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

Aplicación web para la gestión y seguimiento de objetivos personales

PROYECTO FIN DE GRADO

Diego Casero Ramón
Grado en Ingeniería del Software

Madrid, 2025

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

Grado en Ingeniería del Software

Aplicación web para la gestión y seguimiento de objetivos personales

PROYECTO FIN DE GRADO

Diego Casero Ramón
Grado en Ingeniería del Software

Bajo la dirección de:
Dr. Borja Bordel Sánchez

Madrid, 2025

Título: Aplicación web para la gestión y seguimiento de objetivos personales

Autor: Diego Casero Ramón

Grado en Ingeniería del Software

Dirección:

Dr. Borja Bordel Sánchez

Agradecimientos

En primer lugar, quiero expresar mi sincero agradecimiento a mi tutor, Borja Bordel Sánchez, por su confianza, dedicación y orientación durante el desarrollo de este trabajo. Su apoyo ha sido fundamental para poder llevar este proyecto a buen término.

También quiero dar las gracias a mis compañeros y amigos de la carrera, en especial a Alejandro Caro, Daniel Muñoz, David Peñas, Gabriel Alexander Sánchez, Víctor Fernández, Mario Bravo y Johannes Robayna. Gracias por todos los momentos compartidos, por el trabajo en equipo y por haber sido parte esencial de esta etapa académica y personal.

Agradezco igualmente a mis amigos y a toda mi familia, quienes han estado a mi lado en los momentos más importantes y me han ayudado a mantenerme motivado a lo largo del camino.

Gracias de corazón a mi novia, Karina Erika Cimpean, por su apoyo incondicional, especialmente en los momentos más difíciles de la carrera.

Y, sobre todo, gracias a mis padres, Gustavo Casero y Mónica Ramón. Ellos me han apoyado desde el primer día, han creído en mí incluso cuando yo dudaba, y me han dado las oportunidades y la fuerza necesarias para llegar hasta aquí. Este logro también es suyo.

Abstract

This project aims to develop a web application to address the issue of disorganized personal goal management. Today, many people use fragmented methods to track their goals, from handwritten lists to multiple apps that do not connect with each other. This information spread makes it hard to follow progress effectively and reduces long-term motivation.

The application offers a complete solution using **React.js** as the main frontend framework, taking advantage of its ability to create interactive interfaces and reusable components. State management is handled with custom hooks that improve performance and keep data consistent. For the backend, **Firebase** is used as the cloud platform, offering secure authentication, a real-time database with Firestore, and automatic hosting. **Firebase Functions** were also added to automate tasks like sending reminder emails. This serverless architecture removes the need for managing infrastructure and allows automatic scalability.

The app has two types of users: **regular users** and **administrators**. Regular users can create personal goals, set deadlines, define milestones, and see their progress through dynamic charts. The system also includes editing and deleting goals, and a personalized dashboard that shows important statistics about the user's performance. Administrators have access to a control panel where they can monitor global platform activity and manage registered users.

One key feature is the **automated notification system**, which sends emails reminders as deadlines approach. Scheduled functions check goals daily and send specific messages depending on how many days are left. A **secure authentication system** was also implemented, requiring email verification. The app includes real-time form validation and loading state handling to improve user experience. The responsive design adapts to different devices, ensuring access from both desktop and mobile.

Future features may include expanding the notification system with more communication channels, integration with external calendar APIs, and advanced

metrics for deeper analysis of user behaviour. There are also plans to add gamification elements and a native mobile version.

In conclusion, this project shows skills in modern full-stack development, cloud service integration, task automation, and user-centered design. The use of current technologies and scalable architecture makes this solution a viable tool for managing personal goals, with automated features that improve the user experience.

Resumen

Este proyecto tiene como objetivo desarrollar una aplicación web para abordar la problemática de la gestión desorganizada de objetivos personales. En la actualidad, muchas personas utilizan métodos fragmentados para el seguimiento de sus metas: desde listas escritas a mano hasta múltiples aplicaciones que no se comunican entre sí. Esta dispersión de información dificulta el seguimiento efectivo del progreso y, por tanto, reduce la motivación a largo plazo.

El desarrollo implementa una solución integral utilizando **React.js** como framework principal para el frontend, aprovechando su capacidad para crear interfaces interactivas y componentes reutilizables. La gestión del estado se realiza mediante hooks personalizados que optimizan el rendimiento y mantienen la coherencia de los datos. Para el backend, se ha elegido **Firebase** como plataforma de servicios en la nube, que proporciona autenticación segura, base de datos en tiempo real con Firestore y hosting automático. Adicionalmente, se han implementado **Firebase Functions** para automatizar tareas como el envío de recordatorios vía email. Esta arquitectura serverless elimina la necesidad de gestionar infraestructura propia, garantizando escalabilidad automática.

La aplicación distingue entre dos tipos de usuarios: **usuarios estándar** y **administradores**. Los usuarios pueden crear objetivos personalizados, establecer fechas límite, definir hitos intermedios y visualizar el progreso mediante gráficos dinámicos. El sistema incluye funcionalidades de edición y eliminación de objetivos, así como un dashboard personalizado que presenta estadísticas relevantes sobre el rendimiento del usuario. Los administradores acceden a un panel de control especializado que permite supervisar la actividad global de la plataforma y gestionar usuarios registrados.

Un aspecto destacado del desarrollo ha sido la implementación de un **sistema de notificaciones automatizadas** que envía recordatorios por email cuando se aproximan las fechas límite. Las funciones programadas ejecutan verificaciones diarias y envían notificaciones específicas según cuántos días quedan para la fecha límite. También se implementó un **sistema de autenticación robusto** que exige verificación de email como medida de seguridad. La aplicación incorpora validación

en tiempo real de formularios y manejo de estados de carga para mejorar la experiencia del usuario. La interfaz responsive se adapta a diferentes dispositivos, garantizando accesibilidad tanto en equipos de escritorio como en dispositivos móviles.

Entre las **funcionalidades futuras** se considera la expansión del sistema de notificaciones con diferentes canales de comunicación, integración con APIs de calendario externas y un sistema de métricas avanzadas que permita análisis más profundos del comportamiento del usuario. También se evalúa la posibilidad de incorporar elementos de gamificación y la creación de una versión móvil nativa.

En conclusión, este proyecto demuestra competencias en desarrollo full-stack moderno, integración de servicios en la nube, programación de tareas automatizadas y diseño centrado en el usuario. La elección de tecnologías actuales y la arquitectura escalable posicionan la solución como una herramienta viable para la gestión efectiva de objetivos personales con funcionalidades automatizadas que mejoran la experiencia del usuario.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Abreviaturas y Acrónimos

UPM	Universidad Politécnica de Madrid
API	Application Programming Interface
DOM	Document Object Model o Modelo de Objetos del Documento
SSL	Secure Socket Layer o Capa de Conexión Segura
NPM	Node Package Manager o Gestor de Paquetes de Node
ID	Identificador
SMTP	Simple Mail Transfer Protocol
CDN	Content Delivery Network o Red de Distribución de Contenido
RF	Requisito Funcional
RNF	Requisito No Funcional
CU	Caso de Uso
CRUD	Create, Read, Update and Delete
BaaS	Backend-as-a-Service
SDK	Software Development Kit o Kit de Desarrollo de Software
MOOC	Massive Online Open Courses o Cursos Online Masivos y Abiertos
RGPG	Reglamento General de Protección de Datos
ODS	Objetivos de Desarrollo Sostenible

1. Introducción

Este proyecto se ha desarrollado para crear una aplicación web destinada a la gestión de objetivos y el seguimiento de metas. El documento describe las diferentes etapas del desarrollo, desde la conceptualización inicial hasta la implementación final, incluyendo las posibles mejoras futuras que podrían incorporarse para ampliar la funcionalidad del sistema. Las principales características del proyecto son:

Interfaz centrada en el usuario: El sistema está diseñado para proporcionar una experiencia intuitiva que permite a los usuarios crear, gestionar y monitorear sus objetivos personales a través de una interfaz moderna y responsive.

Gestión integral de datos: La aplicación procesa y almacena información relacionada con usuarios, objetivos, hitos y fechas límite, garantizando la persistencia y seguridad de los datos mediante tecnologías de base de datos en tiempo real.

Sistema de notificaciones automatizadas: Implementa un mecanismo proactivo de recordatorios que envía alertas por correo electrónico basadas en la proximidad de fechas límite, mejorando la adherencia del usuario a sus objetivos.

Arquitectura moderna y escalable: El sistema integra un frontend desarrollado en React.js con servicios backend de Firebase, garantizando una arquitectura serverless que facilita el mantenimiento y la escalabilidad.

Funcionalidades diferenciadas por rol: Distingue entre usuarios estándar y administradores, proporcionando capacidades específicas para cada tipo de usuario según sus necesidades y permisos.

Para el desarrollo de este proyecto se adoptó un enfoque basado en tecnologías modernas de desarrollo web, priorizando la experiencia del usuario y la eficiencia en el procesamiento de datos. Se decidió usar React.js como framework frontend porque permite la creación de interfaces dinámicas y componentes reutilizables, mientras que Firebase proporciona una infraestructura robusta para el backend,

incluyendo autenticación segura, base de datos en tiempo real y funciones automatizadas.

