que exista un usuario con ese correo y que la contraseña introducida coincida con la almacenada, utilizando la función *password_verify* sobre el hash guardado en la base de datos. Si alguna de las dos comprobaciones falla, se informa al usuario de que las credenciales no son válidas y se vuelve a mostrar el formulario.

Cuando las credenciales son correctas, se ejecuta *session_regenerate_id(true)* para regenerar el identificador de sesión y mitigar posibles ataques de fijación de sesión. Después, se almacenan en la sesión el identificador del usuario, su correo y su nombre, de modo que el resto de la aplicación pueda saber quién está autenticado en cada petición.

Finalmente, el script redirige al usuario a la página *user_portal.php*, que actúa como punto de entrada a las funcionalidades privadas de la aplicación.

- Cierre de sesión

tfg-metas-web > logic > log_out.php

```php
<?php
session_start();
session_unset();
session_destroy();
header(header: 'Location: ../pages/main_page.php');
exit;

```

Figura 44: Cerrar sesión.

Su funcionamiento es sencillo pero esencial para la seguridad. Comienza iniciando la sesión actual, elimina todas las variables de sesión mediante *session_unset()* y destruye la sesión con *session_destroy()*. Esto garantiza que los datos del usuario autenticado se eliminan del lado del servidor y que un atacante no pueda reutilizar una sesión anterior.

Una vez limpiado el estado de la sesión, el script redirige al usuario a la página principal pública (*main_page.php*) y termina la ejecución. Desde el punto de vista del usuario, el resultado es que deja de estar autenticado y, si intenta acceder a alguna página que requiera login, el sistema le pedirá que vuelva a identificarse.

4.3.3. Cabecera de la aplicación

El fichero logic/header.php concentra la cabecera común que comparten todas las páginas de la aplicación. De este modo, el logo, los botones de acceso y el estilo general del encabezado se definen una única vez y se incluyen allí donde sean necesarios, manteniendo la consistencia visual y evitando duplicar código.

```
tfg-metas-web > logic > header.php
1   <link rel="stylesheet" href="../stylesheet/header.css">
2   <link rel="preconnect" href="https://fonts.googleapis.com">
3   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
4   <link href="https://fonts.googleapis.com/css2?family=Outfit:wght@400;600;700&display=swap" rel="stylesheet">
5
```

Figura 45: Hoja de estilos y fuentes externas.

En la primera parte del archivo se encuentran las etiquetas *link* que cargan los estilos y las fuentes utilizadas en la cabecera. Por un lado, se importa la hoja de estilos *header.css*, donde se define la disposición del logo, la zona de botones y el aspecto general del encabezado. Por otro, se añaden enlaces a Google Fonts para conectar los dominios de las fuentes y cargar la familia tipográfica seleccionada.

```
<header class="header_full">
  <div class="logo">
    <?php
      // Si hay sesión activa, el logo lleva al portal del usuario
      $logoLink = !empty($_SESSION['uid'])
        ? '../pages/user_portal.php'
        : '../pages/main_page.php';
    ?>
    <a href="<?= $logoLink ?>">
      <img src="../images/foto.png" alt="logo" class="logo_img">
    </a>
  </div>

  <div class="login">
    <?php if (!empty($_SESSION['uid'])): ?>
      <a href="../pages/perfil_page.php" class="login-btn btn">Perfil</a>
      <form method="POST" action="">
        <button type="submit" name="logout" class="login-btn btn btn-danger">Salir</button>
      </form>
    <?php else: ?>
      <?php if (!isset($hideLogin) || !$hideLogin): ?>
        <a href="../pages/log_in_page.php" class="login-btn btn btn-success">Iniciar Sesión</a>
      <?php endif; ?>
      <?php if (!isset($hideSignUp) || !$hideSignUp): ?>
        <a href="../pages/register_page.php" class="login-btn btn btn-warning">Registrarse</a>
      <?php endif; ?>
    <?php endif; ?>
  </div>
</header>
```

Figura 46: Estructura HTML y lógica de la cabecera común de la aplicación.

A continuación, aparece el bloque principal <*header*>. Dentro de él se definen dos secciones: una para el logo y otra para los botones de acceso. El logo no es un simple enlace fijo, sino que se genera dinámicamente mediante PHP. Ya que si existe un usuario autenticado (la variable de sesión $_SESSION['uid'] está definida), el logo lleva al portal del usuario. En caso contrario, redirige a la página principal pública. De esta forma, el mismo encabezado se adapta automáticamente al contexto de cada usuario. En la zona derecha se muestran distintos botones según el estado de la sesión. Si el usuario está conectado, se ofrece un enlace a la página de perfil y un botón de "Salir" dentro de un formulario. Si no hay sesión activa, se muestran los enlaces "Iniciar sesión" y "Registrarse", salvo que la propia página los oculte mediante las variables auxiliares *$hideLogin* y *$hideSignUp*, que permiten no repetir botones cuando ya se está en la pantalla correspondiente.

```php
<?php
// logout request
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['logout'])) {
  session_destroy();
  header(header: "Location: ../pages/main_page.php");
  exit();
}
?>
```

Figura 47: Gestión del cierre de sesión desde la cabecera.

Finalmente, al final del archivo se incluye un pequeño bloque PHP que gestiona la petición de cierre de sesión asociada al botón "Salir" de la cabecera. Este código comprueba si la petición HTTP se ha realizado mediante el método POST y si existe el parámetro *logout* en $_POST, lo que indica que el usuario ha pulsado el botón de salida. En ese caso se destruye la sesión actual, se redirige al usuario a la página principal (*main_page.php*) y se detiene la ejecución. Gracias a este mecanismo, cualquier página que incluya header.php hereda automáticamente la misma lógica de cierre de sesión, sin necesidad de duplicar el código en cada fichero.

Por otro lado, el estilo que se ha elaborado para la cabecera busca mantener una apariencia limpia y coherente con el resto de la aplicación. Se ha definido una gama de colores suave, utilizando un fondo claro y acentos en los botones para que destaquen sin resultar agresivos visualmente. La disposición del contenido se ha planteado de forma flexible, de manera que la cabecera se adapte correctamente a

distintos tamaños de pantalla, el logo y los botones se reorganizan o ajustan su tamaño para seguir siendo legibles tanto en resoluciones amplias como en ventanas más pequeñas. Además, los botones incorporan pequeños efectos interactivos, como cambios de color y ligeros desplazamientos al pasar el ratón por encima, que aportan dinamismo y transmiten sensación de respuesta inmediata, haciendo que la interfaz resulte más atractiva y agradable de usar para el usuario.

Aquí se muestran algunas partes del código css del archivo header.css aplicando algunas de las cosas que hemos mencionado antes:

```css
:root {
    --blanco: #ffffff;
    --negro: #000000;
    --rojo: #e43d3d;
    --celeste: #a2dcfd;
    --azul-verdoso: #1EA8C3;
    --azul-oscuro: #1F5CA9;
}
```

Figura 48: Gama de colores de la cabecera (header.css).

```css
.header_full {
    height: 12%;
    max-height: 12%;
    display: flex;
    align-items: center;
    padding: 0.5% 2% 0.5% 0.5%;
    background-color: var(--celeste);
    overflow: hidden;
}
```

Figura 49: Disposición de la cabecera (header.css).

```css
.header_full .login-btn[href="../pages/log_in_page.php"] {
    background-color: var(--azul-verdoso) !important;
    border: 0.18vw solid var(--azul-verdoso) !important;
    border-radius: 5px !important;

}

.header_full .login-btn[href="../pages/log_in_page.php"]:hover {
    background-color: var(--azul-oscuro) !important;
    border: 0.22vw solid var(--azul-oscuro) !important;
}
```

Figura 50: Código de estilo para el botón de inicio se sesión.

El resultado de la cabecera ha sido el siguiente:

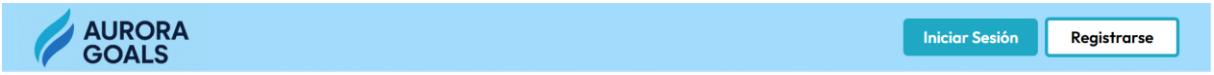

Figura 51: Cabecera de la aplicación en la página principal sin ninguna sesión abierta.

Cabecera tal y como aparece en la página principal cuando no hay ningún usuario autenticado. El logo se sitúa a la izquierda y, a la derecha, se ofrecen los botones "Iniciar sesión" y "Registrarse" con su estilo por defecto.

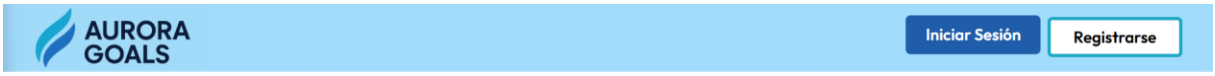

Figura 52: Cabecera de la aplicación manteniendo el botón de inicio.

El efecto *hover* aplicado sobre el botón "Iniciar sesión". Al situar el cursor sobre él, el color de fondo se intensifica y el borde resalta ligeramente, reforzando la sensación de interacción y haciendo más visible la acción disponible.

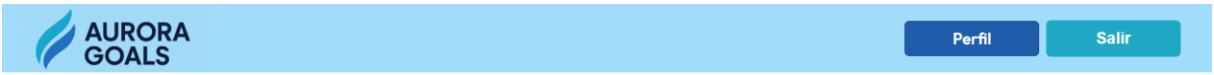

Figura 53: Cabecera de la aplicación en la página principal con la sesión abierta.

Cabecera cuando el usuario ya ha iniciado sesión. Los botones de la derecha cambian a "Perfil" y "Salir", manteniendo el mismo esquema de colores y disposición para conservar la coherencia visual con el resto de la interfaz.

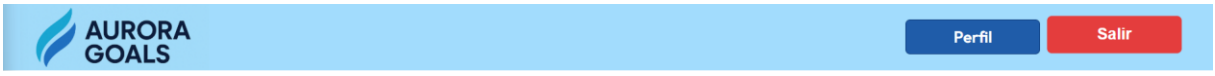

Figura 54: Cabecera de la aplicación manteniendo el botón de salir.

Efecto *hover* sobre el botón "Salir". Al pasar el ratón por encima, el botón adopta un tono rojo más llamativo, indicando de forma clara que se trata de una acción de cierre de sesión y ayudando al usuario a identificarla rápidamente.

4.4. Páginas principales e integración de lógica y estilos

En los apartados anteriores se han descrito los módulos comunes de la carpeta *logic*, encargados de tareas esenciales como la conexión a la base de datos, la autenticación de usuarios o la cabecera compartida. En este apartado se presentan algunas de las páginas principales de la carpeta *pages*.

Cada una de estas páginas sigue un patrón similar:

- Incluye los ficheros comunes necesarios (conexión a la base de datos, cabecera y comprobación de sesión).
- Ejecuta las consultas SQL correspondientes para obtener la información que debe mostrarse (metas, tareas, participantes…).
- Genera la estructura HTML a partir de esos datos, organizándolos en tarjetas y listados.
- Aplica una hoja de estilo específica que define la distribución en pantalla, los colores, las barras de progreso y los efectos visuales.

A continuación, se describen algunas partes de las páginas más representativas, explicando cómo integran la lógica de negocio y como se realiza el diseño visual.

4.4.1. Panel principal del usuario

En este apartado se muestra cómo se integra la lógica PHP con el HTML y el CSS en la página user_portal.php, que actúa como portal del usuario. Las figuras asociadas recogen, por este orden, la parte del código que controla el acceso y obtiene los datos, la consulta SQL que calcula el progreso de las metas, la estructura básica del documento HTML, la generación dinámica del listado de metas y, por último, los estilos principales aplicados a esta sección.

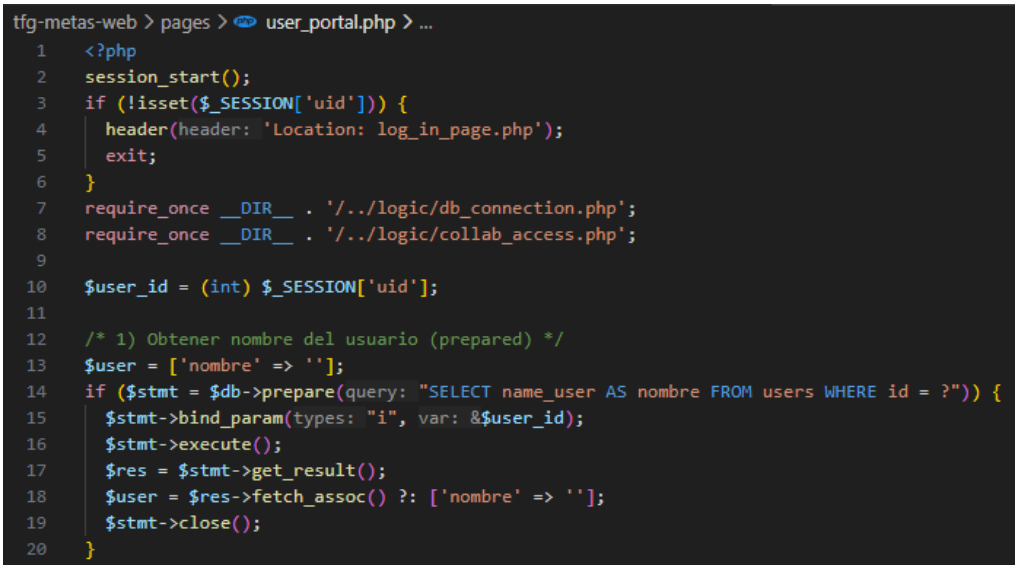

Figura 55: Control de acceso y carga del usuario en user_portal.php.

En la primera figura se ve el inicio del script *user_portal.php*. Nada más comenzar se arranca la sesión y se comprueba si existe el identificador del usuario en $_SESSION['uid']; si no es así, se redirige a la página de inicio de sesión, evitando que un usuario no autenticado pueda acceder al portal. A continuación, se incluyen los módulos *db_connection.php* y *collab_access.php*, necesarios para comunicarse con la base de datos y reutilizar funciones relacionadas con metas colaborativas. Finalmente, se obtiene el identificador del usuario a partir de la sesión y se ejecuta una consulta preparada para recuperar su nombre desde la tabla *users*, que se almacenará en una variable y se utilizará más adelante en la interfaz para hacer del portal más personalizado a cada usuario.

```
// --- TUS METAS (tabla goals) con progreso calculado según el tipo ---
$goals = [];
$sql = "
  SELECT
    g.id,
    g.name_goal AS name,
    g.goal_type,
    g.due_at,
    /* progreso */
    COALESCE(
      CASE
        WHEN g.goal_type='NUMERIC'
             AND g.start_value IS NOT NULL
             AND g.target_value IS NOT NULL
             AND g.start_value <> g.target_value
        THEN ROUND(
          CASE WHEN g.direction='DOWN'
               THEN ((g.start_value - COALESCE(g.current_value, g.start_value))
                     / NULLIF(g.start_value - g.target_value,0)) * 100
               ELSE ((COALESCE(g.current_value, g.start_value) - g.start_value)
                     / NULLIF(g.target_value - g.start_value,0)) * 100
          END, 2
        )
        WHEN g.goal_type='TASK_LIST' THEN (
          SELECT
            CASE WHEN SUM(t.weight) > 0
                 THEN ROUND(SUM(CASE WHEN t.done THEN t.weight ELSE 0 END)
                            / SUM(t.weight) * 100, 2)
                 ELSE 0 END
          FROM task t WHERE t.goal_id = g.id
        )
      END, 0
    ) AS progress
  FROM goals g
  WHERE g.user_id = ?
  HAVING progress < 100
  ORDER BY (g.due_at IS NULL), g.due_at ASC
";
```

Figura 56: Consulta SQL para obtener las metas y calcular su progreso.

La segunda figura muestra el bloque de código encargado de obtener las metas personales del usuario desde la tabla *goals*, calculando para cada una de ellas el porcentaje de progreso. La consulta SQL utiliza una expresión CASE que distingue entre metas numéricas y metas basadas en lista de tareas. En el primer caso, el porcentaje se calcula a partir de los valores inicial, actual y objetivo, teniendo en cuenta si la dirección del progreso es ascendente o descendente. En el segundo caso, se realiza una subconsulta sobre la tabla *task* que suma los pesos de las tareas completadas frente al total de tareas de la meta. El resultado se redondea a dos decimales y se limita a metas con progreso inferior al 100 %, ordenadas por fecha de vencimiento. Todos estos registros se cargarán en el array *$goals*, que es el que alimenta el listado mostrado al usuario.

```html
<!DOCTYPE html>
<html lang="es">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Portal Aurora Goals</title>
  <link rel="icon" type="image/x-icon" href="../images/foto-5.png">
  <link rel="stylesheet" href="../style/user_portal.css">
  <link rel="stylesheet" href="../style/header.css">

</head>
```

Figura 57: Estructura HTML y enlaces a estilos en la cabecera de user_portal.php.