La arquitectura del sistema se basa en principios de diseño centrado en el usuario, donde cada componente ha sido desarrollado considerando la usabilidad y la accesibilidad. El sistema de autenticación implementado requiere verificación de correo electrónico, garantizando la seguridad de las cuentas de usuario, mientras que la interfaz responsive asegura una experiencia consistente entre dispositivos.

El desarrollo del proyecto se estructuró siguiendo un modelo en cascada, permitiendo avanzar de manera secuencial a través de etapas definidas: análisis de requisitos, diseño, implementación, validación y mantenimiento. Esta metodología aseguró que cada fase se completara antes de iniciar la siguiente.

1.1. Objetivos del Proyecto

El objetivo principal de este proyecto es desarrollar una aplicación web que facilite la gestión personal de objetivos, proporcionando herramientas para la planificación, seguimiento y visualización del progreso hacia metas individuales. La aplicación busca centralizar la información relacionada con objetivos personales en una plataforma integrada y fácil de usar.

Objetivos específicos:

1.- Diseño y desarrollo de la aplicación web:

Crear un sistema intuitivo que facilite a los usuarios gestionar sus objetivos personales de manera autónoma, incluyendo la creación, edición, eliminación y seguimiento de metas individuales.

2.- Sistema de roles diferenciados:

Implementar dos niveles de acceso en la aplicación:

Usuarios estándar: pueden crear y gestionar objetivos personales, visualizar progreso mediante gráficos, establecer fechas límite y recibir notificaciones automatizadas.

Administradores: además de las funcionalidades estándar, pueden gestionar usuarios registrados y supervisar el uso general del sistema.

3.- Visualización de progreso y estadísticas:

Desarrollar componentes de visualización que permitan a los usuarios comprender su rendimiento a través de gráficos interactivos, métricas de progreso y estadísticas personalizadas sobre sus objetivos.

4.- Funcionalidades de notificación automatizada:

Implementar un sistema de recordatorios que incluya:

Notificaciones por proximidad: alertas automáticas cuando se aproximan las fechas límite (3 días, 1 día, mismo día).

Notificaciones por vencimiento: recordatorios cada 7 días para objetivos que han superado su fecha límite sin completarse.

Notificaciones de logro: alertas automáticas cuando un usuario completa exitosamente un objetivo.

Verificaciones programadas: funciones que se ejecutan a diario para monitorear el estado de los objetivos y determinar qué notificaciones enviar.

Comunicación por correo electrónico: sistema de notificaciones utilizando servicios de email para mantener a los usuarios informados sobre sus metas en todas las etapas del proceso.

5.- Autenticación y seguridad:

Desarrollar un sistema de autenticación robusto que incluya verificación obligatoria de correo electrónico, gestión segura de sesiones y protección de datos personales del usuario.

6.- Escalabilidad y mantenibilidad:

Diseñar una arquitectura modular que permita futuras expansiones del sistema, incluyendo la posibilidad de integrar nuevas funcionalidades como gamificación, integración con calendarios externos o desarrollo de aplicaciones móviles complementarias.

2. Marco Teórico

En este apartado se presentan los fundamentos teóricos y las tecnologías que sustentan el desarrollo del proyecto. Se comenzará analizando los conceptos relacionados con la gestión de objetivos personales y su importancia en el desarrollo personal, seguido de una descripción de las metodologías de desarrollo web modernas que se han empleado. Por último, se detallarán las tecnologías y herramientas seleccionadas para la implementación de la aplicación, justificando las decisiones técnicas adoptadas.

2.1. Gestión de Objetivos Personales

La gestión efectiva de objetivos personales es un aspecto fundamental en el desarrollo individual que ha cobrado importancia en la era digital. Los estudios en psicología cognitiva y del comportamiento han demostrado que la estructuración adecuada de metas mejora significativamente la motivación y, en consecución, la probabilidad de éxito.

Problemática Actual:

En la actualidad, la gestión de objetivos personales presenta varios desafíos importantes. La mayoría de las personas utilizan métodos fragmentados y desconectados para organizar sus metas: desde listas escritas a mano hasta aplicaciones móviles básicas que no proporcionan un seguimiento integral. Esta dispersión de la información genera varios problemas:

Falta de visibilidad del progreso: Sin un sistema centrado, es difícil evaluar el avance real hacia los objetivos.

Pérdida de motivación: La ausencia de recordatorios y seguimiento visual reduce el compromiso a largo plazo.

Desorganización temporal: La falta de integración de sistemas de planificación dificulta la asignación efectiva de tiempo.

Beneficios de la Digitalización:

La implementación de sistemas digitales para la gestión de objetivos ofrece ventajas significativas. La automatización de recordatorios, la visualización gráfica del progreso y la persistencia de datos permiten mantener un enfoque constante en las metas establecidas. Además, las estadísticas del dashboard permiten al usuario mantener una visión clara de su rendimiento y motivación constante hacia sus objetivos.

2.2. Metodologías de Desarrollo Web

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.

Desarrollo Orientado a Componentes

Se utilizó una arquitectura basada en componentes reutilizables, lo que permite mantener un código más estructurado facilitando las futuras expansiones del sistema. Esta metodología es especialmente efectiva en aplicaciones React, donde cada componente encapsula su propia lógica y presentación.

Desarrollo en Cascada

El proceso de desarrollo se basó en una metodología en cascada, con fases claramente definidas, incluyendo análisis de requisitos, diseño del sistema, implementación y validación. Este enfoque secuencial permitió una progresión ordenada, asegurando que cada fase se completara antes de avanzar a la siguiente.

Principios de Diseño Centrado en el Usuario

Se aplicaron los principios de usabilidad y accesibilidad, aprendidos en la asignatura Construcción y diseño de interfaces gráficas de usuario, en todas las etapas de desarrollo. La interfaz se diseñó considerando diferentes tipos de usuario y contextos de uso, asegurando una experiencia consistente en distintos dispositivos y navegadores.

2.3. Tecnologías y Herramientas Utilizadas

La selección de tecnologías se basó en criterios de modernidad, escalabilidad, seguridad y facilidad de mantenimiento. A continuación, se detallan las herramientas principales empleadas:

2.3.1. Frontend: React y Vite

Para el desarrollo del frontend se eligió **React.js** como framework principal debido a su arquitectura basada en componentes, que permite crear interfaces modulares y reutilizables. Esta característica es realmente valiosa en una aplicación de gestión de objetivos, donde elementos como formularios, gráficos y paneles de control deben ser consistentes y reutilizables en diferentes contextos.

React.js también ofrece el concepto de Virtual DOM (Modelo de Objetos del Documento), que optimiza el rendimiento de la aplicación mediante actualizaciones eficientes del DOM real, lo cual es crucial para mantener una experiencia fluida cuando se actualizan los datos en tiempo real. Esto evita tener que renderizar toda la página cuando hay una actualización, modificando solo los nodos necesarios del DOM real.

La gestión del estado de la aplicación se realizó utilizando tanto hooks nativos de React como hooks que se crearon en el proyecto. Un hook es una función especial que sirve para "engancharte" al estado y al ciclo de vida de un componente. Se utilizaron los hooks nativos useState para manejar el estado local de los componentes y useEffect para gestionar efectos secundarios como llamadas a la API y suscripciones a cambios. También se crearon hooks personalizados como useObjectives para encapsular la lógica específica de gestión de objetivos, incluyendo operaciones de creación, actualización, eliminación y sincronización con Firbase. Esto permite mantener el código organizado y facilitar las pruebas unitarias de los componentes individuales.

El sistema de rutas fue implementado utilizando React Router, que proporciona navegación del lado del cliente para crear una experiencia de usuario fluida. La configuración de rutas protegidas asegura que solo los usuarios autenticados puedan acceder al dashboard y que los administradores tengan acceso exclusivo al

panel de administración. Esta implementación se integra perfectamente con el sistema de autenticación de Firebase, verificando el estado de autenticación del usuario y redirigiendo automáticamente donde corresponda.

Figura 1: React

Figura 2: React Router

Como herramienta de construcción se seleccionó **Vite**, que ofrece ventajas significativas sobre otras herramientas más tradicionales como Create React App.

Vite proporciona hot module replacement instantáneo. Esto significa que cuando se realizan cambios en el código, estos se reflejan inmediatamente en el navegador sin tener que recargar la página completa, preservando el estado de la aplicación y acelerando significativamente el proceso de desarrollo y pruebas.

Además, genera builds optimizados para producción con configuración mínima, reduciendo la c4omplejidad del proyecto.

Figura 3: Vite

2.3.2. Backend: Firebase y Servicios en la Nube

Se eligió **Firebase** como plataforma backend debido a su arquitectura serverless, que elimina la necesidad de gestionar infraestructura propia y garantiza escalabilidad automática. Esta decisión se consideró apropiada para este Trabajo de Fin de Grado que se centra en la funcionalidad de la aplicación en lugar de en la administración de servidores.

Firebase Authentication proporciona un sistema de autenticación robusto que incluye verificación de email obligatoria, gestión automática de sesiones e integración nativa con React.js. La implementación de verificación de email añade una capa de seguridad que protege contra registros fraudulentos, asegurando la autenticidad de las cuentas de usuario.