En la tercera figura aparece la cabecera HTML del documento. Aquí se define el idioma de la página, la codificación de caracteres y la metaetiqueta viewport para que el diseño se adapte correctamente a distintos anchos de pantalla. También se establece el título “Portal Aurora Goals”, se incluye el icono de la aplicación y se enlazan las hojas de estilo *user_portal.css* y *header.css*, encargadas respectivamente del diseño específico de esta página y del encabezado común reutilizado en el resto de las vistas.

```php
<section class="metas">
  <div class="metas_header">
    <h1 class="metas_title">Tus metas:</h1>
  </div>

  <?php if ($user_id && count(value: $goals) > 0): ?>
    <ul class="metas_list">
      <?php foreach ($goals as $g):
        $id = (int) $g['id'];
        $name = htmlspecialchars(string: $g['name']);
        // Normaliza progreso (por si llega null o fuera de rango)
        $progress = max(value: 0, values: min(value: 100, values: (int) ($g['progress'] ?? 0)));
        ?>
        <li class="metas_item">
          <a class="metas_link" href="../pages/goal_page.php?id=<?= $id ?>">
            <?= $name ?>
          </a>

          <span class="metas_progress-text">
            <?= $progress ?>%
          </span>

          <div class="metas_progressbar">
            <div class="metas_progressbar-fill" style="width: <?= $progress ?>%;"></div>
          </div>
        </li>
      <?php endforeach; ?>
    </ul>
    <div class="metas_actions">
      <a href="../pages/new_goal.php" id="nueva-meta" class="btn-primaria">Nueva meta</a>
    </div>
  <?php elseif ($user_id): ?>
    <div class="metas_empty">
      <p>No tienes metas aún.</p>
      <a href="../pages/new_goal.php" class="btn-primaria">Crear tu primera meta</a>
    </div>
  <?php endif; ?>


</section>
```

Figura 58: Generación dinámica del listado de metas y sus barras de progreso.

La cuarta figura recoge la sección principal dedicada al listado de metas. El bloque *<section class="metas">* comienza con un encabezado que muestra el título "Tus metas:". A continuación, mediante PHP, se comprueba si el usuario tiene metas cargadas en el array *$goals*. En caso afirmativo, se recorre el array con un *foreach*, normalizando el nombre de cada meta y limitando el porcentaje de progreso a un rango entre 0 y 100. Por cada meta se genera un elemento de lista con tres partes: un enlace que lleva a la página de detalle *goal_page.php*, un texto que muestra el porcentaje y una barra de progreso cuyo ancho en CSS se establece dinámicamente en función de dicho porcentaje. Si el usuario aún no tiene metas, se muestra un bloque alternativo con un mensaje informativo y un botón que enlaza a la página *new_goal_page.php* para crear la primera meta.

```css
.metas {
  width: 80%;
  margin: 1rem auto;
  color: var(--texto);
  font-family: 'Outfit', sans-serif;
}

/* Header de la sección */
.metas_header {
  display: flex;
  align-items: center;
  justify-content: space-between;
  gap: 1rem;
  padding: 0 1%;
}

.metas_title {
  margin: 0;
  font-weight: 700;
  color: var(--negro);
  font-size: 1.6rem;
}

.metas_actions {
  display: flex;
  gap: 0.6rem;
}
```

Figura 59: Estilos principales de la sección de metas en user_portal.css.

Por último, la quinta figura muestra parte de la hoja de estilos user_portal.css aplicada a esta sección. La clase *.metas* define un ancho máximo del contenido, centrado mediante *margin:* auto, y fija la tipografía general. El bloque *.metas_header* utiliza un contenedor flexible (display: flex) para alinear el título y el botón de acción en una misma línea, distribuidos a ambos extremos. Las clases *.metas_title* y *.metas_actions* ajustan el tamaño de la fuente, el peso del texto y el espaciado entre elementos, asegurando que el encabezado resulte legible y equilibrado. En conjunto, este código CSS es el responsable de que la información generada por PHP se presente en un formato limpio y ordenado, con un diseño coherente con el resto de la aplicación.

- Panel resumen del portal de usuario

El portal de usuario incluye, en su parte superior, un bloque de resumen que ayuda a tener una visión rápida del estado de las metas. Este bloque combina tres elementos: un calendario mensual donde se marcan los días actuales, un listado de próximas metas (metas personales con fecha de vencimiento cercana) y un listado de próximas tareas pendientes. Todo ello se genera dinámicamente a partir de la base de datos y se presenta con un diseño compacto para que el usuario pueda priorizar qué hacer nada más entrar en la aplicación.

```
<section class="resumen">
  <div class="bloque-uno">
    <span class="calendario">
      <div class="cal">
        <div class="cal-head">
          <button id="prev"><</button>
              <h3 id="label"></h3>
              <button id="next">></button>
        </div>
        <div class="cal-grid cal-week">
          <span>L</span><span>M</span><span>X</span><span>J</span><span>V</span><span>S</span><span>D</span>
        </div>
        <div class="cal-grid" id="days"></div>
      </div>
    </span>
```

Figura 60: Estructura HTML del bloque de calendario.

En esta figura se muestra la estructura HTML del bloque de resumen. Dentro de la sección <*section class="resumen*"> se define un contenedor bloque-uno que agrupa el calendario y, a su derecha, los listados de próximas metas y tareas. El elemento <*span class="calendario"*> contiene la caja del calendario (*div.cal*), que a su vez tiene una cabecera (*cal-head*) con dos botones de navegación *(prev y next*) y una etiqueta <*h3 id="label"*> donde se mostrará el mes y el año actuales. Debajo se define una fila fija con las abreviaturas de los días de la semana y un contenedor vacío (*div id="days"*) donde se insertarán dinámicamente las fechas mediante JavaScript.

```php
<!-- Metas que expiran pronto -->
<span class="proximos">
  <h3 style="margin:0 0 .5rem 0;">Próximas metas:</h3>
  <ul style="list-style:none; padding:0; margin:0; display:flex; flex-direction:column; gap:.35rem;">
    <?php
    $soon_goals = [];
    if (
      $stmt = $db->prepare(query: "
    SELECT id, name_goal, due_at, progress
    FROM goals
    WHERE user_id = ?
    AND due_at IS NOT NULL
    AND due_at >= NOW()
    AND progress != 100
    ORDER BY due_at ASC
    LIMIT 6
")
    ) {
      $stmt->bind_param(types: "i", var: &$user_id);
      $stmt->execute();
      $res = $stmt->get_result();
      while ($row = $res->fetch_assoc()) {
        $soon_goals[] = $row;
      }
      $stmt->close();
    }
    if (empty($soon_goals)): ?>
      <li>Sin metas próximas</li>
    <?php else:
      foreach ($soon_goals as $g):
        // Formato de fecha (dd/mm/aaaa). Ajusta si prefieres otro.
        $dueFmt = '';
        if (!empty($g['due_at'])) {
          try {
            $dueFmt = (new DateTime(datetime: $g['due_at']))->format(format: 'd/m/Y');
          } catch (Exception $e) {
            $dueFmt = $g['due_at'];
          }
        }
        ?>
        <li>
          <a href="../pages/goal_page.php?id=<?= (int) $g['id'] ?>">
            <strong><?= htmlspecialchars(string: $g['name_goal']) ?></strong>
            <?php if ($dueFmt): ?> <span style="color: grey">
                (<?= htmlspecialchars(string: $dueFmt) ?>)</span><?php endif; ?>
          </a>
        </li>
      <?php endforeach; endif; ?>
  </ul>
</span>
```

Figura 61: Generación del listado de próximas metas con PHP y SQL.

En esta segunda figura corresponde al código PHP que genera el listado de próximas metas dentro del mismo bloque de resumen. Tras el encabezado "Próximas metas", se declara un array *$soon_goals* y se ejecuta una consulta preparada sobre la tabla *goals* filtrando por el usuario actual, por metas con fecha de vencimiento futura (*due_at > NOW()*), con fecha definida y cuyo progreso sea distinto de 100. Los resultados se ordenan por fecha ascendente y se limita el número de elementos para mostrar como máximo seis metas.

Si la consulta devuelve resultados, se recorren en un *foreach*, formateando la fecha de vencimiento con la clase *DateTime* y generando un elemento <li> por meta. Cada entrada contiene un enlace a la página de detalle de la meta (*goal_page.php*), el nombre de la meta y, opcionalmente, la fecha formateada en gris. Si no hay metas próximas, se muestra un único <*li*> con el mensaje "Sin metas próximas".

```
    <!-- Tareas que expiran pronto -->
    <span class="notas">
      <h3 style="margin:0 0 .5rem 0;">Próximas tareas:</h3>
      <ul style="list-style:none; padding:0; margin:0; display:flex; flex-direction:column; gap:.35rem;">
        <?php
        $soon_tasks = [];
        $sql = "
  SELECT t.id, t.name_task, t.due_at, t.done, g.id AS goal_id, g.name_goal
  FROM task t
  JOIN goals g ON g.id = t.goal_id
  WHERE g.user_id = ?
    AND t.due_at IS NOT NULL
    AND t.due_at >= NOW()
    AND t.done = 0
  ORDER BY t.due_at ASC
  LIMIT 5
";
        if ($stmt = $db->prepare(query: $sql)) {
          $stmt->bind_param(types: 'i', var: &$user_id);
          $stmt->execute();
          $res = $stmt->get_result();
          while ($row = $res->fetch_assoc()) {
            $soon_tasks[] = $row;
          }
          $stmt->close();
        }

        if (empty($soon_tasks)): ?>
          <li>Sin tareas próximas</li>
        <?php else:
          foreach ($soon_tasks as $t):
            // Formato de fecha (dd/mm/aaaa)
            $dueFmt = '';
            if (!empty($t['due_at'])) {
              try {
                $dueFmt = (new DateTime(datetime: $t['due_at']))->format(format: 'd/m/Y');
              } catch (Exception $e) {
                $dueFmt = $t['due_at'];
              }
            }
            ?>
```

Figura 62: Consulta y carga de próximas tareas pendientes desde la base de datos.

La tercera y cuarta figura muestran el código encargado de construir el listado de próximas tareas. La estructura es muy similar al caso anterior: se define un encabezado "Próximas tareas" y un array *$soon_tasks*. La consulta SQL se realiza sobre la tabla *task*, uniendo con *goals* para poder recuperar también el nombre de la meta a la que pertenece cada tarea. Se filtra por usuario, por tareas con fecha futura (*t.due_at > NOW()*), que aún no están completadas *(t.done = 0)* y se ordenan por fecha de vencimiento, limitando el resultado a cinco tareas.

Una vez obtenidos los resultados, si no hay tareas pendientes se muestra el mensaje “Sin tareas próximas”. En caso contrario, el *foreach* recorre cada tarea calculando, de nuevo, una versión formateada de la fecha. Cada <li> contiene un enlace a la meta asociada (*goal_page.php?id),* mostrando en negrita el nombre de la tarea y en cursiva el nombre de la meta. Si existe fecha límite, se añade a continuación en gris, permitiendo al usuario visualizar de un vistazo qué tareas se vencen antes y dentro de qué meta se encuentran.

```
            <li>
              <a href="../pages/goal_page.php?id=<?= (int) $t['goal_id'] ?>">
                <strong><?= htmlspecialchars(string: $t['name_task']) ?></strong>
                <em><?= htmlspecialchars(string: $t['name_goal']) ?></em>
                <?php if ($dueFmt): ?>
                  <?php if ($dueFmt): ?>
                    <span style="color: grey"> (<?= htmlspecialchars(string: $dueFmt, flags: ENT_QUOTES, encoding: 'UTF-8') ?>)</span>
                  <?php endif; ?>
                <?php endif; ?>
              </a>
            </li>
          <?php endforeach; endif; ?>
        </ul>
      </span>

    </div>
</section>
```

Figura 63: Generación del listado HTML de próximas tareas con enlace a sus metas.

En las siguientes dos figuras se muestra el script JavaScript completo encargado de generar el calendario del bloque de resumen y permitir la navegación entre meses.

En la primera figura, el código se encapsula en una función autoejecutable para no contaminar el ámbito global. Se obtienen referencias a los elementos del DOM donde se pintará el calendario: el contenedor de días (*days*), la etiqueta del mes (*label*) y los botones de navegación (*prev y next*). Después se crean dos objetos Date:

- *view*, que representa el mes actualmente visible en el calendario;
- today, que almacena la fecha actual, normalizada a medianoche para facilitar las comparaciones.

Dentro de la función *render*() se limpian los días previamente generados y se extraen el año y el mes de *view*. A continuación se actualiza el texto de la cabecera con el mes y el año en formato legible (*toLocaleDateString* en español). Finalmente, se calculan varios valores clave para construir la cuadrícula: el índice del primer día de la semana (*firstDow*), el número de días del mes actual (*daysInMonth*) y el

número de días del mes anterior (*prevMonthDays*), que se usarán para rellenar las celdas.

```
<script>
  (function () {
    const daysEl = document.getElementById('days');
    const label = document.getElementById('label');
    const prev = document.getElementById('prev');
    const next = document.getElementById('next');

    let view = new Date();
    view.setDate(1);
    const today = new Date();
    today.setHours(0, 0, 0, 0);

    function render() {
      daysEl.innerHTML = '';
      const year = view.getFullYear();
      const month = view.getMonth();

      label.textContent = view.toLocaleDateString('es-ES', { month: 'long', year: 'numeric' }).replace(' de ', ' ');

      const firstDow = (view.getDay() + 6) % 7; // Lunes=0
      const daysInMonth = new Date(year, month + 1, 0).getDate();
      const prevMonthDays = new Date(year, month, 0).getDate();
```

Figura 64: Inicialización del calendario y cálculo de la información del mes en el script.

En la segunda figura aparece el núcleo de la generación de celdas. Primero, un bucle recorre los días anteriores al inicio del mes y crea elementos *div* con la clase *day muted*, que representan los últimos días del mes anterior en un tono atenuado. Después, otro bucle genera los días reales del mes actual: para cada número de día se crea un *div* con la clase *day* y, si coincide con la fecha almacenada en *today*, se añade la clase *today* y un título "hoy" para resaltarlo visualmente.

Una vez generados los días del mes, se calcula si la cuadrícula necesita celdas adicionales para completar la última semana (*remainder*). Si es así, se añaden más *div* con la clase *day muted*, simulando los primeros días del mes siguiente. Por último, se registran los manejadores de eventos de los botones *prev* y *next*: al hacer clic se modifica el mes de *view* (restando o sumando uno) y se vuelve a invocar a *render*(), actualizando por completo el calendario sin recargar la página.

```
    // Días previos
    for (let i = firstDow; i > 0; i--) {
      const d = document.createElement('div');
      d.className = 'day muted';
      d.textContent = (prevMonthDays - i + 1);
      daysEl.appendChild(d);
    }

    // Días del mes actual
    for (let dNum = 1; dNum <= daysInMonth; dNum++) {
      const d = document.createElement('div');
      d.className = 'day';
      d.textContent = dNum;

      const cellDate = new Date(year, month, dNum);
      cellDate.setHours(0, 0, 0, 0);

      if (cellDate.getTime() === today.getTime()) {
        d.classList.add('today');
        d.title = 'Hoy';
      }

      daysEl.appendChild(d);
    }

    // Relleno final
    const remainder = daysEl.children.length % 7;
    if (remainder !== 0) {
      const fill = 7 - remainder;
      for (let i = 1; i <= fill; i++) {
        const d = document.createElement('div');
        d.className = 'day muted';
        d.textContent = i;
        daysEl.appendChild(d);
      }
    }
  }

  prev.addEventListener('click', () => { view.setMonth(view.getMonth() - 1); render(); });
  next.addEventListener('click', () => { view.setMonth(view.getMonth() + 1); render(); });
  render();
})();
</script>
```

Figura 65 Generación dinámica de las celdas del calendario y navegación entre meses.

La última figura muestra parte de la hoja de estilos dedicada al calendario dentro de user_portal.css. La clase *.calendario* se define como un contenedor flexible en columna, con un ancho reducido para que pueda colocarse junto a los listados de metas y tareas. La clase *.cal* aplica un fondo blanco, bordes redondeados y una sombra suave para que el calendario destaque como una tarjeta independiente. La cabecera *.cal-head* utiliza *display: flex* para alinear los botones de navegación y el título del mes en una misma línea, mientras que las reglas de *.cal-head h3* ajustan el peso y el color de la fuente. Los botones del calendario se estilizan con bordes redondeados, un degradado de color azul y una transición suave, de manera que al

pasar el ratón por encima se perciba un efecto visual agradable y coherente con el resto de la interfaz.

```css
/* ===== CALENDARIO ===== */
.calendario {
  display: flex;
  flex-direction: column;
  width: 22%;
  margin-left: 3%;
  box-sizing: border-box;
}

.cal {
  flex: 1;
  width: 100%;
  max-height: none;
  font-family: 'Outfit', sans-serif;
  padding: 16px;
  border-radius: 14px;
  background: var(--blanco);
  box-shadow: 0 8px 24px #00000014;
  box-sizing: border-box;
}

.cal-head {
  display: flex;
  align-items: center;
  justify-content: space-between;
  margin-bottom: 8px;
}

.cal-head h3 {
  margin: 0;
  font-weight: 700;
  text-transform: capitalize;
  color: var(--azul-oscuro);
}

.cal-head button {
  border: none;
  border-radius: 10px;
  padding: 6px 10px;
  cursor: pointer;
  font-weight: 700;
  background: linear-gradient(135deg, var(--azul-verdoso), var(--azul-oscuro));
  color: var(--blanco);
  transition: 0.2s;
}
```

Figura 66: Estilos CSS aplicados al calendario en el bloque de resumen.

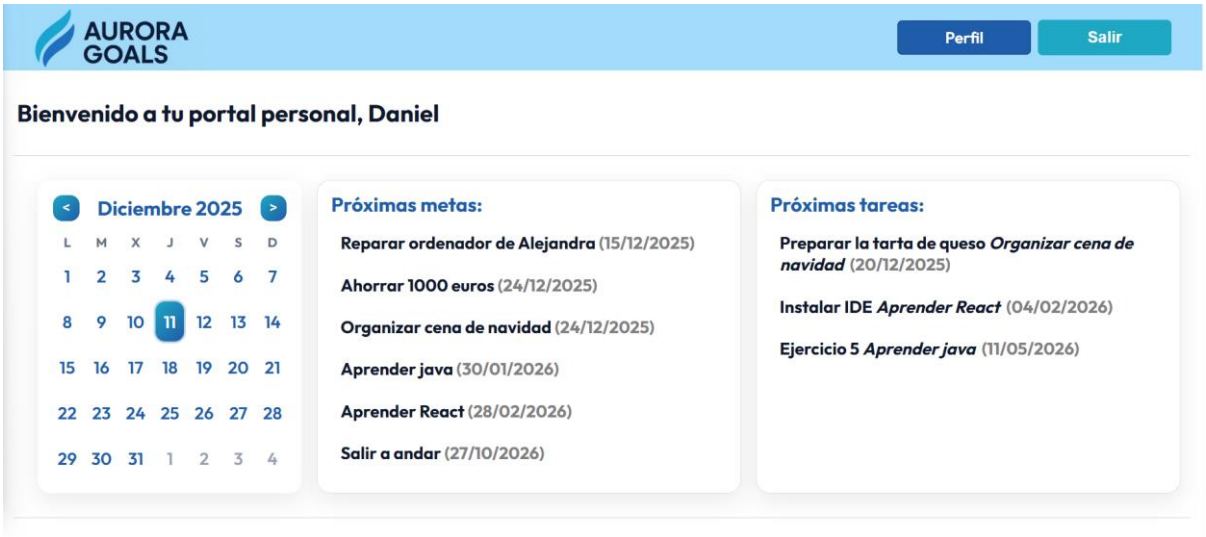

Figura 67: Resultado final del panel resumen de portal del usuario.

4.4.2. Página de inicio de sesión

La página de inicio de sesión permite al usuario acceder a la zona privada de la aplicación introduciendo su correo y contraseña. Además del formulario básico, se ha añadido un pequeño comportamiento interactivo en JavaScript para mostrar y ocultar la contraseña mientras se mantiene pulsado un botón con icono de "ojo", mejorando así la usabilidad sin comprometer la seguridad.

```
<section class="card">
  <h2>Iniciar sesión</h2>
  <form action="../logic/log_in.php" method="post">
    <label>Email
      <input type="email" name="email" required>
    </label>

    <label>Contraseña
      <div>
        <input type="password" id="password" name="password" required>
        <button type="button" onclick="togglePassword('password')">Show</button>
      </div>
    </label>
    <?php if ($flash): ?>
      <div class="flash"><?= htmlspecialchars(string: $flash) ?></div>
    <?php endif; ?>
    <button type="submit">Entrar</button>
  </form>

  <p style="margin-top:10px">¿No tienes cuenta?
    <a href="./register_page.php">Regístrate</a>
  </p>


</section>
```

Figura 68: Formulario HTML de inicio de sesión.

En la primera figura se muestra el bloque principal del formulario. El contenido se organiza dentro de una sección con clase *card*, que se estiliza como una tarjeta centrada en pantalla. El formulario envía los datos mediante POST al script *../logic/log_in.php*, encargado de validar las credenciales (explicado en el apartado 4.3.2.2). Se definen dos campos obligatorios: un input de tipo email para el correo y un input de tipo *password* para la contraseña. Junto al campo de contraseña se coloca un botón de tipo *button* que inicialmente llama a

togglePassword('password'), aunque más adelante este comportamiento se sustituirá por el manejador definido en JavaScript. Debajo de los campos, el bloque PHP comprueba si existe un mensaje "flash" en la variable *$flash* y, de ser así, lo muestra en un *div* con clase *flash* para informar al usuario de errores previos (por ejemplo, credenciales incorrectas). El formulario se completa con un botón de envío "Entrar" y un pequeño texto que enlaza con la página de registro para los usuarios que todavía no tienen cuenta.

```
<script>
  document.addEventListener('DOMContentLoaded', () => {
    const eyeButtons = document.querySelectorAll('.card label button[type="button"]');

    eyeButtons.forEach(btn => {
      btn.removeAttribute('onclick');
      const wrapper = btn.closest('div');
      const input = wrapper ? wrapper.querySelector('input') : null;
      if (!input) return;

      btn.classList.add('toggle-eye');
      btn.innerHTML = `
    <span>Mostrar contraseña mientras se mantiene presionado</span>
    <svg class="eye" viewBox="0 0 24 24" aria-hidden="true">
      <path d="M2 12s3.5-7 10-7 10 7 10 7-3.5 7-10 7-10-7-10-7Z"/>
      <circle cx="12" cy="12" r="3.5"/>
    </svg>
    <svg class="eye-off" viewBox="0 0 24 24" aria-hidden="true">
      <path d="M3 3l18 18"/>
      <path d="M2 12s3.5-7 10-7c2.4 0 4.4.8 6 2"/>
      <path d="M22 12s-3.5 7-10 7c-2.4 0-4.4-.8-6-2"/>
      <path d="M9.5 9.5a3.5 3.5 0 004.9 4.9"/>
    </svg>
  `;

      const show = () => {
        input.type = 'text';
        btn.classList.add('is-on');
      };
      const hide = () => {
        input.type = 'password';
        btn.classList.remove('is-on');
      };

      btn.addEventListener('mousedown', show);
      btn.addEventListener('touchstart', show);
      ['mouseup', 'mouseleave', 'touchend', 'touchcancel'].forEach(ev => {
        btn.addEventListener(ev, hide);
      });
    });
  });
</script>
```

Figura 69: Script JavaScript para mostrar y ocultar temporalmente la contraseña.

4.4.3. Página de detalle de meta personal

La página *goal_page.php* muestra toda la información de una meta personal y permite, según el tipo de meta, eliminarla, actualizar su progreso numérico o gestionar sus tareas asociadas. A continuación, se describen los fragmentos más relevantes del código y cómo se relacionan entre sí.

```php
<?php
// pages/goal_page.php
session_start();
require_once __DIR__ . '/../logic/db_connection.php';

// ---------- Autenticación ----------
$userId = $_SESSION['uid'] ?? $_SESSION['user_id'] ?? null;
if (!$userId) {
    $_SESSION['flash'] = 'Inicia sesión para ver tus metas.';
    header(header: 'Location: ../pages/log_in_page.php');
    exit;
}

// ---------- Parámetro ----------
$goalId = isset($_GET['id']) ? (int) $_GET['id'] : 0;
if ($goalId <= 0) {
    $_SESSION['flash'] = 'Meta no encontrada.';
    header(header: 'Location: ../pages/main_page.php');
    exit;
}
```

Figura 70: Comprobación de sesión y validación del parámetro id.

En la primera figura se ve el inicio del script. Se arranca la sesión y se incluye el fichero de conexión a la base de datos. A continuación se obtiene el identificador del usuario desde la sesión ($_SESSION['uid'] o $_SESSION['user_id']) y, si no existe, se guarda un mensaje "flash" indicando que es necesario iniciar sesión y se redirige a la página de *login*. Después se recupera el parámetro id de la URL, que identifica la meta a mostrar; si el parámetro no existe o no es válido, se vuelve a la página principal con un mensaje de error. Esta parte garantiza que solo se pueda acceder a la vista de meta estando autenticado y con un identificador de meta correcto.

La figura 71 muestra cómo se cargan los datos de la meta desde la base de datos. Se construye una consulta preparada sobre la tabla *goals* que recupera todos los campos relevantes (titulo, descripción, tipo de meta, estado, visibilidad, valores numéricos, unidad, dirección, fechas, etc.) filtrando por el id recibido. Tras ejecutar la consulta y obtener la fila correspondiente, se verifica que la meta exista y que el campo *user_id* coincida con el identificador del usuario autenticado. Si no se cumple alguna de estas condiciones, el script termina redirigiendo a la página principal y mostrando un mensaje de "Meta no encontrada o no tienes permisos". Así se evita que un usuario pueda consultar metas que no le pertenecen.

```php
// ---------- Cargar meta ----------
$goal = null;
$sqlGoal = "SELECT id, user_id, name_goal, description_goal, goal_type, status, visibility,
                   start_value, target_value, current_value, unit, direction,
                   created_at, due_at
            FROM goals
            WHERE id = ?
            LIMIT 1";
if ($st = $db->prepare(query: $sqlGoal)) {
    $st->bind_param(types: "i", var: &$goalId);
    $st->execute();
    $goal = $st->get_result()->fetch_assoc();
    $st->close();
}
if (!$goal || (int) $goal['user_id'] !== (int) $userId) {
    $_SESSION['flash'] = 'Meta no encontrada o no tienes permisos.';
    header(header: 'Location: ../pages/main_page.php');
    exit;
}
```

Figura 71: Carga de la meta desde la tabla goals y verificación de permisos sobre ella.

En la figura 72 aparece el bloque que gestiona las acciones enviadas mediante formularios POST. Nada más comenzar, se comprueba que en la petición exista un *csrf_token* y que coincida con el valor almacenado en la sesión mediante una comparación segura (*hash_equals*). Si el token no es válido, se informa al usuario y se recarga la página de la meta, evitando procesar acciones potencialmente maliciosas. Después se lee el parámetro *action*, que indica qué operación se desea realizar.

En este fragmento se detalla el caso de eliminación de meta (action === 'delete'). Se prepara una sentencia DELETE sobre la tabla *goals* filtrando por el identificador de la meta y del usuario, de forma que solo el propietario pueda borrarla. Tras ejecutar la sentencia, se comprueba si se ha afectado al menos una fila; si el borrado ha sido correcto, se elimina el token CSRF de la sesión, se guarda un mensaje de éxito y se redirige a la página principal. Si no se ha podido borrar, se informa al usuario con un mensaje de error y se permanece en la página de la meta. Esta estructura se reutiliza para el resto de las acciones POST.

```php
// ---------- Acciones POST ----------
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    if (!isset($_POST['csrf_token']) || !hash_equals(known_string: $csrf, user_string: $_POST['csrf_token'])) {
        $_SESSION['flash'] = 'Token de seguridad inválido.';
        header(header: "Location: ../pages/goal_page.php?id=" . $goalId);
        exit;
    }

    $action = $_POST['action'] ?? '';

    // Eliminar meta
    if ($action === 'delete') {
        if ($st = $db->prepare(query: "DELETE FROM goals WHERE id = ? AND user_id = ?")) {
            $st->bind_param(types: "ii", var: &$goalId, vars: &$userId);
            $st->execute();
            $ok = $st->affected_rows > 0;
            $st->close();
            if ($ok) {
                unset($_SESSION['csrf_token']);
                $_SESSION['flash'] = 'Meta eliminada correctamente.';
                header(header: 'Location: ../pages/main_page.php');
                exit;
            } else {
                $_SESSION['flash'] = 'No se pudo eliminar la meta.';
                header(header: "Location: ../pages/goal_page.php?id=" . $goalId);
                exit;
            }
        }
    }
```

Figura 72: Gestión de acciones POST y eliminación segura de una meta personal.

```php
// Añadir progreso (NUMERIC)
if ($action === 'numeric_add_progress' && $canEdit && $goal['goal_type'] === 'NUMERIC') {
    $amount = isset($_POST['amount']) ? (float) $_POST['amount'] : 0.0;
    // $note = trim($_POST['note'] ?? ''); // si en el futuro quieres guardar un log

    if ($amount > 0) {
        $dir = $goal['direction'] ?? 'UP';
        $signedDelta = ($dir === 'DOWN') ? -$amount : $amount;

        $prevCurrent = ($goal['current_value'] !== null)
            ? (float) $goal['current_value']
            : (float) $goal['start_value'];

        $newCurrent = $prevCurrent + $signedDelta;

        // Limitar a [min(start, target), max(start, target)]
        $start = (float) $goal['start_value'];
        $target = (float) $goal['target_value'];
        $minB = min(value: $start, values: $target);
        $maxB = max(value: $start, values: $target);
        if ($newCurrent < $minB)
            $newCurrent = $minB;
        if ($newCurrent > $maxB)
            $newCurrent = $maxB;

        if ($st = $db->prepare(query: "UPDATE goals SET current_value = ? WHERE id = ? AND user_id = ?")) {
            $st->bind_param(types: "dii", var: &$newCurrent, vars: &$goalId, $userId);
            $st->execute();
            $st->close();
        }
    }
    header(header: "Location: ../pages/goal_page.php?id=" . $goalId);
    exit;
}
```

Figura 73: Actualización del valor actual en metas numéricas.

La figura 73 recoge la lógica específica para actualizar el progreso de metas numéricas. Esta parte solo se ejecuta cuando la acción es *numeric_add_progress*, el usuario tiene permiso para editar y la meta es de tipo NUMERIC. Se obtiene la cantidad introducida en el formulario (*amount*) y se determina el signo en función de la dirección de la meta (UP o DOWN), de modo que aumentar progreso en una meta que "baja" (por ejemplo, reducir peso) reste en lugar de sumar. A partir del valor actual (o del valor inicial si todavía no hay progreso registrado) se calcula el nuevo valor, y luego se limita para que nunca quede fuera del rango definido entre el valor inicial y el objetivo (*min/max*). Finalmente se prepara y ejecuta un UPDATE sobre la tabla *goals* para almacenar el *current_value* resultante, siempre filtrando por id y *user_id* para reforzar los permisos. Tras la actualización, la página se recarga redirigiendo de nuevo a la misma meta.

```php
<?php else: ?>
    <?php
    // Cargar tareas (solo lectura)
    $tasks = [];
    $sqlTasks = "SELECT id, name_task, description_task, done, weight, due_at
                 FROM task
                 WHERE goal_id = ?
                 ORDER BY position, id";
    if ($st = $db->prepare(query: $sqlTasks)) {
        $st->bind_param(types: "i", var: &$goalId);
        $st->execute();
        $res = $st->get_result();
        while ($row = $res->fetch_assoc())
            $tasks[] = $row;
        $st->close();
    }
    ?>
    <div class="tasklist">
        <h2 class="tasklist-title">Tareas</h2>
        <?php if (count(value: $tasks) === 0): ?>
            <div class="tasklist-empty">Aún no hay tareas.</div>
        <?php else: ?>
            <ul class="tasklist-list">
                <?php foreach ($tasks as $t):
                    $isDone = (int) $t['done'] === 1;
                    $dFmt = '';
                    if (!empty($t['due_at'])) {
                        try {
                            $dFmt = (new DateTime(datetime: $t['due_at']))->format(format: 'd/m/y');
                        } catch (Exception $e) {
                            $dFmt = h(v: $t['due_at']);
                        }
                    }
                    ?>
```

Figura 74: Carga de las tareas asociadas a una meta de tipo lista.