Para el almacenamiento y la gestión de datos se utilizó **Firebase Firestore**, una base de datos NoSQL en tiempo real que permite una sincronización automática entre el frontend y el backend. Esta característica es ideal para una aplicación de gestión de objetivos, donde los usuarios necesitan ver actualizaciones inmediatas del progreso. Además, Firestore ofrece reglas de seguridad configurables que permiten establecer permisos granulares sobre los datos, asegurando que un usuario solo pueda acceder a su información personal.

Firesetore Functions desempeña un papel crucial en la automatización de tareas del sistema. Estas funciones serverless ejecutan código backend sin la necesidad

de gestionar servidores, lo que las hace ideales para implementar el sistema de notificaciones automatizadas. Por ejemplo, la función dailyCheck se ejecuta diariamente mediante un cron job, revisa todos los objetivos almacenados en la base de datos y envía notificaciones a los usuarios vía email según la proximidad de las fechas límite.

Se utiliza **Firebase Hosting** para el despliegue de la aplicación, que ofrece una red de distribución de contenido global para optimizar la velocidad de carga y certificados SSL automáticos para asegurar las comunicaciones.

Figura 4: Firebase

2.3.3. Herramientas de Desarrollo y Despligue

Para el control de versiones se implementó **Git** con **GitHub** como repositorio remoto, permitiendo un seguimiento detallado de los cambios realizados en el código y proporcionando un respaldo del proyecto. Esta configuración también facilita una posible colaboración futura en caso de que el proyecto se expanda con más desarrolladores.

Figura 5: Git y GitHub

Visual Studio Code se utilizó como entorno de desarrollo integrado, aprovechando extensiones específicas para React y JavaScript que proporcionan funcionalidades como el autocompletado inteligente y terminal para comandos de desarrollo.

Figura 6: Visual Studio Code

Para la funcionalidad de envío de correos electrónicos se integró **Nodemailer** usando servicios de Gmail, permitiendo configurar plantillas de correo personalizadas. Esta integración es totalmente necesaria para el sistema de notificaciones automatizadas que mantiene a los usuarios informados sobre el estado de sus objetivos.

Figura 7: Nodemailer

2.3.4. Gestión de Dependencias: Node.js y npm

Aunque el backend de la aplicación está completamente implementado con Firebase, el desarrollo del frontend requiere un entorno de **Node.js** para la gestión de dependencias y la construcción del proyecto. Node.js proporciona el runtime necesario para ejecutar Vite y otras herramientas de desarrollo, mientras que **npm** (Node Package Manager) actúa de gestor de paquetes para manejar todas las dependencias del proyecto.

npm desempeña un papel fundamental en la gestión del ecosistema de dependencias de React. A través del archivo package.json, se definen todas las librerías que se necesitan para el correcto funcionamiento de la aplicación. El directorio node_modules contiene todas estas dependencias que se descargan automáticamente, asegurando que el proyecto tenga acceso a todas las funcionalidades requeridas.

La utilización de npm también permite la automatización de tareas comunes de desarrollo mediante scripts definidos en package.json, como npm run dev para iniciar el servidor local de desarrollo, npm run build para construir la aplicación para producción, y npm run preview para previsualizar el build.

La gestión de versiones de dependencias se realiza a través de package-lock.json que garantiza que todos los entornos de desarrollo y producción utilicen exactamente las mismas versiones de las librerías, eliminando así problemas de compatibilidad.

Figura 8: Node.js y npm

En conclusión, esta combinación de tecnologías forma un ecosistema coherente que facilita el desarrollo, despliegue y mantenimiento de la aplicación, asegurando que el sistema sea escalable, seguro y mantenible a largo plazo, mientras proporciona una experiencia de usuario moderna y eficiente.

3. Análisis y Diseño del Sistema

En este apartado se presenta el análisis detallado de los requisitos del sistema y el diseño arquitectónico de la aplicación web para gestión de objetivos personales. El proceso de análisis y diseño es una fase fundamental del desarrollo, ya que es donde se establecen las bases técnicas y funcionales sobre las cuales se construirá la solución final.

Se comenzará con el análisis de requisitos funcionales y no funcionales que debe satisfacer el sistema, seguido del diseño de la arquitectura basada en servicios en la nube utilizando Firebase y React.js. Posteriormente, se detallarán los casos de uso principales, el diseño de la base de datos en Firestore y, por último, los aspectos clave del diseño de la interfaz de usuario.

Esta documentación sirve como guía técnica para la implementación del sistema y también como referencia para futuras expansiones o modificaciones de la aplicación.

Como se mencionó anteriormente, el desarrollo del proyecto siguió un modelo en cascada, donde cada fase del análisis y diseño se completó antes de proceder con la implementación, asegurando una base sólida y documentada.

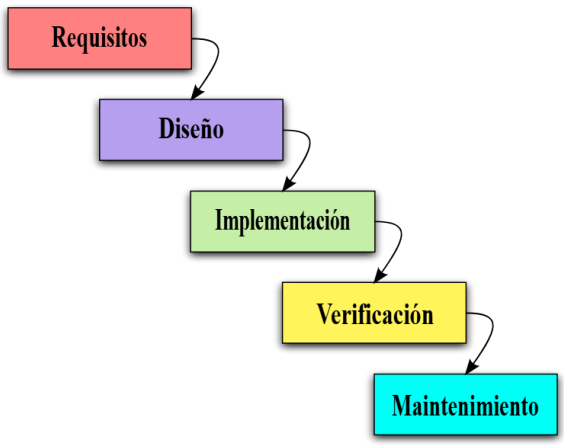

Figura 9: Modelo de desarrollo en cascada

3.1. Análisis de Requisitos

El análisis de requisitos constituye la base fundamental para el desarrollo del sistema, ya que define tanto las funcionalidades que debe de tener la aplicación como las características de calidad que debe cumplir. Los requisitos se han clasificado en funcionales y no funcionales para la posterior implementación.

3.1.1. Requisitos Funcionales

Los requisitos funcionales describen las funcionalidades específicas que debe proporcionar el sistema. Se han definido los requisitos organizados por módulos:

RF-01. Gestión de Autenticación

ID del requisito	Descripción
RF-01.1	El sistema debe permitir a los usuarios iniciar sesión con email y contraseña.
RF-01.2	El sistema debe permitir a los usuarios registrarse proporcionando email y contraseña.
RF-01.3	El sistema debe enviar un correo de verificación tras el registro.
RF-01.4	El sistema debe requerir verificación de email antes de permitir el acceso completo.
RF-01.5	El sistema debe permitir a los usuarios cerrar sesión de forma segura.

Tabla 1: RF-01. Gestión de Autenticación

RF-02. Gestión de Objetivos (Usuario)

ID del requisito	Descripción

RF-02.1	El sistema debe permitir a los usuarios crear nuevos objetivos personales.
RF-02.2	El sistema debe permitir a los usuarios editar objetivos existentes.
RF-02.3	El sistema debe permitir a los usuarios eliminar objetivos.
RF-02.4	El sistema debe permitir a los usuarios visualizar la lista de sus objetivos.
RF-02.5	El sistema debe permitir a los usuarios marcar objetivos como completados.
RF-02.6	El sistema debe permitir a los usuarios añadir comentarios a los objetivos.
RF-02.7	El sistema debe permitir a los usuarios añadir fechas límite a los objetivos.

Tabla 2: RF-02. Gestión de Objetivos (Usuario)

RF-03. Visualización y Progreso

ID del requisito	Descripción
RF-03.1	El sistema debe mostrar el progreso de objetivos de forma visual mediante gráficos.
RF-03.2	El sistema debe mostrar el porcentaje de tiempo transcurrido desde la creación del objetivo hasta la fecha límite.
RF-03.3	El sistema debe proporcionar un progreso general de todos los objetivos.
RF-03.4	El sistema debe mostrar gráficos estadísticos de los objetivos del usuario.
RF-03.5	El sistema debe permitir personalizar el dashboard del usuario.

Tabla 3: RF-03. Visualización y Progreso

RF-04. Sistema de Notificaciones

ID del requisito	Descripción
RF-04.1	El sistema debe enviar notificaciones 3 días antes de la fecha límite.
RF-04.2	El sistema debe enviar notificaciones 1 día antes de la fecha límite.
RF-04.3	El sistema debe enviar notificaciones el día de vencimiento.
RF-04.4	El sistema debe enviar notificaciones cada 7 días para objetivos vencidos.
RF-04.5	El sistema debe enviar notificaciones cuando se completa un objetivo.
RF-04.6	El sistema debe ejecutar verificaciones automáticas diarias.

Tabla 4: RF-04. Sistema de Notificaciones

RF-05. Funcionalidades de Administración

ID del requisito	Descripción
RF-05.1	El sistema debe permitir al administrador visualizar todos los usuarios registrados.
RF-05.2	El sistema debe permitir al administrador añadir objetivos a otros usuarios.
RF-05.3	El sistema debe permitir al administrador editar objetivos de otros usuarios.
RF-05.4	El sistema debe permitir al administrador eliminar objetivos de otros usuarios.
RF-05.5	El sistema debe permitir al administrador completar objetivos de otros usuarios.