En la figura 74 se muestra el bloque dedicado a metas de tipo lista de tareas. En el caso de metas no numéricas, una vez resueltas las acciones POST se procede a cargar las tareas asociadas. Se declara un array *$tasks* y se ejecuta una consulta preparada sobre la tabla *task* filtrando por *goal_id* y ordenando por position e id para mantener un orden estable. Los resultados se guardan en el array y, a continuación, se genera la sección * *. Si no hay tareas, se muestra un mensaje “Aún no hay tareas”; en caso contrario, se recorre el array y se construye una lista <ul> donde cada tarea incluye su estado de completado, la descripción y, si existe, la fecha de vencimiento formateada mediante *DateTime*.

```php
            <li class="tasklist-item">
                <div class="tasklist-info">
                    <div class="tasklist-name">
                        <?= h(v: $t['name_task']) ?>
                        <?= ((int) $t['weight'] > 1) ? ": Duración " . (int) $t['weight'] : "" ?>
                        <?= $isDone ? ' - <em>Completada</em>' : '' ?>
                    </div>
                    <div class="tasklist-desc"><?= h(v: $t['description_task']) ?></div>
                    <?php if ($dFmt): ?>
                        <div class="tasklist-due">Vence: <?= h(v: $dFmt) ?></div><?php endif; ?>
                </div>

                <?php if ($canEdit): ?>
                    <div class="tasklist-actions">
                        <!-- Toggle -->
                        <form method="post" action="">
                            <input type="hidden" name="csrf_token" value="<?= h(v: $csrf) ?>">
                            <input type="hidden" name="action" value="task_toggle">
                            <input type="hidden" name="task_id" value="<?= (int) $t['id'] ?>">
                            <button type="submit" class="task-toggle <?= $isDone ? 'is-done' : '' ?>">
                                <span class="task-toggle__box"></span>
                            </button>
                        </form>

                        <!-- Delete -->
                        <form method="post" action="" onsubmit="return confirm('¿Eliminar esta tarea?');">
                            <input type="hidden" name="csrf_token" value="<?= h(v: $csrf) ?>">
                            <input type="hidden" name="action" value="task_delete">
                            <input type="hidden" name="task_id" value="<?= (int) $t['id'] ?>">
                            <button type="submit" class="task-delete" aria-label="Eliminar tarea">
                                <svg viewBox="0 0 24 24" width="18" height="18" aria-hidden="true">
                                    <path
                                        d="M9 3h6l1 2h4v2H4V5h4l1-2zm2 6v8m2-8v8M6 7h12l-1 12a2 2 0 0 1-2 2H9a2 2 0 0 1-2-2L6 7z"
                                        fill="none" stroke="currentColor" stroke-width="2" stroke-linecap="round"
                                        stroke-linejoin="round" />
                                </svg>
                            </button>
                        </form>
                    </div>
                <?php endif; ?>
            </li>
        <?php endforeach; ?>
    </ul>
<?php endif; ?>
```

Figura 75: Representación de cada tarea, con acciones de completar y eliminar protegidas por token CSRF.

La figura 75 detalla la estructura de cada elemento de la lista de tareas y las acciones disponibles sobre ellas. Dentro de cada *<li class="tasklist-item">* se muestra el nombre de la tarea, su peso (por ejemplo, "Duración 2") y un texto opcional "Completada" cuando el campo done está activo. Debajo se presenta la descripción y, si se ha definido, la fecha de vencimiento. Cuando el usuario tiene permisos de edición *($canEdit),* se añaden dos formularios dentro de la zona de acciones: uno para alternar el estado de la tarea *(task_toggle)* y otro para eliminarla *(task_delete).* Ambos incluyen el *csrf_token*, el identificador de la tarea y un botón de tipo *submit*; en el caso de la eliminación se añade además un *onsubmit* con un mensaje de confirmación y un icono SVG de papelera. Estas acciones se procesan en la parte de manejo de POST (no mostrada en las figuras, pero estructurada de forma similar a la acción de borrado de metas).

4.4.4. Página de metas colaborativas: ranking de aportaciones

En las metas colaborativas de tipo numérico, además del progreso global, la aplicación muestra un ranking de aportaciones por miembro. Esta sección permite ver qué porcentaje del avance total corresponde a cada participante y cuánta cantidad ha aportado en términos absolutos (horas, unidades, etc.). Para ello se combinan una consulta SQL específica y la generación de una lista con barras de porcentaje.

```php
// ---------------- Ranking de aportes (solo NUMÉRICA) ----------------
$contributors = [];
if ($goal['goal_type'] === 'NUMERIC') {
    $sqlC = "
    SELECT u.id AS uid,
           COALESCE(NULLIF(u.name_user,''), u.email) AS name,
           ROUND(SUM(
             CASE g.direction
               WHEN 'UP'   THEN GREATEST(0, c.delta_value) / NULLIF(g.target_value - g.start_value, 0)
               WHEN 'DOWN' THEN GREATEST(0, -c.delta_value) / NULLIF(g.start_value - g.target_value, 0)
             END
           ) * 100, 2) AS pct_contrib,
           ROUND(SUM(c.delta_value), 2) AS raw_delta
    FROM collab_goal_updates c
    JOIN users u ON u.id = c.user_id
    JOIN collab_goals g ON g.id = c.collab_goal_id
    WHERE c.collab_goal_id = ?
    GROUP BY u.id, name
    ORDER BY pct_contrib DESC, name ASC
  ";
    if ($st = $db->prepare(query: $sqlC)) {
        $st->bind_param(types: "i", var: &$goalId);
        $st->execute();
        $res = $st->get_result();
        while ($r = $res->fetch_assoc())
            $contributors[] = $r;
        $st->close();
    }
}
```

Figura 76: Consulta SQL y carga del ranking de aportaciones por usuario en metas colaborativas numéricas.

En la figura 76 se muestra el código PHP encargado de calcular las aportaciones. Solo se ejecuta cuando la meta colaborativa es numérica. Se inicializa el array *$contributors* y se construye una consulta SQL sobre la tabla *collab_goal_updates*, donde se registran todos los incrementos de progreso.

La consulta une (JOIN) esta tabla con *users* para obtener el nombre de cada participante y con *collab_goals* para conocer los valores de referencia de la meta (valor inicial y objetivo).

Mediante una expresión CASE se tiene en cuenta la dirección de la meta (UP o DOWN) y se suman las deltas (*c.delta_value*) transformándolas en un porcentaje de contribución respecto al rango total de la meta. Así se obtiene, para cada usuario, tanto el porcentaje *(pct_contrib)* como la suma bruta de sus aportaciones *(raw_delta)*. Los resultados se agrupan por usuario y se ordenan de mayor a menor porcentaje para mostrar primero a quienes más han contribuido. Finalmente, se ejecuta la consulta preparada, se recorren los resultados y se van guardando en el array *$contributors*.

```php
<!-- Ranking de aportes -->
<div class="contributors">
    <h2>Aportes por miembro</h2>
    <?php if (empty($contributors)): ?>
        <div class="contributors-empty">Todavía no hay aportes registrados.</div>
    <?php else: ?>
        <ul class="contributors-list">
            <?php foreach ($contributors as $c):
                $you = ((int) $c['uid'] === (int) $userId) ? ' (tú)' : '';
                ?>
                <li class="contributors-item">
                    <div class="contributors-row">
                        <div class="contributors-name">
                            <?= h(v: $c['name']) . $you ?>
                        </div>
                        <div class="contributors-stats">
                            <span class="chip chip-pct"><?= h(v: $c['pct_contrib']) ?>%</span>
                            <span class="chip chip-delta"><?= h(v: $c['raw_delta']) ?><?= $unit ?: '' ?></span>
                        </div>
                    </div>
                    <div class="contributors-bar">
                        <div class="contributors-bar_fill"
                            style="width: <?= max(value: 0, values: min(value: 100, values: (float) $c['pct_contrib'])) ?>%;"></div>
                    </div>
                </li>
            <?php endforeach; ?>
        </ul>
    <?php endif; ?>
</div>
```

Figura 77: Generación del listado de aportaciones por miembro y visualización mediante barras de porcentaje.

La figura 77 muestra cómo se representan estos datos en la vista. Dentro del * * se incluye un título “Aportes por miembro” y, a continuación, se comprueba si el array *$contributors* está vacío. Si aún no se han registrado aportaciones, se muestra un mensaje informando de ello. En caso contrario, se genera una lista <ul> donde cada elemento corresponde a un participante.

En cada iteración del *foreach* se calcula la variable *$you* para añadir la etiqueta "(tú)" junto al nombre si el usuario del ranking coincide con el usuario autenticado, lo que facilita reconocerse en la lista. Cada elemento incluye una fila con el nombre, el porcentaje de contribución y la cantidad total aportada (*raw_delta*) seguida de la unidad de la meta. Debajo se muestra una barra de progreso visual: un div cuya anchura en CSS se establece con *style="width: …%",* limitando el valor entre 0 y 100 mediante *max* y *min* para evitar desbordes.

Gracias a esta combinación de datos agregados y representación gráfica, los miembros del equipo pueden ver de forma clara cómo se reparte el esfuerzo dentro de una meta colaborativa.

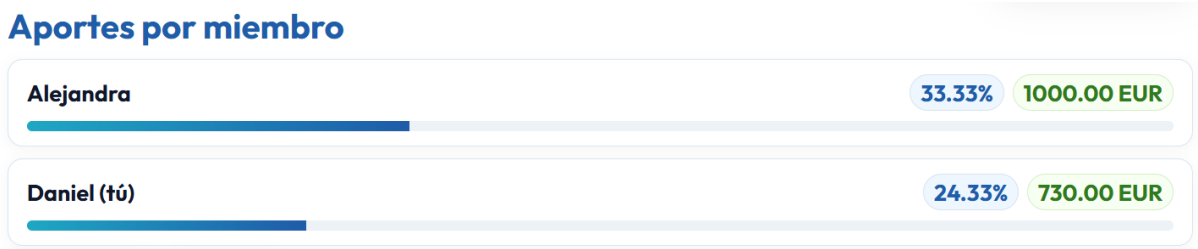

Figura 78: Demostración de la página de aportes por miembro en metas colaborativas.

4.5. Aspectos de la lógica de la aplicación

Además de las funcionalidades específicas de cada página, la aplicación incorpora una serie de mecanismos que afectan a casi todo el código: la gestión de sesiones y el control de acceso, la validación de formularios y el tratamiento de mensajes al usuario, y un conjunto de medidas básicas de seguridad. En este apartado se describen estos aspectos, que no pertenecen a una única pantalla concreta, pero que son fundamentales para el correcto funcionamiento del sistema.

4.5.1. Gestión de sesiones y control de acceso

La aplicación utiliza el sistema de sesiones de PHP para identificar a los usuarios y mantener su estado entre peticiones HTTP. Todos los scripts que requieren

conocer la identidad del usuario comienzan con la llamada a *session_start(),* lo que permite acceder al contenido de *$_SESSION*.

Durante el proceso de inicio de sesión, una vez verificadas las credenciales, se almacena en la sesión el identificador del usuario (uid), junto con otros datos básicos como el correo electrónico y el nombre. Además, se invoca a *session_regenerate_id(true*) para generar un nuevo identificador de sesión y mitigar ataques de fijación de sesión.

Las páginas realizan siempre una comprobación similar al inicio del script: si no existe *$_SESSION['uid'],* se registra un mensaje en la sesión y se redirige al formulario de inicio de sesión. De esta forma se evita que un usuario no autenticado pueda acceder a información privada modificando manualmente la URL.

El control de acceso no se limita a la existencia de sesión: en las operaciones sensibles, como la carga o eliminación de metas, las consultas SQL incluyen siempre una condición adicional por *user_id*. Esto garantiza que cada usuario solo pueda ver y modificar sus propios datos, incluso aunque intentase manipular parámetros como el identificador de la meta en la URL.

Finalmente, el cierre de sesión se implementa mediante el script *log_out.php* (y el formulario integrado en la cabecera), que destruye la sesión, limpia las variables asociadas y redirige al usuario a la página principal. Así se garantiza que, al salir, no quede información residual que permita recuperar el estado anterior.

4.5.2. Medidas básicas de seguridad

Aunque se trata de una aplicación ejecutada en entorno local, se han incorporado varias medidas básicas de seguridad para reducir riesgos habituales en aplicaciones web.

En primer lugar, las contraseñas de los usuarios nunca se almacenan en texto plano. Durante el registro se utiliza la función *password_hash()* de PHP con el algoritmo por defecto recomendado, generando un hash seguro que incluye un salt aleatorio. En el inicio de sesión, la comprobación se realiza mediante *password_verify(),* comparando la contraseña introducida con el hash almacenado. Gracias a este enfoque, incluso si la base de datos se viera comprometida, no se podrían recuperar directamente las contraseñas originales.

Para prevenir inyecciones SQL, todas las operaciones sobre la base de datos que utilizan datos proporcionados por el usuario se realizan mediante sentencias preparadas *($db->prepare()* y *bind_param()*. De este modo, los parámetros se envían separados de la consulta y el motor de MySQL los trata como datos, no como parte del código SQL, neutralizando intentos de manipular la consulta mediante cadenas especialmente construidas.

Estos mecanismos aportan a la aplicación de una base técnica sólida y la preparan para una posible evolución futura hacia entornos de despliegue más exigentes.

5. Resultado final de la aplicación web

En este apartado se recogen distintas capturas de pantalla que muestran el resultado final de la aplicación Aurora Goals y el recorrido típico que realiza un usuario. Se comienza por la página principal pública y los formularios de registro e inicio de sesión, para después mostrar el portal de usuario con su resumen de metas, las pantallas de detalle de metas personales y colaborativas y, finalmente, los formularios de creación de nuevas metas. De esta forma veremos tanto el aspecto visual final de la aplicación web como la navegación básica del sistema.

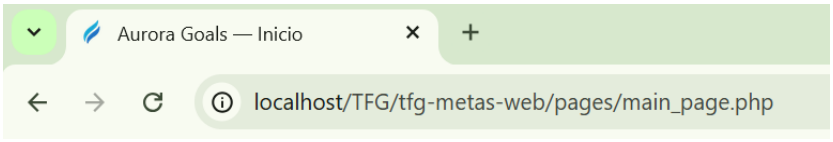

Figura 79: Vista general de la aplicación en el navegador.

Se muestra la aplicación *Aurora Goals* ejecutándose en el navegador en entorno local, confirmando el despliegue correcto sobre XAMPP. Es el punto de partida desde el que el usuario comienza a interactuar con la web. Destacar que se ha cuidado cada detalle de la aplicación, y podemos apreciar que en todas las cabeceras del navegador se verá la imagen característica de *Aurora Goals* y un título que nos indicará en qué página estamos.

Todo esto con el objetivo de que se asemeje a una aplicación profesional, que el usuario se sienta a gusto navegando por la web y que ayude a crear una marca reconocible.

Figura 80: Primer banner de la página principal pública.

La captura muestra la cabecera, con el logotipo, el menú y un primer mensaje de bienvenida que invita al usuario a organizar sus metas. Es la primera impresión visual de la herramienta con un mensaje claro sobre para qué sirve la aplicación Aurora Goals (Establece y organiza tus metas personales).

Figura 81: Segundo banner informativo de la página principal pública.

Segundo banner de la aplicación que destaca la posibilidad de seguir el progreso de forma clara y estructurada. Refuerza el propósito de la aplicación para que el nuevo usuario se registre.

Figura 82: Tercer banner informativo de la página principal pública.

El tercer banner resalta el enfoque en alcanzar objetivos. El texto y los colores mantienen la coherencia de la identidad visual de Aurora Goals.

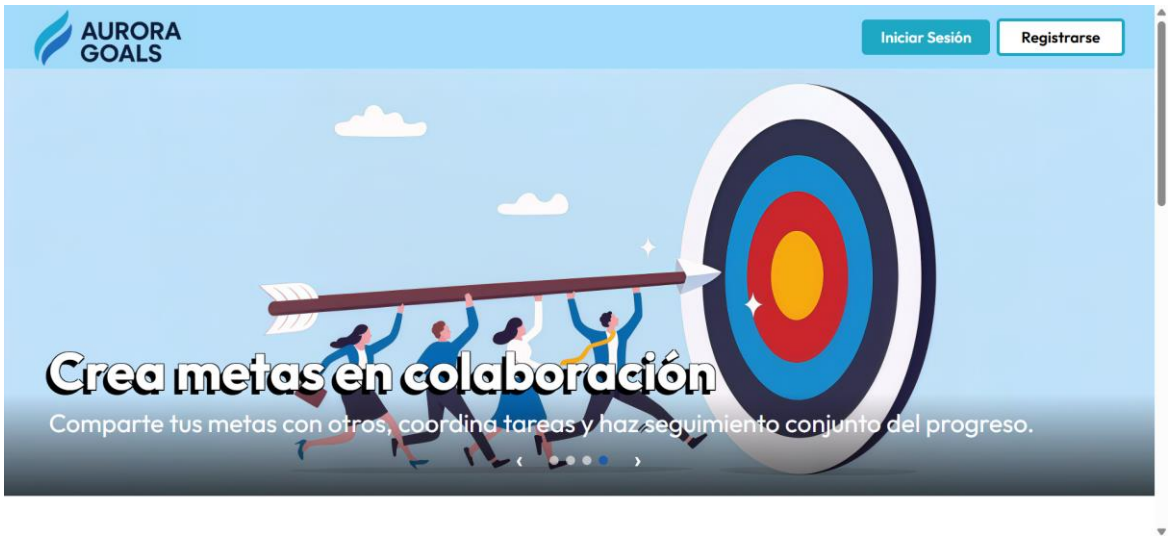

Figura 83: Cuarto banner informativo de la página principal pública.

En este banner se explica que la aplicación también permite trabajar en metas compartidas con otros usuarios. Se refuerza así la idea de colaboración y trabajo en equipo y se completa la explicación de todas las funcionalidades de la aplicación.

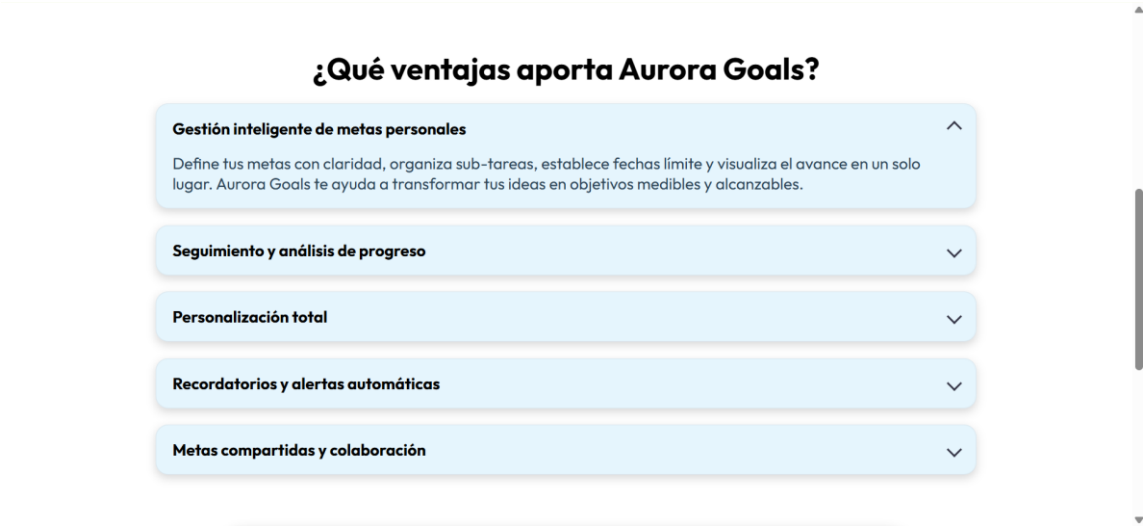

Figura 84: Sección de ventajas de Aurora Goals en la página principal pública.

Bajando por la página después de los banner, se muestran varios bloques o tarjetas que resumen los beneficios principales de la herramienta. Sirven para convencer al usuario antes de crear una cuenta.

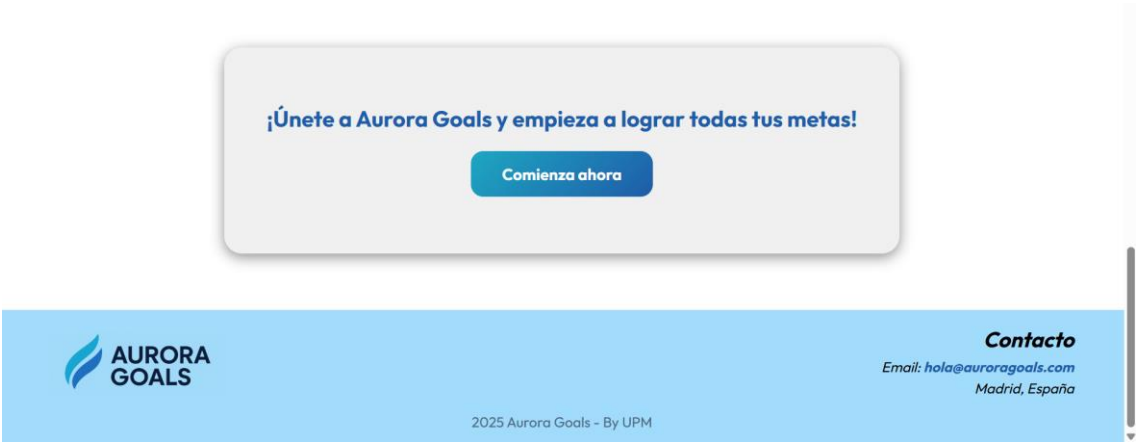

Figura 85: Llamada a registrarse al final página principal pública.

La pantalla presenta un mensaje final animando a comenzar a usar la aplicación, junto con un botón de registro “Comienza ahora”, que llevaría directamente a la página de registrarse. Actúa como cierre de la página pública.

Figura 86: Formulario de registro de usuario.

Se ve el formulario donde el usuario introduce correo electrónico, nombre y contraseña para crear una nueva cuenta.

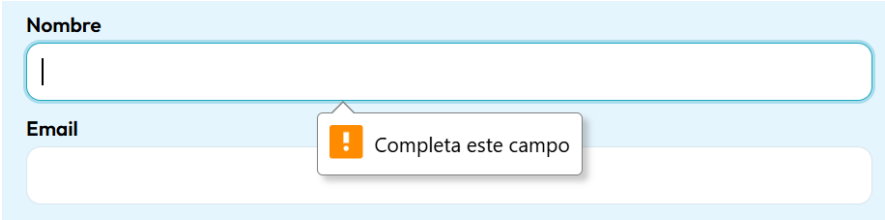

Figura 87: Validación del formulario de registro.

La captura muestra los mensajes del navegador cuando falta algún dato en el formulario. Permite comprobar que se impide enviar el registro con campos vacíos.

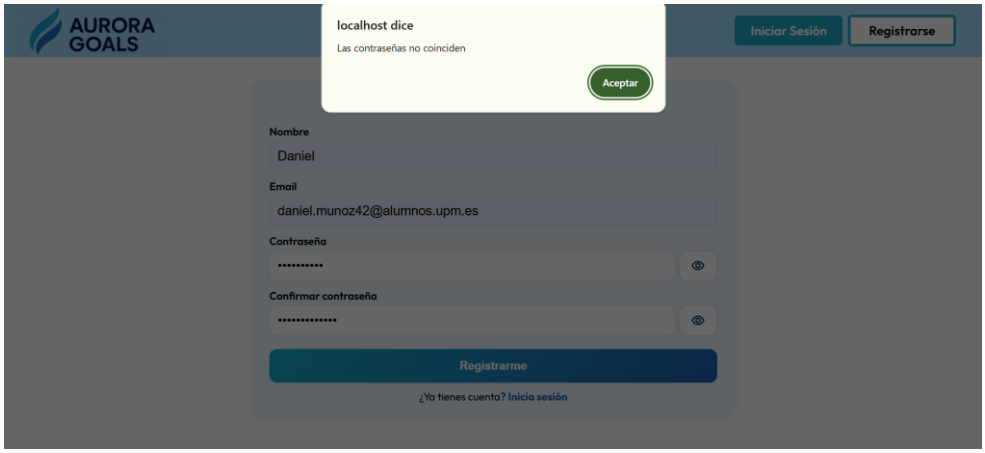

Figura 88: Mensaje de error al no coincidir la confirmación de la contraseña.

Aquí se muestra el mensaje del navegador al no coincidir las contraseñas propuestas por el usuario. Es una capa más de seguridad para confirmar que el usuario es consciente de qué contraseña ha puesto y que no se ha equivocado en ningún carácter.

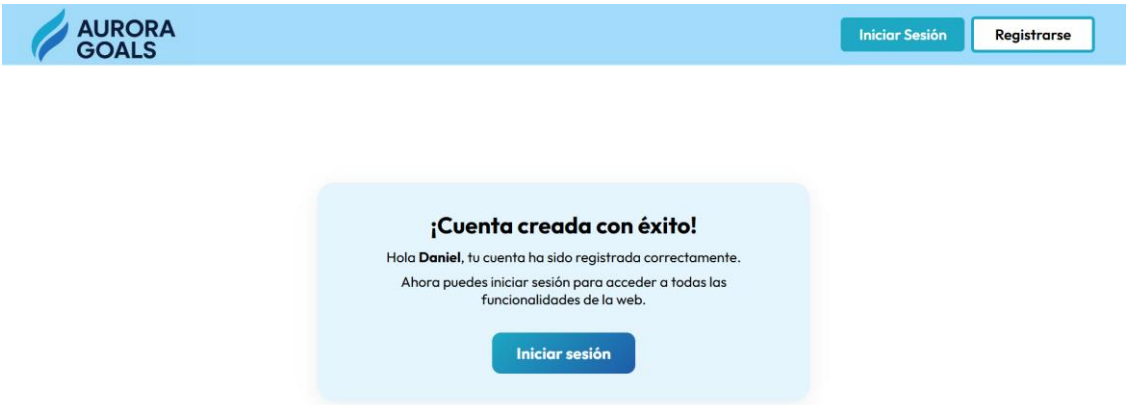

Figura 89: Pantalla de cuenta creada correctamente.

La aplicación informa de que el registro se ha completado con éxito y ofrece un enlace para ir a la pantalla de inicio de sesión. Cierra el flujo de alta de nuevos usuarios.

Figura 90: Formulario de inicio de sesión.

Se muestra la pantalla donde el usuario introduce su correo y contraseña para acceder al portal personal. El diseño es sencillo, con un botón para mostrar/ocultar la contraseña.

Figura 91: Mensaje de error en inicio de sesión.

La captura enseña el formulario cuando las credenciales no son válidas, acompañado de un mensaje de error en la parte central.

Figura 92: Portal de usuario: calendario y panel de resumen.

Una vez autenticado, el usuario ve un mensaje de bienvenida con su nombre, un calendario interactivo y un bloque con las próximas metas y tareas con sus respectivas fechas límite. Es la vista central del uso diario de la aplicación y donde puede ver que tiene que priorizar, ya que están ordenadas por fechas de expiración. Es importante comentar que cada meta y tarea, son botones que llevan directamente a la página de detalles de esa meta en particular.

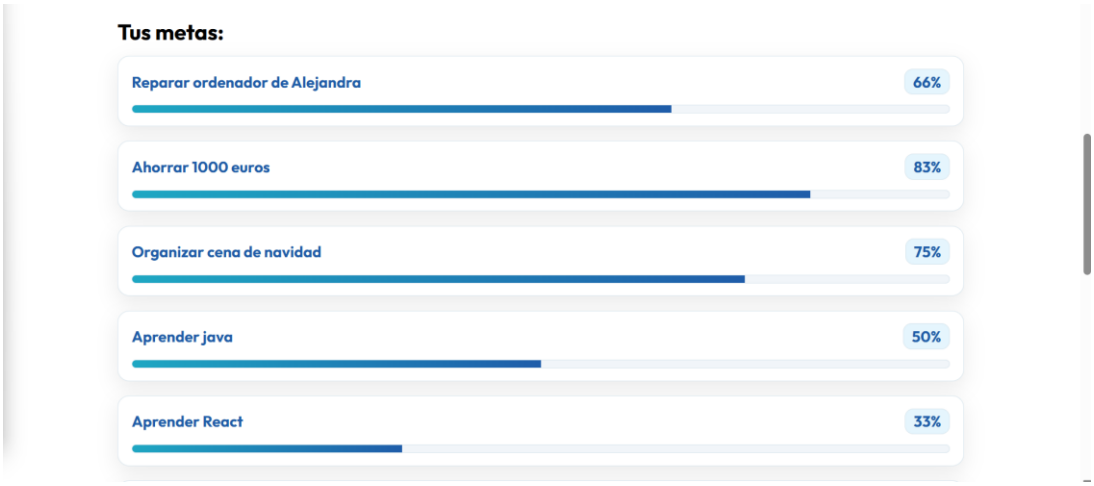

Figura 93: Listado de metas personales con progreso.

Se muestran las metas individuales del usuario como una lista de tarjetas con su porcentaje de avance. Cada elemento enlaza a la página de detalle de la meta.

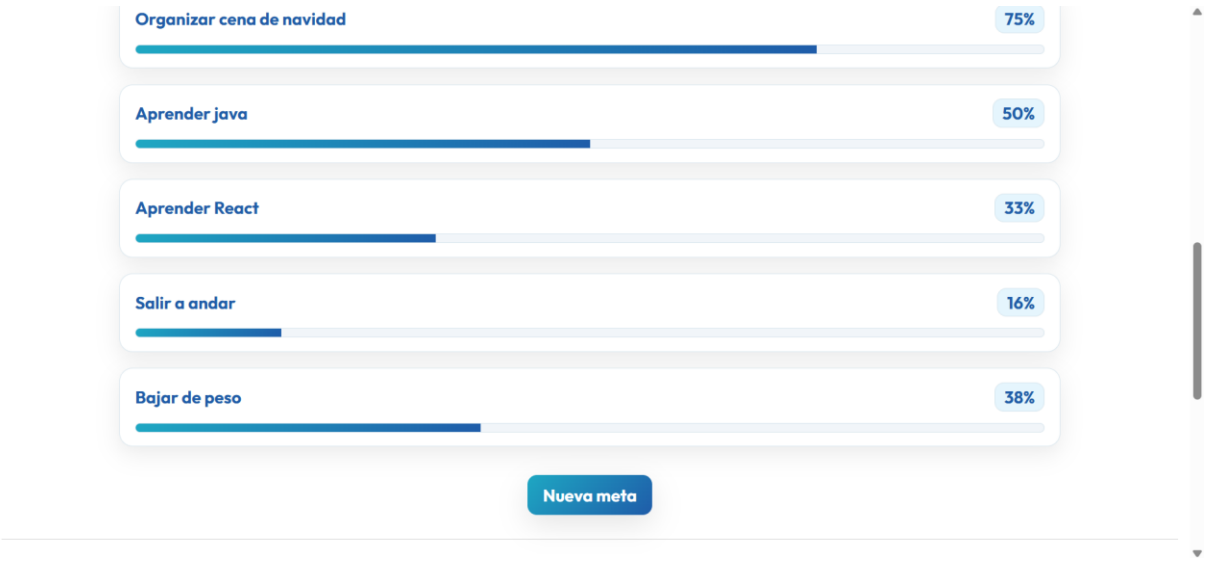

Figura 94: Continuación del listado y acción “Nueva meta”.

La captura incluye más metas personales y el botón para crear una nueva. Se aprecia la consistencia del diseño de tarjetas y barras de progreso.