Tabla 5: RF-05. Funcionalidades de Administración

3.1.2. Requisitos No Funcionales

Los requisitos no funcionales definen las características de calidad que debe cumplir el sistema para asegurar una experiencia de usuario satisfactoria y un funcionamiento eficiente.

RNF-01. Rendimiento

ID del requisito	Descripción
RNF-01.1	El tiempo de respuesta para operaciones básicas no debe exceder los 2 segundos.
RNF-01.2	El tiempo de carga inicial de la aplicación no debe superar los 3 segundos.
RNF-01.3	El sistema debe soportar actualizaciones en tiempo real sin degradación del rendimiento.
RNF-01.4	Las notificaciones automáticas deben procesarse en menos de 1 minuto.

Tabla 6: RNF-01. Rendimiento

RNF-02. Seguridad

ID del requisito	Descripción
RNF-02.1	Todas las comunicaciones deben utilizar protocolo HTTPS.
RNF-02.2	Las contraseñas deben almacenarse de forma segura utilizando hash.
RNF-02.3	El acceso a datos debe estar restringido según el rol del usuario.
RNF-02.4	El sistema debe implementar verificación obligatoria de email.

Tabla 7: RNF-02. Seguridad

RNF-03. Usabilidad

ID del requisito	Descripción
RNF-03.1	La interfaz debe ser responsive y adaptarse a dispositivos móviles y desktop.
RNF-03.2	La navegación debe ser intuitiva y consistente en toda la aplicación.
RNF-03.3	Los formularios deben proporcionar validación en tiempo real.
RNF-03.4	El sistema debe proporcionar mensajes de error claros y útiles.
RNF-03.5	La aplicación debe mantener el estado del usuario durante la sesión.

Tabla 8: RNF-03. Usabilidad

RNF-04. Disponibilidad

ID del requisito	Descripción
RNF-04.1	El sistema debe estar disponible 24/7 con un uptime mínimo del 99%.

Tabla 9: RNF-04. Disponibilidad

RNF-05. Escalabilidad

ID del requisito	Descripción
RNF-05.1	El sistema debe soportar un crecimiento en el número de usuarios sin modificaciones arquitecturales.
RNF-05.2	La base de datos debe escalar automáticamente según la demanda.
RNF-05.3	El sistema debe optimizar el uso de recursos de Firebase.

Tabla 10: RNF-05. Escalabilidad

RNF-06. Mantenibilidad

ID del requisito	Descripción
RNF-06.1	El código debe seguir estándares de programación y estar bien documentado.
RNF-06.2	La arquitectura debe permitir actualizaciones sin interrumpir el servicio.
RNF-06.3	El sistema debe generar logs para facilitar el debugging y monitoreo.

Tabla 11: RNF-06. Mantenibilidad

RNF-07. Compatibilidad

ID del requisito	Descripción
RNF-07.1	La aplicación debe funcionar en navegadores modernos (Chrome, Firefox, Safari, Edge).
RNF-07.2	El sistema debe ser compatible con diferentes tamaños de pantalla.

Tabla 12: RNF-07. Compatibilidad

3.2. Diseño de la Arquitectura

El diseño arquitectónico del sistema se basa en una arquitectura moderna de aplicaciones web que utiliza servicios en la nube para tener escalabilidad, seguridad y facilidad de mantenimiento. La arquitectura adoptada sigue el patrón **cliente-servidor** con un enfoque serverless eliminando la necesidad de gestionar infraestructura propia.

3.2.1. Capas de la Arquitectura

El sistema se divide en **tres capas** principales: la **capa de presentación** (frontend), la **capa de servicios** (backend) y la **capa de datos**. Esta división permite separar responsabilidades facilitando el mantenimiento y la escalabilidad del sistema.

Capa de Presentación (Frontend)

Está implementada utilizando React.js y se ejecuta completamente en el navegador del usuario. Su función principal es renderizar de forma dinámica la interfaz del usuario.

Además, esta capa es la encargada de gestionar el estado local de la aplicación usando React hooks. También maneja la interacción del usuario con formularios y elementos gráficos, facilitando una experiencia intuitiva.

La comunicación con los servicios de backend se realiza a través de la SDK de Firebase, garantizando una integración directa y segura con las funcionalidades del servidor. Esta capa también proporciona una experiencia responsive, adaptado a diferentes dispositivos y tamaños de pantalla, mejorando la accesibilidad y la usabilidad.

Los componentes principales son las páginas de autenticación (Login, Register), el dashboard principal (Dashboard), el panel de administración (Admin) y componentes reutilizables como gráficos y contenedores.

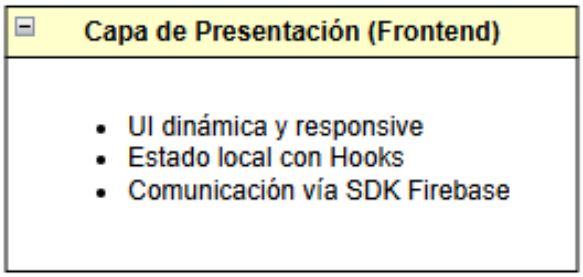

Figura 10: Capa de presentación

Capa de Servicios (Backend)

La capa de servicios está implementada utilizando Firebase, que proporciona una arquitectura serverless. Los servicios que se utilizan en nuestro sistema son:

Firebase Authentication: Gestiona toda la lógica de autenticación, incluyendo registro de usuarios, inicio de sesión, verificación de email y gestión de sesiones. Proporciona tokens de autenticación seguros que se utilizan para autorizar las operaciones en la base de datos.

Firebase Functions: Ejecuta código backend en respuesta a eventos específicos. En nuestro sistema, la función principal dailyCheck se ejecuta automáticamente cada día mediante un cron job programado, revisando todos los objetivos y enviando notificaciones por email según corresponda. Estas funciones también manejan la lógica de envío de correos utilizando Nodemailer con configuración SMTP (Simple Mail Transfer Protocol).

Firebase Hosting: Proporciona hosting para la aplicación React construida, con distribución global a través de CDN (Content Delivery Network), certificados SSL (Secure Sockets Layer) automáticos y configuración de reglas de redirección para el enrutamiento del lado del cliente.

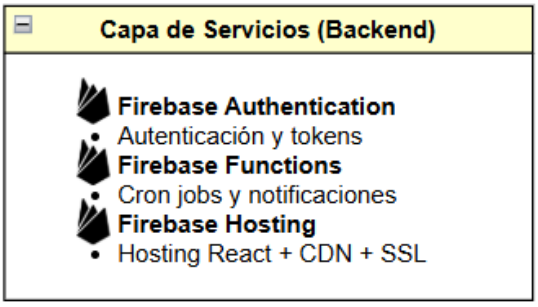

Figura 11: Capa de Servicios (Backend)

Capa de Datos

La persistencia de datos se gestiona mediante Firebase Firestore, una base de datos NoSQL. Esta tecnología nos permite una sincronización en tiempo real entre el frontend y el backend.

Firestore se adapta automáticamente a la demanda, ofreciendo una escalabilidad automática, esta solución resulta ideal para aplicaciones que pueden tener fluctuaciones en el tráfico o que crecen rápidamente. Además, proporciona un sistema de reglas de seguridad configurable para definir quién puede leer o escribir datos.

Su estructura se basa en documentos y colecciones, lo que da una gran flexibilidad para modelar distintos tipos de datos. Esto permite organizar la información de forma intuitiva.

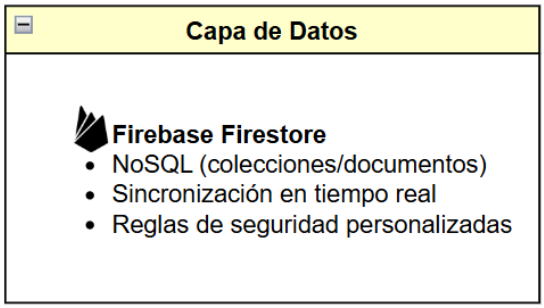

Figura 12: Capa de Datos

3.2.2. Beneficios

Esta arquitectura serverless proporciona múltiples beneficios:

Escalabilidad automática: Los servicios de Firebase escalan automáticamente según la demanda sin intervención manual.

Seguridad integrada: Firebase proporciona autenticación robusta y reglas de seguridad configurables.

Desarrollo simplificado: No se necesita gestionar servidores, bases de datos ni infraestructura de red.

Rendimiento optimizado: La CDN global y el caching automático mejoran los tiempos de respuesta.

Costo eficiente: Firebase utiliza un modelo de pago por uso que optimiza los gastos operativos. Durante el desarrollo del proyecto, el consumo de recursos se mantiene dentro del plan gratuito.