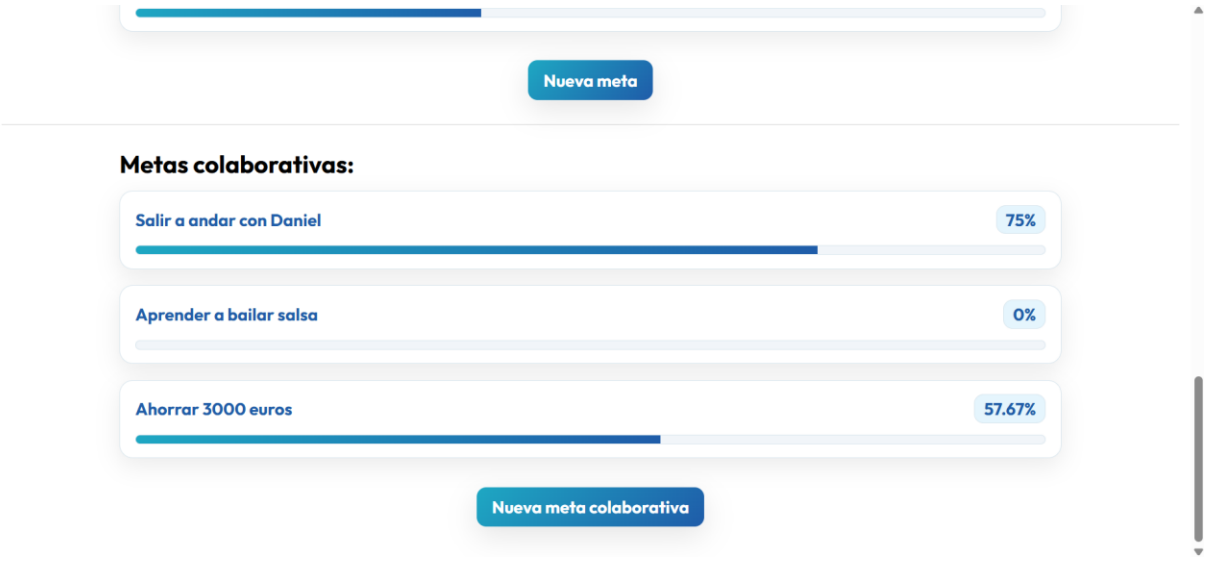

Figura 95: Listado de metas colaborativas.

Aquí se muestran las metas compartidas con otros usuarios, también con su porcentaje de avance. Se incluye el botón para crear una nueva meta colaborativa.

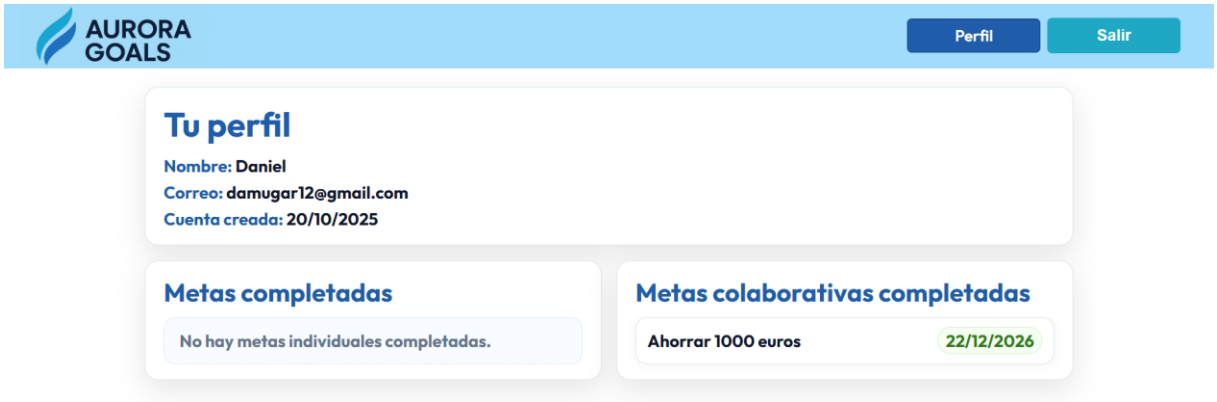

Figura 96: Página de perfil del usuario.

La vista recoge los datos básicos de la cuenta y un pequeño resumen de actividad con las metas completadas, tanto personales como colaborativas. Sirve como referencia personal del usuario dentro de la aplicación.

Figura 97: Detalle de meta personal basada en tareas.

Se presenta la ficha de una meta de tipo lista de tareas, con su título, descripción y un gráfico de progreso. Es el punto de entrada para gestionar las tareas asociadas.

Figura 98: Listado de tareas de una meta personal.

Se ven las tareas individuales con su descripción, estado y fecha límite. Desde esta pantalla el usuario puede marcar tareas como completadas o eliminarlas. Además, debajo de la pantalla se ven las acciones de “Editar” y “Eliminar”, acciones para la meta personal.

Figura 99: Formulario de edición de meta personal.

La captura muestra el formulario para modificar una meta ya existente: título, descripción, fecha y parámetros de configuración. Permite ajustar la meta sin tener que crear una nueva.

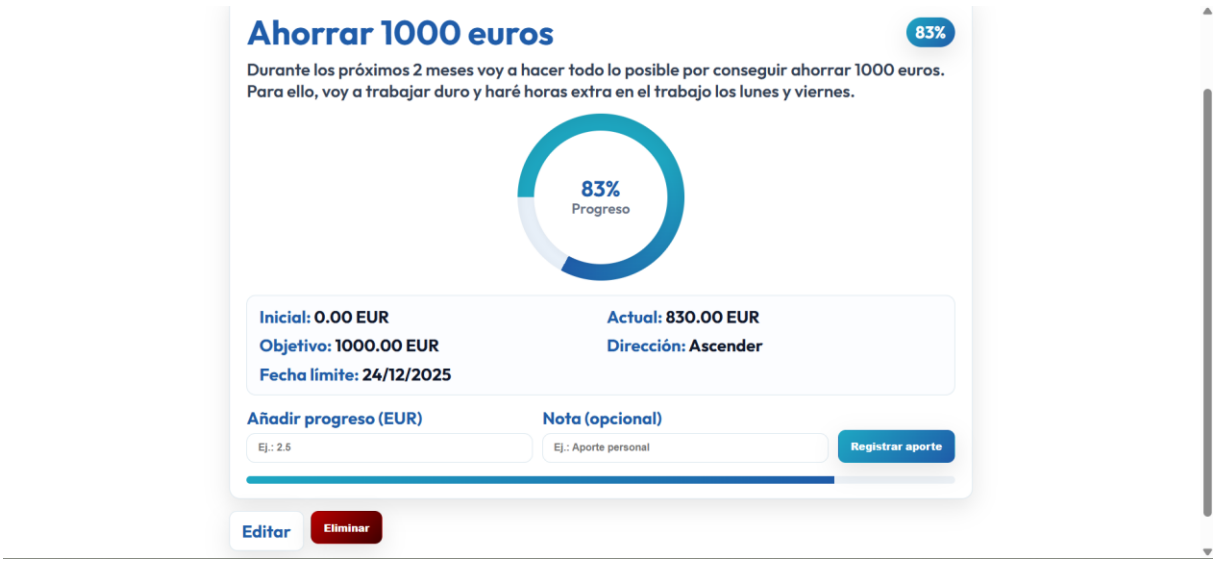

Figura 100: Meta personal numérica con indicadores de progreso.

Se representa una meta de tipo numérico incluyendo valor inicial, objetivo, valor actual y porcentaje de cumplimiento. El usuario puede registrar nuevas aportaciones desde esta misma vista.

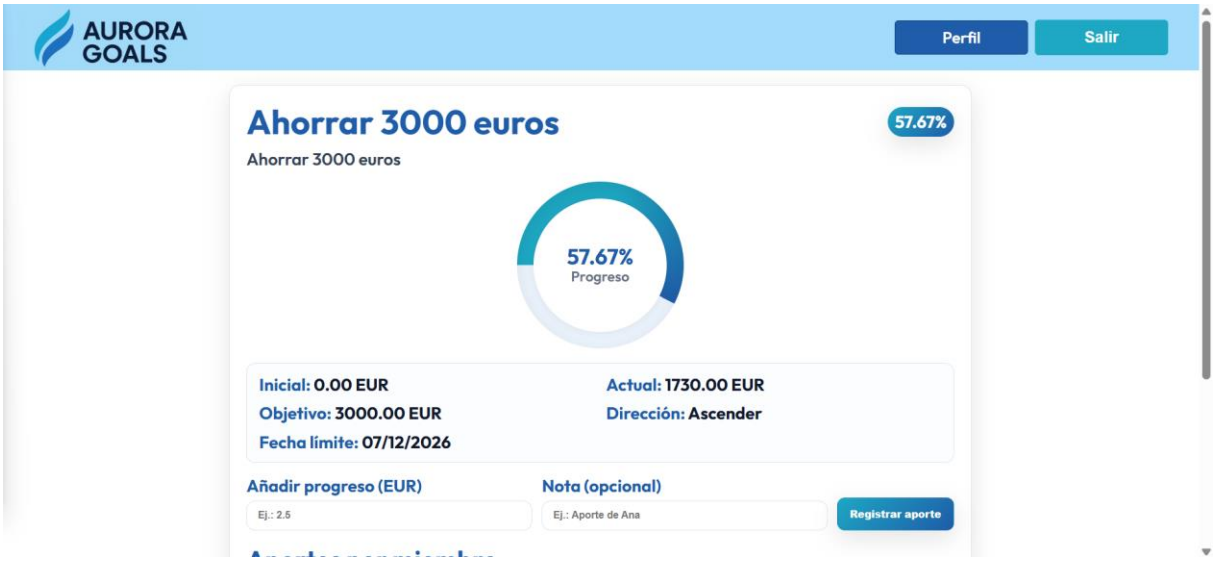

Figura 101: Meta colaborativa numérica con progreso global.

La pantalla es similar a la anterior, pero aplicada a una meta compartida. Se aprecia el progreso global del equipo y el contexto de colaboración.

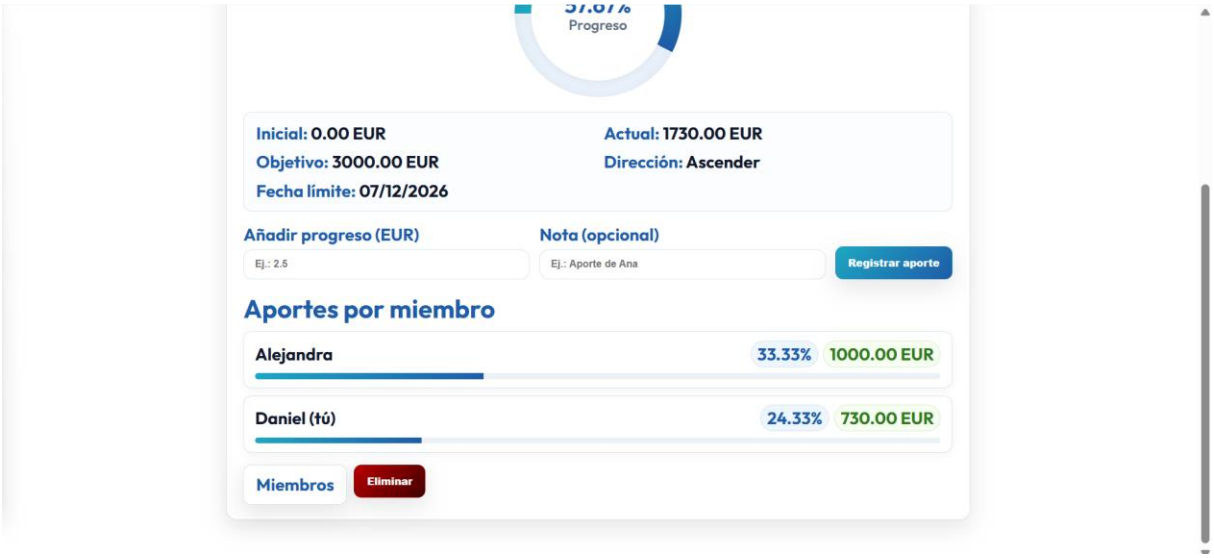

Figura 102: Ranking de aportes por miembro en meta colaborativa.

Se muestra la sección donde se indica qué porcentaje del avance total aporta cada participante, junto con la cantidad absoluta. Las barras horizontales permiten comparar fácilmente las contribuciones.

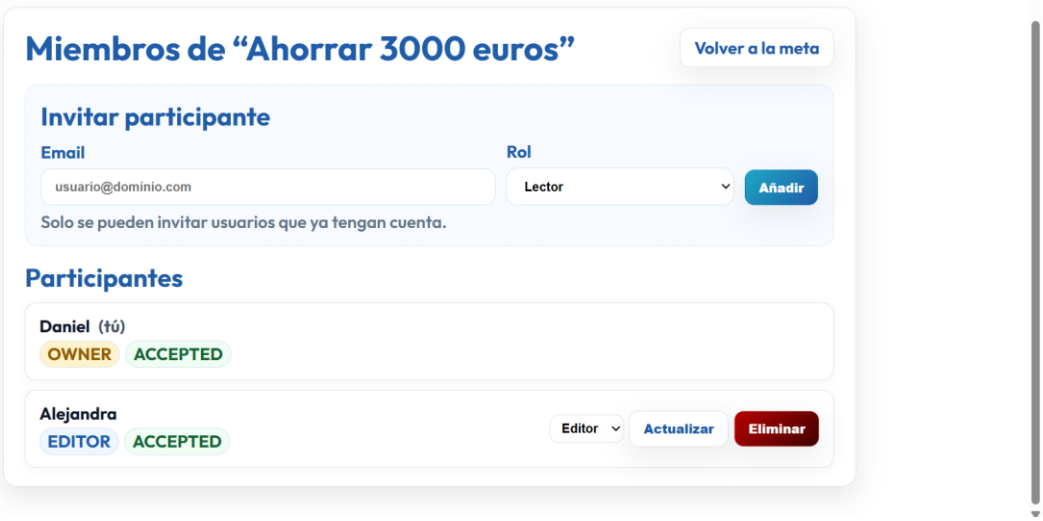

Figura 103: Listado de miembros de la meta colaborativa.

La captura enseña la lista de participantes, su rol dentro de la meta y su estado (invitado, aceptado, etc.). Desde aquí se controla quién forma parte del equipo y puede actualizar el rol que tiene o eliminarlo de la meta colaborativa.

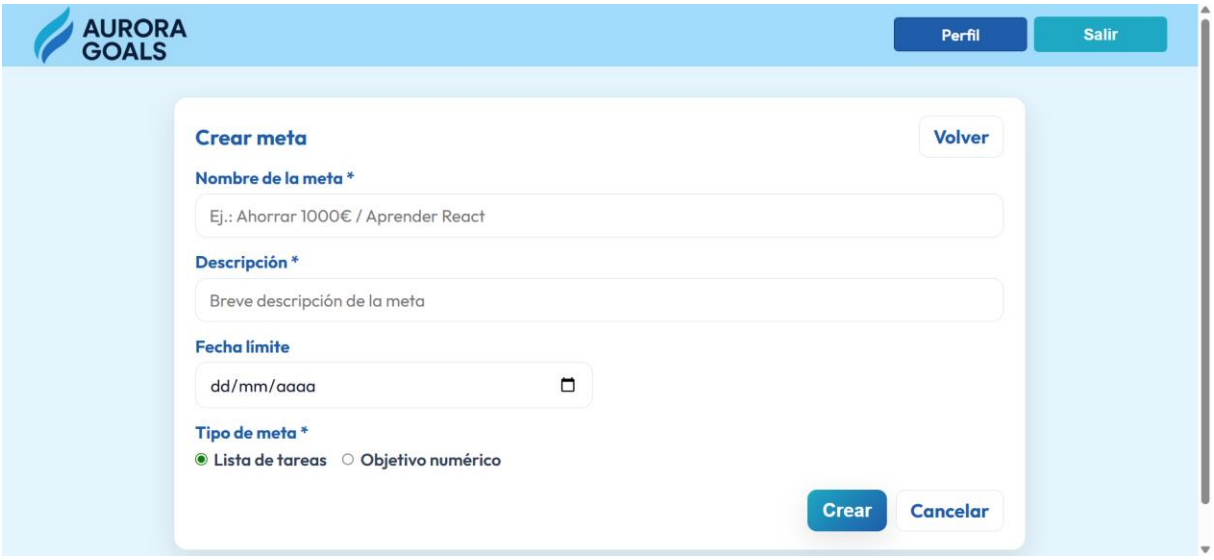

Figura 104: Formulario de creación de meta personal de tareas.