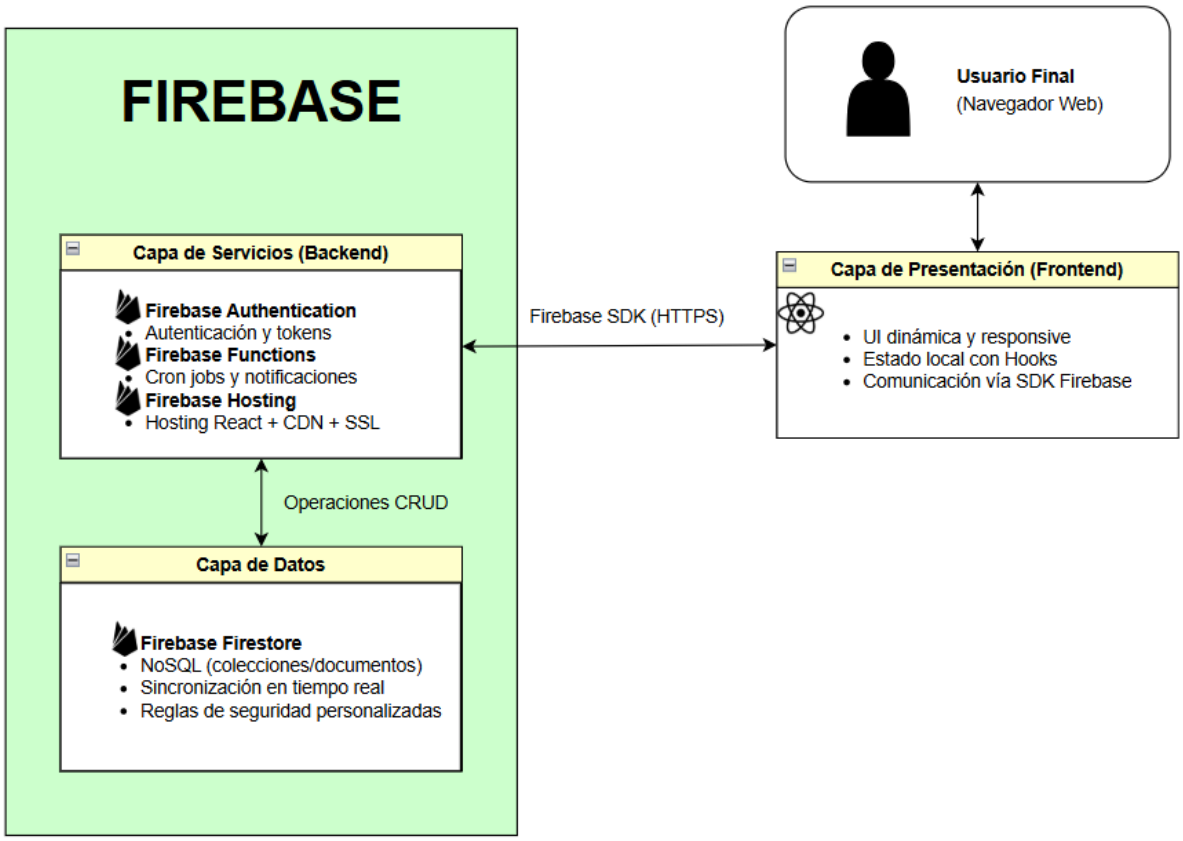

Figura 13: Arquitectura serverless (Creación propia)

3.3. Casos de uso

Los casos de uso describen las **interacciones** entre los **actores** del sistema (usuarios y administradores) y las **funcionalidades** que tiene la aplicación. Se han separado en tres módulos principales según su funcionalidad: autenticación, gestión de objetivos para usuarios regulares y administración del sistema.

3.3.1. Actores del sistema

Se han identificado dos tipos de actores principales:

Usuario: Es la persona que utiliza la aplicación para gestionar sus objetivos personales.

Administrador: Usuario con permisos elevados que puede gestionar objetivos de otros usuarios.

3.3.2. Casos de Uso por Módulo

Módulo de Autenticación

Código	CU-01
Nombre	Iniciar Sesión
Actor	Usuario, Administrador
Descripción	Permite al usuario acceder al sistema mediante email y contraseña
Precondiciones	El usuario debe estar registrado y tener el email verificado
Postcondiciones	El usuario accede a su dashboard personalizado
Flujo Principal	El usuario ingresa credenciales, el sistema valida con Firebase Auth y redirige según el rol

Tabla 13: CU-01 Iniciar sesión

Código	CU-02
Nombre	Registrarse
Actor	Usuario, Administrador
Descripción	Permite crear una nueva cuenta en el sistema
Precondiciones	El email no debe estar previamente registrado
Postcondiciones	Se crea la cuenta y se envía email de verificación
Flujo Principal	El usuario proporciona email y contraseña, el sistema crea la cuenta en Firebase Auth

Tabla 14: CU-02 Registrarse

Código	CU-03
Nombre	Cerrar Sesión
Actor	Usuario, Administrador
Descripción	Permite al usuario salir del sistema de forma segura
Precondiciones	El usuario debe estar autenticado
Postcondiciones	La sesión se cierra y se redirige al login
Flujo Principal	El usuario selecciona cerrar sesión, el sistema invalida el token de Firebase

Tabla 15: CU-03 Cerrar Sesión

Código	CU-04
Nombre	Verificar Correo Electrónico
Actor	Usuario, Administrador
Descripción	Proceso obligatorio para activar la cuenta tras el registro
Precondiciones	El usuario debe haber completado el registro
Postcondiciones	La cuenta queda verificada y habilitada para uso completo
Flujo Principal	El usuario recibe email, hace click en enlace, Firebase valida y activa la cuenta

Tabla 16: CU-04 Verificar Correo Electrónico

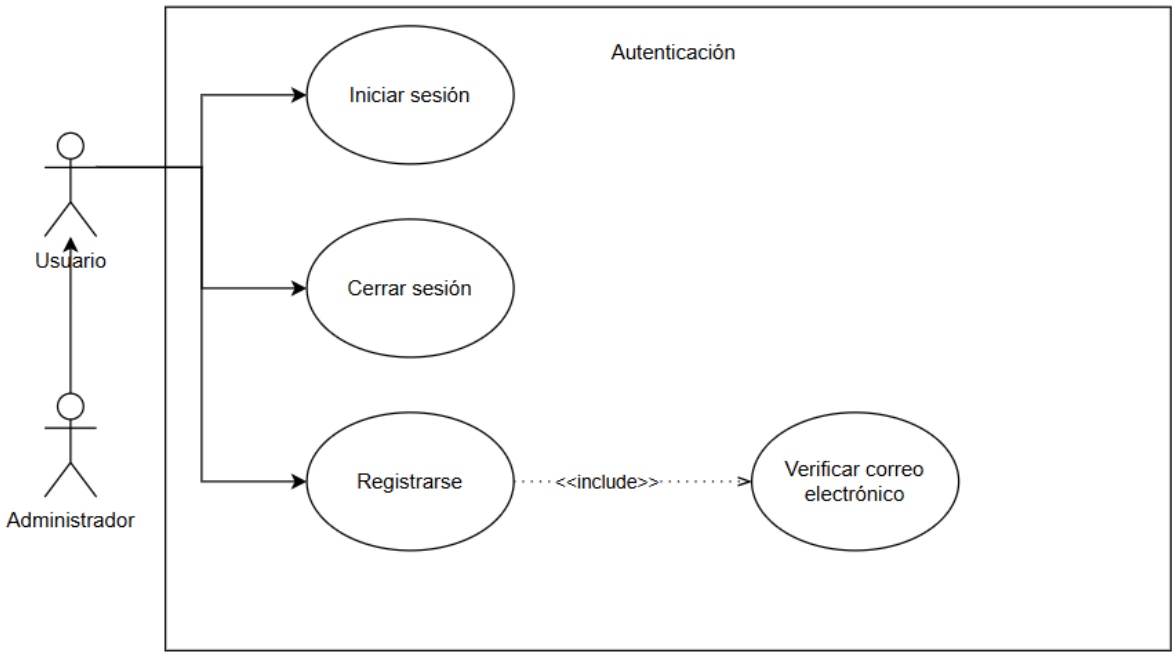

Figura 14: Diagrama de Casos de Uso - Autenticación

Módulo de Gestión de Objetivos (Usuario)

Código	CU-05
Nombre	Crear Objetivo
Actor	Usuario, Administrador
Descripción	Permite crear un nuevo objetivo personal con título, descripción y fecha límite
Precondiciones	El usuario debe estar autenticado
Postcondiciones	El objetivo se almacena en Firestore asociado al usuario
Flujo Principal	El usuario completa el formulario, el sistema valida y guarda en la base de datos

Tabla 17: CU-05 Crear Objetivo

Código	CU-06

Nombre	Editar Objetivo
Actor	Usuario, Administrador
Descripción	Permite modificar un objetivo existente
Precondiciones	El objetivo debe pertenecer al usuario autenticado
Postcondiciones	Los cambios se reflejan en tiempo real en la interfaz
Flujo Principal	El usuario selecciona editar, modifica campos y confirma los cambios

Tabla 18: CU-06 Editar Objetivo

Código	**CU-07**
Nombre	Ver Objetivo
Actor	Usuario, Administrador
Descripción	Permite visualizar la lista de objetivos personales
Precondiciones	El usuario debe estar autenticado
Postcondiciones	Se muestran los objetivos con su estado actual
Flujo Principal	El sistema recupera objetivos de Firestore y los presenta en la interfaz

Tabla 19: CU-07 Ver Objetivo

Código	**CU-08**
Nombre	Eliminar Objetivo
Actor	Usuario, Administrador
Descripción	Permite eliminar un objetivo de forma permanente
Precondiciones	El objetivo debe pertenecer al usuario autenticado
Postcondiciones	El objetivo se elimina de la base de datos

Flujo Principal	El usuario confirma la eliminación, el sistema borra el registro

Tabla 20: CU-08 Eliminar Objetivo

Código	CU-09
Nombre	Completar Objetivo
Actor	Usuario, Administrador
Descripción	Permite marcar un objetivo como completado
Precondiciones	El objetivo debe existir y no estar previamente completado
Postcondiciones	El objetivo se marca como completado y se envía notificación de logro
Flujo Principal	El usuario marca como completado, el sistema actualiza el estado y dispara notificación

Tabla 21: CU-09 Completar Objetivo

Código	CU-10
Nombre	Personalizar Dashboard
Actor	Usuario, Administrador
Descripción	Permite configurar qué componentes visualizar en el dashboard
Precondiciones	El usuario debe estar autenticado
Postcondiciones	La configuración se guarda y se aplica al dashboard
Flujo Principal	El usuario selecciona componentes a mostrar/ocultar (progreso general, lista de objetivos, gráfico de objetivos)

Tabla 22: CU-10 Personalizar Dashoard

Código	CU-11
Nombre	Añadir Comentario
Actor	Usuario, Administrador
Descripción	Permite agregar comentarios o notas a un objetivo
Precondiciones	El objetivo debe existir y pertenecer al usuario
Postcondiciones	El comentario se asocia al objetivo
Flujo Principal	El usuario ingresa comentario, el sistema lo guarda y actualiza la vista

Tabla 23: CU-11 Añadir comentario

Código	CU-12
Nombre	Eliminar Comentario
Actor	Usuario, Administrador
Descripción	Permite eliminar comentarios previamente agregados
Precondiciones	El comentario debe pertenecer al usuario autenticado
Postcondiciones	El comentario se elimina permanentemente
Flujo Principal	El usuario confirma eliminación, el sistema borra el comentario

Tabla 24: CU-12 Eliminar Comentario

Código	CU-13
Nombre	Ver Progreso de Tiempo

Actor	Usuario, Administrador
Descripción	Permite visualizar el porcentaje de tiempo transcurrido desde la creación hasta la fecha límite
Precondiciones	El objetivo debe tener fecha límite establecida
Postcondiciones	Se muestra el progreso temporal del objetivo
Flujo Principal	El sistema calcula y muestra el porcentaje de tiempo transcurrido

Tabla 25: CU-13 Ver Progreso de Tiempo

Código	CU-14
Nombre	Ver Progreso General
Actor	Usuario, Administrador
Descripción	Permite visualizar estadísticas consolidadas de todos los objetivos
Precondiciones	El usuario debe tener objetivos creados
Postcondiciones	Se muestran métricas generales de objetivos
Flujo Principal	El sistema calcula estadísticas globales y las presenta gráficamente

Tabla 26: CU-14 Ver Progreso General

Código	CU-15
Nombre	Ver Gráfico de Objetivos
Actor	Usuario, Administrador
Descripción	Permite visualizar representaciones gráficas del progreso de objetivos
Precondiciones	El usuario debe tener objetivos para graficar
Postcondiciones	Se muestra un gráfico interactivo (donut chart)

Flujo Principal	El sistema genera gráficos basados en los objetivos completados

Tabla 27: CU-15 Ver Gráfico de Objetivos

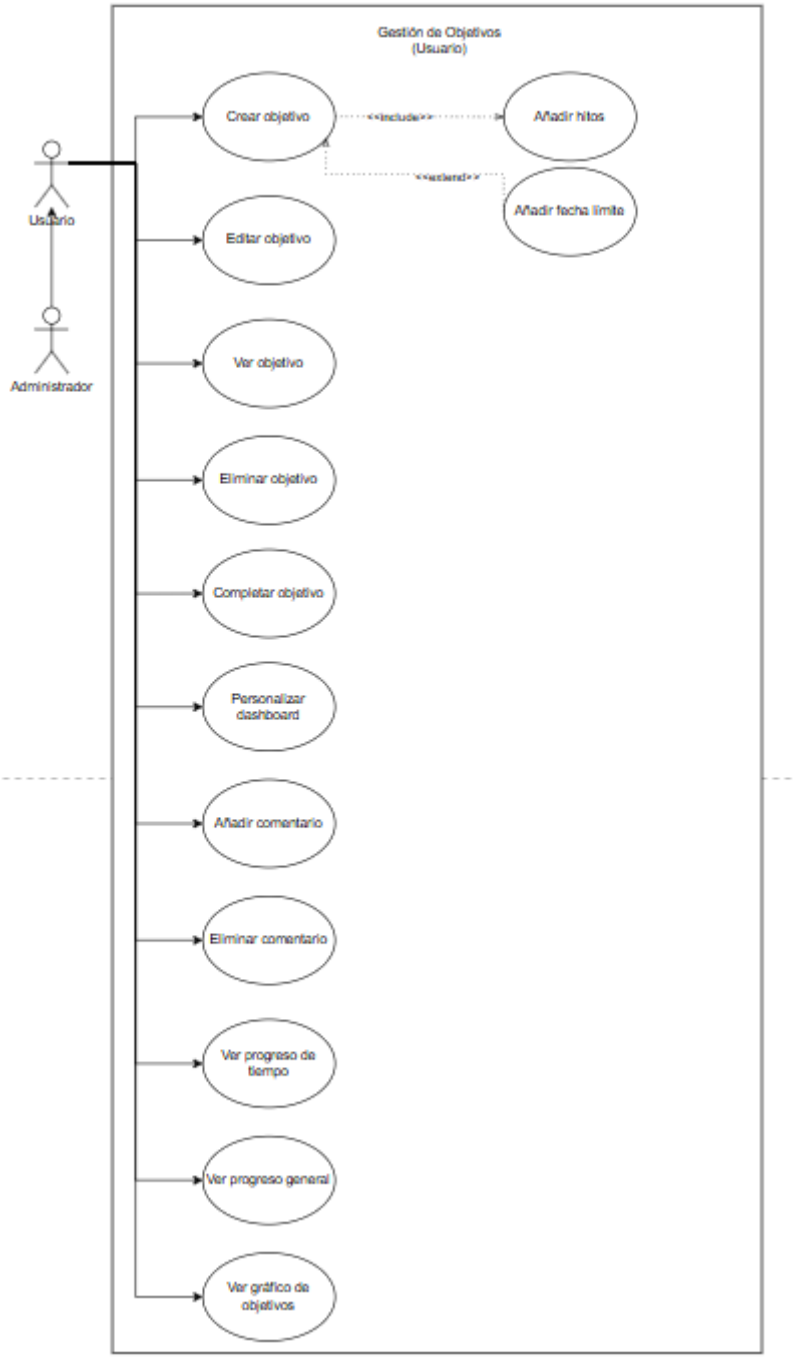

Figura 15: Diagrama de Casos de Uso – Gestión de Objetivos (Usuario)

Módulo de Administración

Código	CU-16
Nombre	Ver Usuarios
Actor	Administrador
Descripción	Permite visualizar la lista de usuarios registrados en el sistema
Precondiciones	El usuario debe tener rol de administrador
Postcondiciones	Se muestra información de usuarios registrados
Flujo Principal	El sistema recupera usuarios de Firebase Auth y los presenta en tabla

Tabla 28: CU-16 Ver Usuarios

Código	CU-17
Nombre	Añadir Objetivos a Otros Usuarios
Actor	Administrador
Descripción	Permite crear objetivos para otros usuarios del sistema
Precondiciones	El administrador debe estar autenticado y el usuario destino debe existir
Postcondiciones	El objetivo se crea y se asocia al usuario especificado
Flujo Principal	El administrador selecciona usuario, crea objetivo y lo asigna

Tabla 29: CU-17 Añadir Objetivos a Otros Usuarios

Código	CU-18

Nombre	Editar Objetivos de Otros Usuarios
Actor	Administrador
Descripción	Permite modificar objetivos de cualquier usuario
Precondiciones	El administrador debe estar autenticado y el objetivo debe existir
Postcondiciones	Los cambios se reflejan en el objetivo del usuario
Flujo Principal	El administrador selecciona objetivo, modifica campos y confirma cambios

Tabla 30: CU-18 Editar Objetivos de Otros Usuarios

Código	**CU-19**
Nombre	Eliminar Objetivos de Otros Usuarios
Actor	Administrador
Descripción	Permite eliminar objetivos de cualquier usuario
Precondiciones	El administrador debe estar autenticado y el objetivo debe existir
Postcondiciones	El objetivo se elimina permanentemente
Flujo Principal	El administrador confirma eliminación, el sistema borra el objetivo

Tabla 31: CU-19 Eliminar Objetivos de Otros Usuarios

Código	**CU-20**
Nombre	Completar Objetivos de Otros Usuarios
Actor	Administrador
Descripción	Permite marcar como completados objetivos de otros usuarios

Precondiciones	El administrador debe estar autenticado y el objetivo debe existir
Postcondiciones	El objetivo se marca como completado y se notifica al usuario
Flujo Principal	El administrador marca objetivo como completado, el sistema actualiza estado y envía notificación