Se ve el formulario para definir una meta basada en lista de tareas: título, descripción, fecha límite y parámetros básicos. Es el punto de partida para metas del tipo tareas.

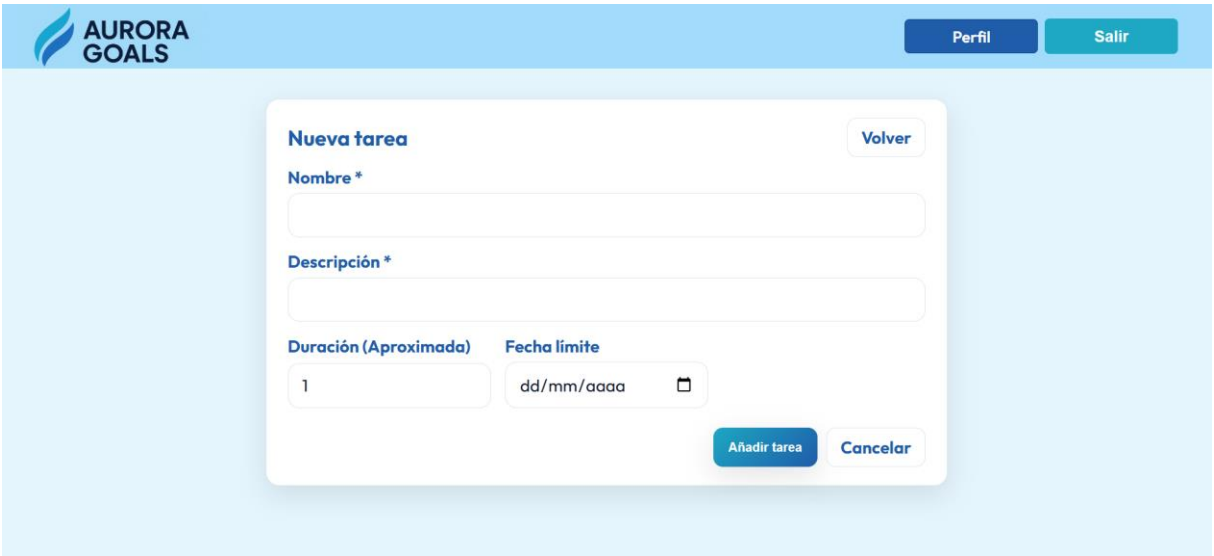

Figura 105: Formulario de creación de una nueva tarea en meta personal.

Aquí se ve el formulario para crear una nueva tarea dentro de una meta personal. El usuario debe de rellenar el nombre de la tarea, la descripción, la duración aproximada y la fecha límite de la tarea.

Figura 106: Formulario de creación de meta personal numérica.

En este caso el formulario permite introducir valor inicial, objetivo, unidad y dirección del progreso. Con esta configuración se crean metas orientadas a acumular o reducir una cantidad.

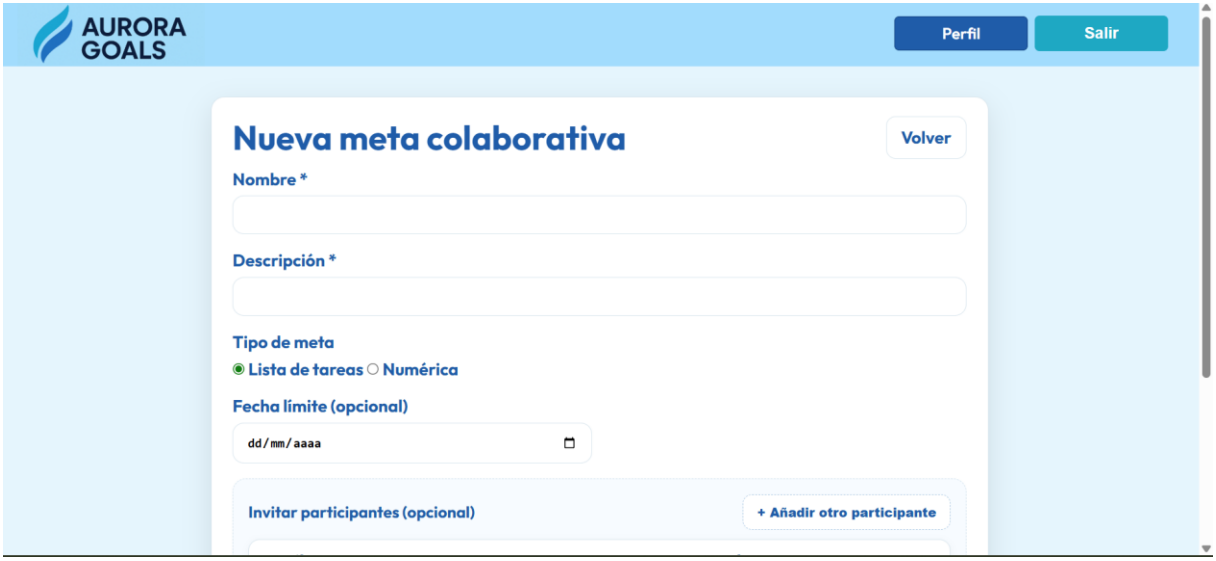

Figura 107: Formulario inicial de meta colaborativa.

La captura muestra la pantalla para crear una meta compartida, indicando nombre, descripción, tipo de meta y opciones generales. Marca el inicio del trabajo en equipo dentro de la aplicación.

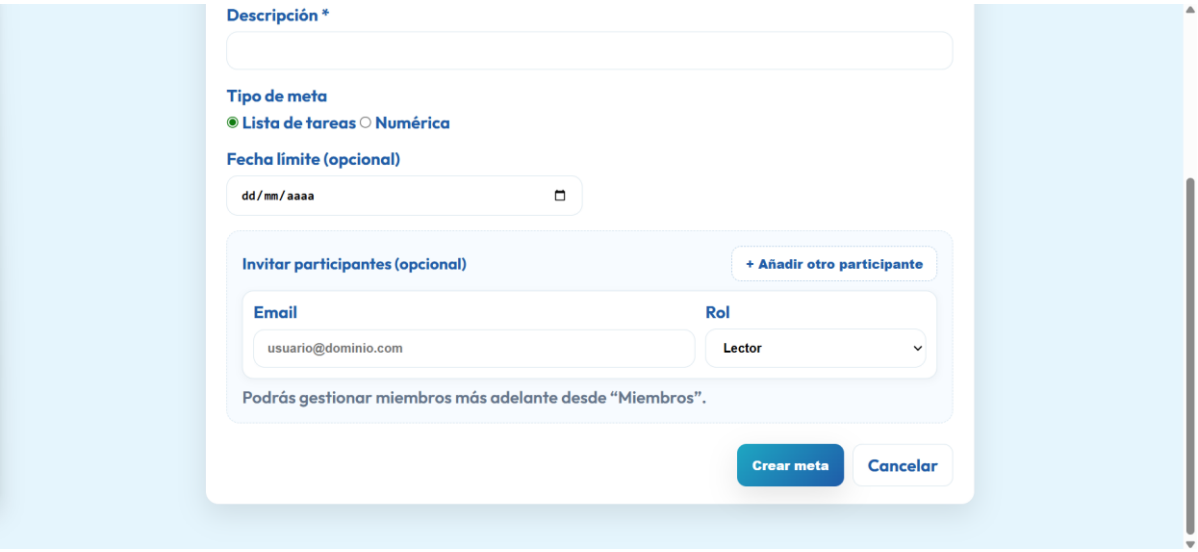

Figura 108: Selección de participantes en meta colaborativa.

Finalmente, se ilustra el formulario donde se añaden los participantes a la meta recién creada. El usuario define qué personas formarán parte del equipo y podrán registrar aportaciones.

Figura 109: Formulario de creación de una nueva tarea en meta colaborativa.

Como se puede apreciar, la única diferencia del formulario, es que en las tareas de las metas colaborativas, se va a pedir a qué usuario miembro del equipo se le va a asignar la tarea.

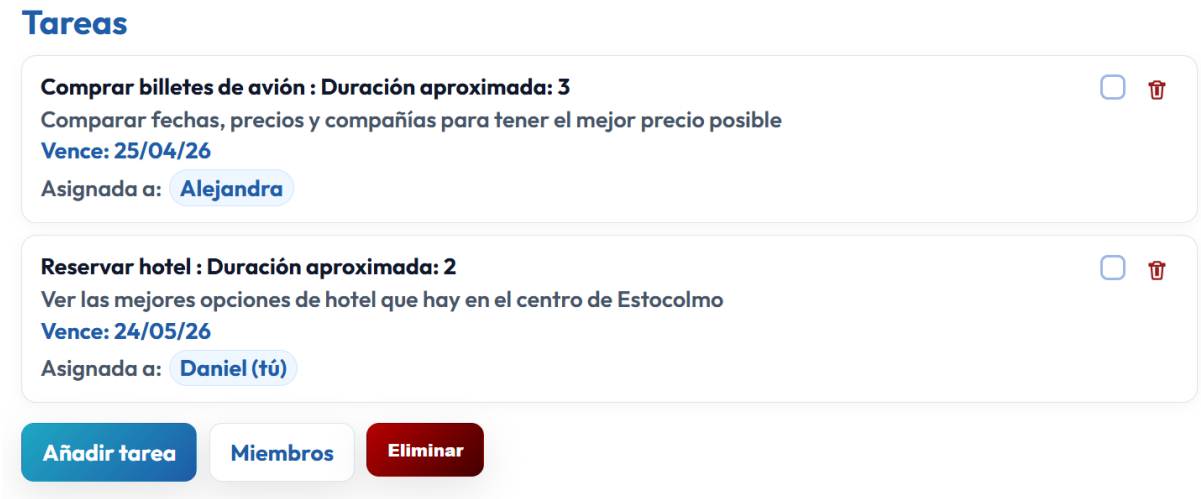

Figura 110: Lista de tareas con asignaciones de responsables.

Vemos como en una lista de tareas de una meta colaborativa se muestra la asignación de responsabilidades. Además de el titulo de la tarea, la duración aproximada, la descripción y el vencimiento.

6. Pruebas y validación

En este capítulo se describe el proceso de verificación de la aplicación Aurora Goals, con el objetivo de comprobar que las funcionalidades implementadas cumplen los requisitos definidos y que el usuario puede utilizar el sistema de forma coherente y sin errores críticos.

Dado el alcance del proyecto y el entorno de ejecución, se ha optado por realizar principalmente pruebas funcionales manuales, apoyadas con algunas comprobaciones informales de usabilidad con usuarios reales de mi entorno.

Además, se recogen las principales limitaciones detectadas durante el proceso.

6.1. Estrategia de pruebas

La estrategia de pruebas se ha centrado en validar todos los flujos de la aplicación, para que un usuario pueda gestionar sus metas personales y colaborativas y aprovechar al máximo todas las funcionalidades de la aplicación sin errores.

- Gestión de usuarios: registro, inicio de sesión, cierre de sesión.
- Gestión de metas personales.
- Gestión de metas colaborativas y participantes.
- Gestión de tareas asociadas a cada meta.
- Visualización del resumen en el portal de usuario.

Las pruebas se han organizado en casos de prueba que describen para cada escenario la acción a realizar, los datos de entrada y el resultado esperado. Estos casos se han ejecutado manualmente sobre la aplicación desplegada en entorno local, utilizando Chrome como principal navegador y comprobando siempre que no se producen errores en la interfaz ni mensajes de fallo en el servidor.

Aunque no se han desarrollado pruebas automatizadas, todos los casos de prueba se han repetido tras realizar cambios relevantes en el código, para asegurarse de que las modificaciones no introducían comportamientos inesperados.

6.2. Pruebas funcionales

Las pruebas funcionales se han organizado por módulos, para todos los escenarios de la aplicación. Todas las pruebas se han ejecutado manualmente sobre la aplicación en entorno local, comprobando que el comportamiento observado coincide con el esperado y que no se producen errores en el servidor.

6.2.1. Pruebas funcionales de autenticación

En el módulo de autenticación se han probado los siguientes casos:

- Registro correcto (PF-AUT-01).

Se introduce un correo electrónico no registrado, un nombre y una contraseña válidos. Tras enviar el formulario, el sistema crea el nuevo usuario en la tabla users y muestra un mensaje confirmando que la cuenta se ha creado correctamente.

- Registro con email duplicado (PF-AUT-02).

Se intenta registrar una cuenta utilizando un correo ya existente en la base de datos. El sistema detecta el conflicto, no inserta un nuevo usuario y devuelve un mensaje de error indicando que el email ya está registrado.

- Inicio de sesión correcto (PF-AUT-03).

Se introduce un correo registrado y su contraseña correspondiente. El sistema verifica el hash almacenado, genera un nuevo identificador de sesión y redirige al portal de usuario, donde ya aparecen las opciones de menú correspondientes a un usuario autenticado.

- Inicio de sesión con credenciales incorrectas (PF-AUT-04).

Se introduce un correo válido pero una contraseña incorrecta. El sistema no crea sesión y muestra un mensaje de “Credenciales no válidas”, manteniendo al usuario en la página de login.

- Cierre de sesión (PF-AUT-05).

Con una sesión activa, se pulsa el botón “Salir” de la cabecera. El sistema destruye la sesión, elimina las variables asociadas y redirige a la página principal pública, donde vuelven a mostrarse las opciones de Iniciar sesión y Registrarse.

6.2.2. Pruebas funcionales de metas personales

Para las metas individuales se han verificado diferentes operaciones:

- Creación de meta de lista de tareas (PF-MP-01).

Un usuario autenticado accede al formulario de nueva meta, introduce título, descripción, fecha de vencimiento y selecciona el tipo “lista de tareas”. Al guardar, la meta se inserta correctamente en la tabla *goals* con *goal_type = 'TASK_LIST'* y aparece en el listado de metas personales del portal.

- Creación de meta numérica (PF-MP-02).

Se repite el proceso anterior, pero seleccionando el tipo numérico y definiendo valor inicial, valor objetivo, unidad y dirección del progreso. La meta queda almacenada con *goal_type = 'NUMERIC'* y, al acceder a su página de detalle, se muestran correctamente los campos numéricos y el porcentaje de avance calculado.

- Actualización de progreso en meta numérica (PF-MP-03).

Desde la página de detalle de una meta numérica, se introduce una cantidad positiva de avance. El sistema toma el valor actual o el valor inicial, suma la cantidad (teniendo en cuenta la dirección UP/DOWN), limita el resultado a los rangos mínimo y máximo y actualiza el campo *current_value*. El porcentaje de progreso se actualiza en la interfaz según el nuevo valor.

- Gestión de tareas en meta de lista (PF-MP-04).

En una meta de tipo lista de tareas se crea una tarea introduciendo nombre, descripción y, opcionalmente, fecha de vencimiento. La tarea se guarda en la tabla *task* y aparece en el listado de la meta. Al marcarla como completada, su estado

cambia visualmente y, si se trata de una meta basada en tareas, el porcentaje de progreso de la meta también se incrementa.

- Eliminación de meta personal (PF-MP-05).

Se selecciona una meta personal existente y se confirma la acción de borrar. El sistema elimina la fila correspondiente en *goals* y, gracias a las restricciones ON DELETE CASCADE, también desaparecen automáticamente las tareas asociadas. La meta deja de aparecer en el portal y se muestra un mensaje confirmando la eliminación.

6.2.3. Pruebas funcionales de metas colaborativas

En el módulo colaborativo y en el portal de resumen se han ejecutado, entre otras, las siguientes pruebas:

- Creación de meta colaborativa numérica (PF-MC-01).

El usuario accede al formulario de nueva meta colaborativa, introduce título y descripción, selecciona tipo numérico y define rango de valores y unidad. Tras guardar, la meta se inserta en collab_goals con el usuario actual como propietario y aparece en el listado de metas colaborativas del portal.

- Invitación de participantes a meta colaborativa (PF-MC-02).

Desde la página de detalle de una meta colaborativa, el propietario añade otro usuario como participante. Se crea un registro en collab_goal_participants con el rol y estado configurados (por ejemplo, OWNER / EDITOR / VIEWER y PENDING o ACCEPTED), y el nuevo miembro pasa a figurar en el listado de participantes.

- Registro de aportaciones en meta colaborativa (PF-MC-03).

Un usuario participante en una meta colaborativa numérica introduce una aportación de progreso. El sistema registra la entrada en la tabla collab_goal_updates, actualiza el valor actual de la meta y recalcula el progreso global. Al volver a la página de detalle, la barra de progreso refleja el nuevo estado.

- Visualización del ranking de aportes (PF-MC-04).

En una meta colaborativa con aportaciones de varios usuarios, se accede a la sección “Aportes por miembro”. La aplicación muestra la lista de participantes con el porcentaje de contribución y la cantidad aportada, ordenados de mayor a menor. Se comprueba que los porcentajes suman aproximadamente el total del avance registrado.

- Resumen de próximas metas y tareas (PF-MC-05).

En el portal de usuario, con metas y tareas asignadas con fecha de vencimiento futura, se verifica que el bloque de “Próximas metas” muestra un máximo de seis metas ordenadas por fecha, y que “Próximas tareas” presenta las tareas pendientes ordenadas por su vencimiento. Cuando no hay elementos, se muestra un mensaje indicando que no existen metas o tareas próximas.

- Acceso al portal sin autenticación (PF-MC-06).

Por último, se comprueba el comportamiento de seguridad intentando acceder directamente a *user_portal.php* sin haber iniciado sesión. El sistema detecta la ausencia de *$_SESSION['uid'],* almacena un mensaje en la sesión y redirige automáticamente a la página de inicio de sesión, evitando el acceso no autorizado.

6.3. Pruebas de usabilidad

Además de las pruebas funcionales, se realizaron algunas pruebas de usabilidad informales con varios usuarios de mi entorno cercano que no habían participado en el desarrollo de la aplicación. El objetivo no era hacer un estudio exhaustivo, sino obtener impresiones reales sobre la claridad de la interfaz, la navegación y la motivación para usar la herramienta de forma continuada.

A estos usuarios se les pidió que completaran tareas sencillas: registrarse, iniciar sesión, crear una meta personal, añadir alguna tarea, consultar el panel de resumen y crear al menos una meta colaborativa. Mientras realizaban estas acciones, se observó cómo interactuaban con la aplicación y se recogieron sus comentarios.

De estas pruebas surgieron varias sugerencias concretas que se han incorporado al diseño final:

- Un usuario comentó que, nada más entrar al portal, le gustaría tener una "foto general" de su mes. A raíz de ese comentario se decidió incorporar un panel resumen más destacado, que combinara un calendario en la parte izquierda con los listados de próximas metas y tareas a la derecha. Esta idea resultó especialmente útil para orientar la pantalla principal hacia la planificación a corto plazo, y se terminó integrando como elemento central del portal de usuario.
- Otro usuario echó en falta un lugar donde ver "todo lo que ya había conseguido". A partir de esa sugerencia se añadió en la página de perfil una sección dedicada a metas completadas, donde se listan las metas personales que han alcanzado el 100% de progreso. De esta forma, el perfil no solo muestra datos estáticos del usuario, sino también un pequeño "historial de logros" que refuerza la motivación.

En general, la aplicación recibió un feedback muy positivo por parte de las personas que la probaron. Varios usuarios comentaron que se veían utilizando Aurora Goals en su día a día para organizar estudios, proyectos personales o hábitos, ya que les resultaba más clara y motivadora que soluciones genéricas como notas sueltas o hojas de cálculo.

Además, todos destacaron que la interfaz les parecía agradable a la vista, con una estética cuidada pero no recargada, y que la navegación era muy sencilla e intuitiva, incluso sin recibir explicaciones previas sobre cómo funcionaba la herramienta.

7. Conclusiones y trabajos futuros

En este último apartado de la memoria se presentan las conclusiones académicas, técnicas y personales que se han obtenido con el desarrollo de este proyecto. Además, se analiza el impacto ético, legal y medioambiental del mismo y, por último, se proponen algunas posibles mejoras futuras de la aplicación web.

7.1. Conclusiones

7.1.1. Conclusiones académicas

Desde el punto de vista académico, el proyecto ha permitido integrar de forma práctica contenidos que se habían trabajado de manera separada en distintas asignaturas como análisis de requisitos, diseño de bases de datos, programación web, ingeniería del software, gestión de proyectos, construcción y diseño de interfaces web, y un largo etc.

La definición de requisitos funcionales y no funcionales, su trazabilidad mediante problemas, objetivos y features, y la construcción de casos de uso han servido para aplicar en un contexto real los conceptos de modelado y especificación que se habían visto de forma teórica. Del mismo modo, el diseño del modelo entidad-relación y su transformación a un modelo lógico en MySQL han reforzado los conocimientos de modelado de datos y normalización.

En conjunto, el trabajo ha cumplido su función como ejercicio integrador, demostrando la capacidad para abordar un proyecto completo de desarrollo de software, desde el análisis inicial hasta la validación del sistema final.

7.1.2. Conclusiones técnicas

En el plano técnico, el principal resultado es una aplicación web funcional que permite gestionar metas personales y colaborativas con dos tipos de funcionamiento (metas numéricas y metas basadas en tareas). El sistema implementa:

- Una arquitectura cliente-servidor basada en PHP y MySQL, organizada en capas (presentación en pages, lógica en logic, estilos en style y scripts SQL independientes).
- Un modelo de datos que recoge usuarios, metas individuales, metas colaborativas, tareas y aportaciones, con sus relaciones y restricciones de integridad.
- Funcionalidades completas de gestión: registro e inicio de sesión, creación, edición y eliminación de metas, definición de tareas, registro de avances y visualización de progreso.
- Módulos específicos para la parte colaborativa, incluyendo gestión de participantes y cálculo del porcentaje de aportación de cada miembro.
- Un panel resumen con calendario, próximas metas y tareas que ayuda a priorizar el trabajo.

Además, se han incorporado medidas básicas de seguridad técnicas, como el hash de contraseñas, las consultas preparadas, la gestión de sesiones y el uso de tokens CSRF para proteger operaciones sensibles. Todo ello demuestra que es posible construir una solución coherente y relativamente completa utilizando tecnologías estándar tan importantes y conocidas como son PHP, MySQL, HTML, CSS y JavaScript.

7.1.3. Conclusiones personales

A nivel personal, este proyecto ha sido mucho más que un ejercicio académico, ha sido una prueba real de hasta dónde soy capaz de llegar por mí mismo.

Ver la aplicación funcionando, me produce mucha satisfacción y orgullo. Supone comprobar que, después de años de estudio, soy capaz de afrontar un trabajo grande y complejo, empezar desde cero y llevarlo hasta un resultado completo y coherente. Saber que este proyecto marca el final de una etapa y el inicio de mi vida profesional le da todavía más peso y significado.

Durante el desarrollo me he visto obligado a madurar en varios sentidos. He tenido que organizar mi tiempo, priorizar tareas, tomar decisiones de diseño sin tener siempre una respuesta clara, aceptar errores propios y corregirlos. Esto me ha hecho mucho más consciente de las responsabilidades que implica sacar adelante un proyecto, ya que nadie iba a hacerlo por mí, y el resultado dependía directamente del esfuerzo, la constancia y la capacidad que yo pusiera.

Sobre todo, me quedo con la sensación de haber demostrado que soy capaz de hacerlo. Trabajo que ha requerido investigar, probar, equivocarme y volver a intentarlo. Esa experiencia de ir resolviendo problemas uno a uno, hasta ver el sistema completo funcionando, me da mucha confianza de cara al futuro y me hace sentir preparado para afrontar nuevos retos profesionales con una actitud mucho más segura y madura.

7.2. Aspectos éticos, legales y medioambientales

El desarrollo de esta página web ha considerado las implicaciones éticas, legales y medioambientales garantizando una gestión de la información personal responsable y promoviendo el bienestar de los usuarios.

7.2.1. Aspectos éticos

El sistema maneja información relacionada con la organización personal de los usuarios (metas, tareas, hábitos, colaboraciones), por lo que resulta fundamental tratar estos datos de forma responsable.

En primer lugar, se ha seguido el principio de minimización de datos. La aplicación solo solicita la información necesaria para su funcionamiento (correo electrónico, nombre y contraseña, además de los datos de las propias metas), evitando recopilar información innecesaria o sensible. Las contraseñas nunca se almacenan en texto plano, sino mediante un hash, reduciendo el riesgo en caso de acceso no autorizado a la base de datos.

Desde el punto de vista del uso de la herramienta, el diseño favorece la colaboración y no incluye elementos que puedan fomentar prácticas de competición tóxica o exposición innecesaria, ya que las metas colaborativas están pensadas para entornos de confianza y el control de participantes recae en el propietario de la meta. En un despliegue real, sería importante acompañar la aplicación de políticas claras de uso responsable.

7.2.2. Aspectos legales

Aunque la aplicación se ha desarrollado y probado en un entorno local, en un escenario de despliegue real debería cumplir la normativa de protección de datos aplicable (por ejemplo, el Reglamento General de Protección de Datos RGPD y la legislación nacional correspondiente).

La arquitectura actual ya incorpora algunas buenas prácticas alineadas con estas normativas: uso de contraseñas cifradas, separación de credenciales de base de datos, y posibilidad de eliminar metas y datos asociados. No obstante, para un uso en producción sería necesario incorporar:

- Textos legales visibles (política de privacidad y condiciones de uso).
- Mecanismos de consentimiento para el tratamiento de datos.
- Posibilidad de ejercer derechos de acceso, rectificación y eliminación de datos personales.

Además, se han utilizado herramientas y recursos (por ejemplo, fuentes de Google Fonts y librerías estándar de PHP) que se distribuyen bajo licencias compatibles con el uso en proyectos académicos.

7.2.3. Aspectos medioambientales

El impacto medioambiental directo del proyecto es reducido, ya que se ha desarrollado y ejecutado en un entorno local. Sin embargo, en un despliegue real la aplicación estaría alojada en uno o varios servidores que consumirían recursos energéticos.

En este sentido, el diseño del sistema contribuye a un uso razonablemente eficiente de los recursos, ya que las consultas a la base de datos están indexadas en los campos más utilizados. Así se evita el procesamiento innecesario en el servidor y la interfaz no carga recursos excesivamente pesados. En evoluciones futuras podría valorarse el uso de servicios de alojamiento que apuesten por energías renovables y la optimización adicional de recursos para reducir el consumo energético asociado.

7.2.4. Contribución a los Objetivos de Desarrollo Sostenible (ODS)

La herramienta contribuye de forma directa con varios Objetivos de Desarrollo Sostenible propuestos por la agenda 2030 de la Organización de las Naciones Unidas (ONU):

- ODS 3. Salud y bienestar.

Una mejor planificación de metas personales (estudios, deporte, hábitos saludables) y una sensación clara de progreso pueden contribuir a reducir el estrés y mejorar el bienestar individual.

Figura 111: Salud y Bienestar ODS 3.

- ODS 4. Educación de calidad.

La aplicación puede utilizarse en contextos educativos (por ejemplo, estudiantes que organizan trabajos, proyectos o estudios a largo plazo), apoyando la gestión del tiempo y la autonomía del aprendizaje.

Figura 112: Educación de Calidad ODS 4.

- ODS 8. Trabajo decente y crecimiento económico.

La gestión de metas y tareas orientadas a proyectos profesionales o emprendimientos personales puede ayudar a mejorar la productividad y la organización del trabajo.

Figura 113: Trabajo decente y crecimiento económico ODS 8.

- ODS 9. Industria, innovación e infraestructura.

El desarrollo de soluciones digitales que optimizan la planificación y la colaboración se enmarca en la innovación tecnológica que impulsa este objetivo.

Figura 114: Industria, Innovación e infraestructura ODS 9.

Si bien la herramienta no está directamente orientada a políticas públicas o a grandes organizaciones, sí puede considerarse una pequeña contribución al fomento de hábitos de planificación y colaboración que apoyan estos objetivos.

7.3. líneas futuras

A partir del estado actual de Aurora Goals, se identifican varias líneas de trabajo futuro que permitirían enriquecer la funcionalidad y acercar la aplicación a un producto desplegable en producción:

- Recordatorios y notificaciones.

Implementar un sistema de recordatorios automáticos (por correo electrónico o notificaciones web/móviles) que avisen de metas o tareas próximas a vencer y de periodos prolongados sin actividad.

- Integración con calendarios externos.

Sincronizar las fechas límite de metas y tareas con calendarios como Google Calendar, permitiendo que el usuario vea sus objetivos dentro de su agenda habitual.

- Estadísticas y visualizaciones avanzadas.

Añadir gráficos de evolución del progreso, comparativas entre metas o resúmenes por semanas/meses, reforzando el componente analítico y motivador de la aplicación.

- Despliegue en un servidor de producción.

Adaptar la aplicación para su ejecución en un servidor real (configuración de Apache/Nginx, HTTPS, copias de seguridad, logs y monitorización), revisando además todas las medidas de seguridad y cumplimiento legal.

- Aplicación móvil complementaria.

Desarrollar una aplicación móvil que consuma la misma base de datos mediante una API, centrada en registrar avances y consultar metas de forma rápida desde el teléfono.

- Mejoras de usabilidad y accesibilidad.

Introducir filtros avanzados de metas, opciones de personalización del panel de inicio y mejoras de accesibilidad (contrastes, navegación por teclado, etiquetas ARIA) que hagan la herramienta más inclusiva.

Estas líneas muestran que el trabajo realizado no supone un punto final, sino una base sólida sobre la que seguir experimentando y evolucionando la aplicación en futuras iteraciones.

Referencias

[1] Diagrama Entidad Relación: tipos, características y ventajas. https://universidadeuropea.com/blog/modelo-entidad-relacion/

[2] Modelamiento de datos. https://bookdown.org/paranedagarcia/database/modelamiento-de-datos.html

[3] Draw.io: Herramienta de diagramas online. https://app.diagrams.net/

[4] Muncharaz García, José Manuel (2025). Aplicación web para la reserva de laboratorios de la ETSISI. Proyecto Fin de Carrera / Trabajo Fin de Grado, E.T.S.I de Sistemas Informáticos (UPM). Archivo Digital UPM. https://oa.upm.es/87753/

[5] Documentación oficial de XAMPP. https://www.apachefriends.org/docs/

[6] Repositorio GitHub del proyecto. https://github.com/danielmgrc/tfg-metas-web

[7] Google Fonts. "Outfit – Google Fonts". https://fonts.google.com/specimen/Outfit

[8] Apache Friends. "XAMPP – Installers and Downloads". https://www.apachefriends.org/

[9] Microsoft. “Visual Studio Code Documentation”. code.visualstudio.com. https://code.visualstudio.com/docs

[10] Git-scm.com. “Git – Documentation & Pro Git book”. https://git-scm.com/docs

[11] GitHub. “GitHub Docs”. docs.github.com. https://docs.github.com/es

[12] “Waterfall Model – Software Engineering”. GeeksforGeeks. https://www.geeksforgeeks.org/software-engineering/waterfall-model/

[13] PHP Group. “PHP Manual” y documentación oficial. php.net. https://www.php.net/

[14] Página de ayuda SQL “SQL Tutorial”. W3Schools. https://www.w3schools.com/sql/default.asp

[15] Página de ayuda PHP “PHP Tutorial”. W3Schools. https://www.w3schools.com/php/default.asp

[16] Página de ayuda JavaScript “JavaScript Tutorial”. W3Schools. https://www.w3schools.com/js/default.asp

[17] Página de ayuda CSS “CSS Tutorial”. W3Schools. https://www.w3schools.com/css/default.asp

[18] Página de ayuda HTML “HTML Tutorial”. W3Schools. https://www.w3schools.com/html/default.asp

[19] Página de ayuda HTML and CSS “HTML and CSS Tutorial”. W3Schools. https://www.w3schools.com/htmlcss/default.asp

[20] Nielsen, J. “10 Usability Heuristics for User Interface Design”. Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/

[21] Objetivos y metas de desarrollo sostenible – Desarrollo sostenible https://www.un.org/sustainabledevelopment/es/

[22] Objetivos de Desarrollo Sostenible - Naciones Unidas. https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[23] RGPD - Reglamento General de Protección de Datos. https://gdpr.eu/