Tabla 32: CU-20 Completar Objetivos de Otros Usuarios

Código	CU-21
Nombre	Comentar en Objetivos de Otros Usuarios
Actor	Administrador
Descripción	Permite agregar comentarios o notas a objetivos de cualquier usuario
Precondiciones	El administrador debe estar autenticado y el objetivo debe existir
Postcondiciones	El comentario se asocia al objetivo del usuario especificado
Flujo Principal	El administrador selecciona objetivo de otro usuario, ingresa comentario y lo guarda en el sistema

Tabla 33: CU-21 Comentar en Objetivos de Otros Usuarios

Código	CU-22
Nombre	Eliminar Comentarios de Cualquier Usuario
Actor	Administrador
Descripción	Permite eliminar comentarios de objetivos de cualquier usuario

Precondiciones	El administrador debe estar autenticado y el comentario debe existir
Postcondiciones	El comentario se elimina permanentemente del sistema
Flujo Principal	El administrador selecciona comentario, confirma eliminación y el sistema borra el registro

Tabla 34: CU-22 Eliminar Comentarios de Cualquier Usuario

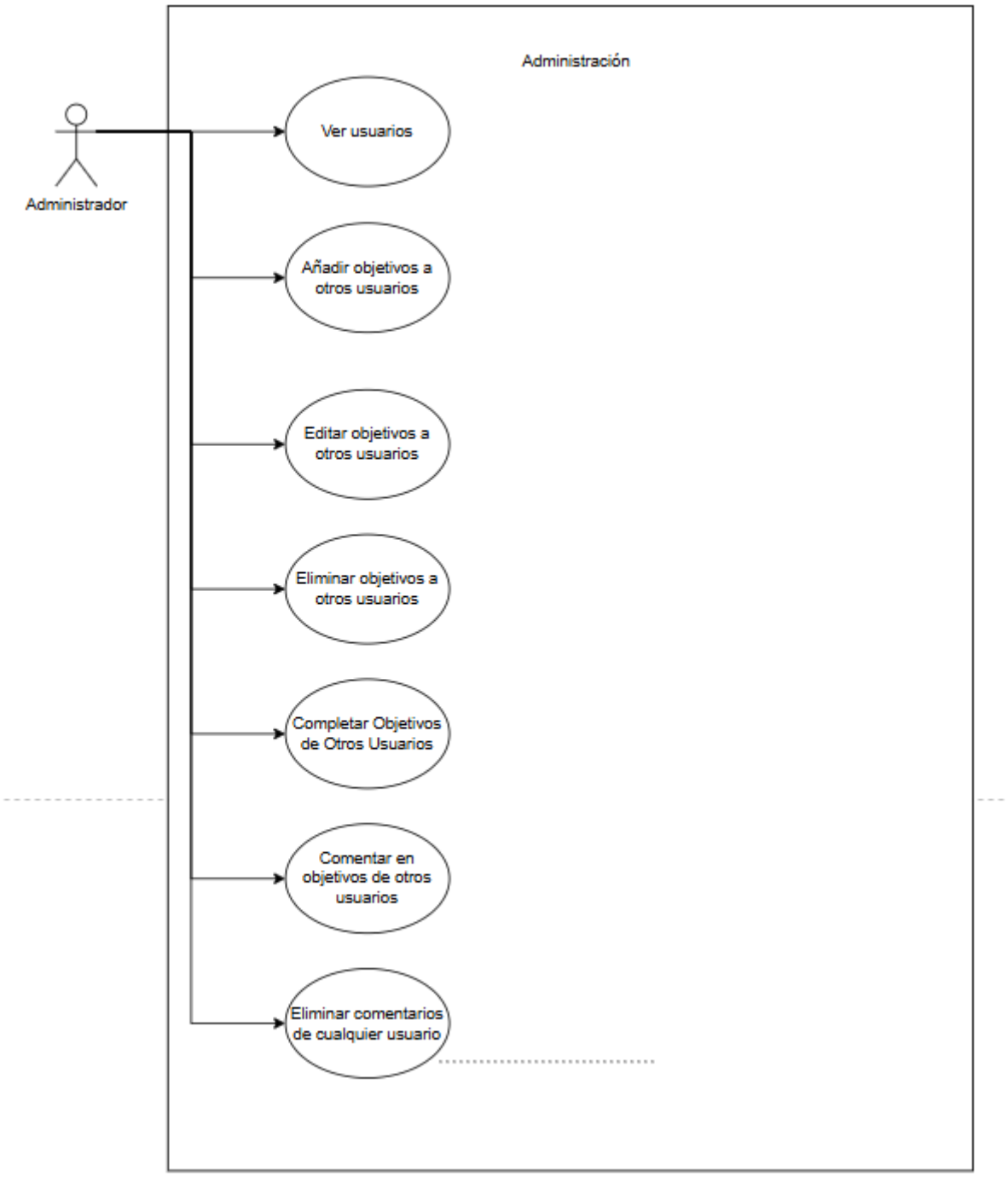

Figura 16: Diagrama de Casos de Uso – Administración

La implementación de estos casos de uso garantiza que el sistema cubra todas las necesidades que se han identificado tanto para usuarios regulares como para administradores, proporcionando así una experiencia completa de gestión de objetivos personales.

3.4. Diseño de la Base de Datos

El diseño de la base de datos se basa en Firebase Firestore, una base de datos NoSQL orientada a documentos. Esto nos ofrece una flexibilidad en la estructura de los datos y escalabilidad automática según la demanda. La elección de una base de datos NoSQL es apropiada para este tipo de aplicación debido a la naturaleza variable de los objetivos personales y la necesidad por la sincronización a tiempo real.

Características de Firebase Firestore

Firestore organiza los datos en colecciones y documentos, donde cada documento contiene campos con valores, además permite incluir subcolecciones anidadas. Esta estructura jerárquica permite modelar relaciones complejas manteniendo la eficiencia en las consultas y una flexibilidad si evoluciona el esquema.

Estructura general de la base de datos

La base de datos está compuesta por dos colecciones principales que almacenan toda la información necesaria: **users** y **objectives**.

Colección: users

Esta colección almacena la información de los usuarios registrados en el sistema. Cada documento representa a un usuario único identificado por su UID de Firebase Authentication.

```
{
  uid: "aVciVsL6tjeqgWgZF8QnS3x81Or2",
  email: "diegocr565@gmail.com",
  name: "Diego Casero Ramón",
  createdAt: Timestamp("2 de julio de 2025, 2:10:52 a.m. UTC+2"),
  dashboardWidgets: {
    0: "chart",
    1: "objectives",
    2: "progress"
  }
}
```

uid: Identificador único del usuario generado por Firebase Authentication

email: Dirección de correo electrónico del usuario, utilizada para autenticación

name: Nombre completo del usuario

createdAt: Timestamp que registra la fecha y hora de la creación de la cuenta

dashboardWidgets: Objeto que define la configuración personalizada del dashboard, especificando qué componentes mostrar

Colección: objectives

Esta colección contiene todos los objetivos creados por usuarios del sistema. Cada documento representa un objetivo individual con toda la información asociada.

```
{
  uid: "aVciVsL6tjeqgWgZF8QnS3x81Or2",
  createdAt: Timestamp("2 de julio de 2025, 2:29:43 a.m. UTC+2"),
  deadline: Timestamp("5 de julio de 2025, 2:00:00 a.m. UTC+2"),
  text: "Preparar cumpleaños",
  milestones: {
    0: {
      completed: true,
      title: "Cocinar"
    },
    1: {
      completed: true,
      title: "Comprar decorativos"
    },
    2: {
      completed: true,
      title: "Invitar a amigos y familiares"
    }
  },
  comments: {
    0: {
      createdAt: Timestamp("2 de julio de 2025, 2:30:38 a.m. UTC+2"),
      text: "Martín es alérgico a los frutos secos",
      user: "diegocr565@gmail.com"
    },
    1: {
      createdAt: Timestamp("9 de julio de 2025, 1:58:28 a.m. UTC+2"),
      text: "Diego, el alérgico no es Martín, sino Marcos.",
      user: "conquistalogros@gmail.com"
    }
  }
}
```

uid: Referencia del usuario propietario del objetivo (clave foránea hacia users)

createdAt: Timestamp de creación del objetivo

deadline: Fecha límite para completar el objetivo

text: Descripción del objetivo

milestones: Objeto que contiene hitos (o sub-objetivos) del objetivo principal

- Cada milestone tiene un índice como clave
- **completed:** Boolean que indica si el hito se ha completado
- **title:** Descripción del hito

comments: Objeto que almacena comentarios asociados al objetivo

- Cada comentario tiene un índice como clave
- **createdAt:** Timestamp de creación del comentario
- **text:** Descripción del comentario
- **user:** email del usuario que creó el comentario

Relaciones entre colecciones

La relación users → objectives es uno a muchos (1:N). Esto significa que un usuario puede tener múltiples objetivos. La clave foránea uid en objectives referencia el documento del usuario en users. Esta estructura permite consultas eficientes para obtener todos los objetivos de un usuario específico.

Ventajas del diseño elegido

Flexibilidad: La estructura NoSQL permite agregar campos sin tener que modificar esquemas

Escalabilidad: Firestore escala automáticamente según la demanda

Tiempo real: Sincronización automática entre cliente y servidor

Consultas eficientes: Estructura optimizada para los patrones de acceso más comunes

3.5. Diseño de la Interfaz de Usuario

El diseño de la interfaz de la aplicación se basa en principios de simplicidad y claridad. La aplicación adopta un enfoque minimalista que facilita la gestión de objetivos sin distracciones.

3.5.1. Tecnologías utilizadas

CSS personalizado: Estilos en línea en React para control granular y estilos dinámicos

Recharts: Librería para gráficos interactivos y responsivos

Componentes modulares: Arquitectura basada en componentes reutilizables

3.5.2. Sistema de colores

Se han utilizado los siguientes colores según qué representa cada elemento:

Azul: Elementos primarios

Verde: Progreso completado y confirmaciones

Naranja: Advertencias

Rojo: Errores y eliminaciones

Amarillo: Acciones de edición

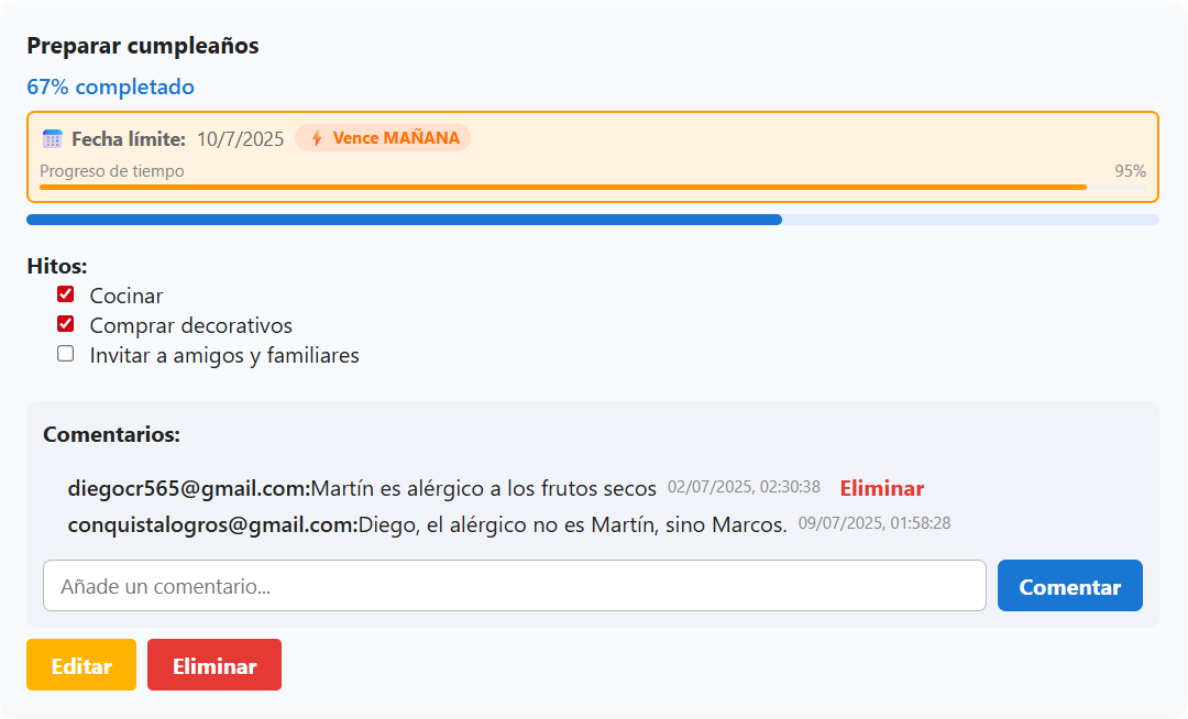

Figura 17: Ejemplo 1 Sistema de colores

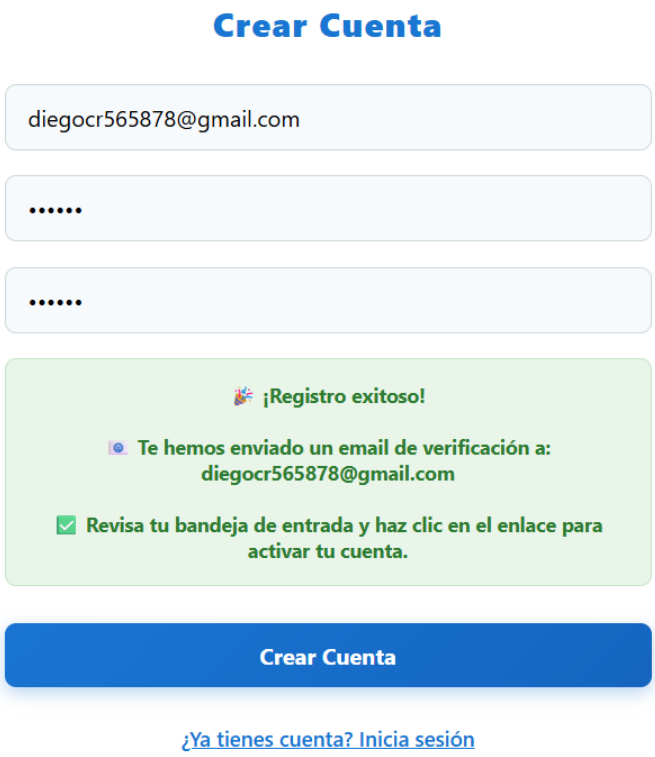

Figura 18: Ejemplo 2 Sistema de colores

3.5.3. Componentes de Visualización

Se han creado los siguientes componentes de visualización utilizando la librería Recharts:

ProgressChart (Gráfico de Barras): Visualización horizontal del progreso de objetivos con codificación por colores: verde para completados, azul para en progreso.

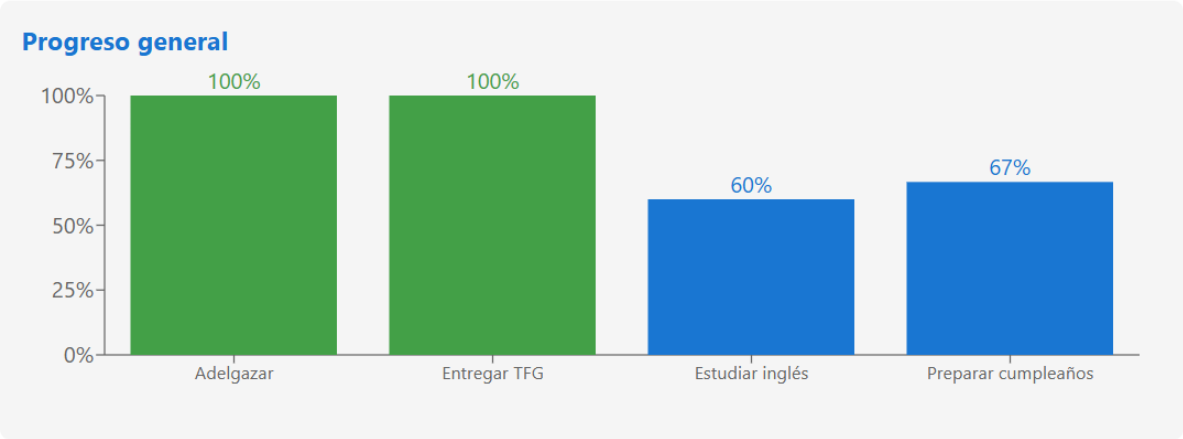

Figura 19: Ejemplo ProgressChart

ObjectivesDonutChart (Gráfico Donut): Gráfico circular que muestra la proporción de objetivos completados vs pendientes.

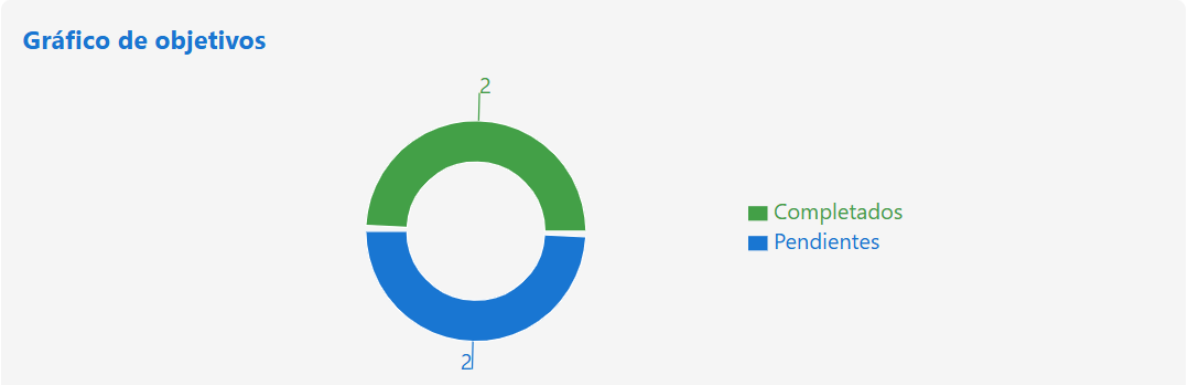

Figura 20: Ejemplo ObjectivesDonutChart

3.5.4. Indicadores de progreso

Los objetivos tienen barras de progreso para hitos y el tiempo transcurrido. Según la proximidad de la fecha límite hay un color distinto para indicar los diferentes niveles de urgencia.

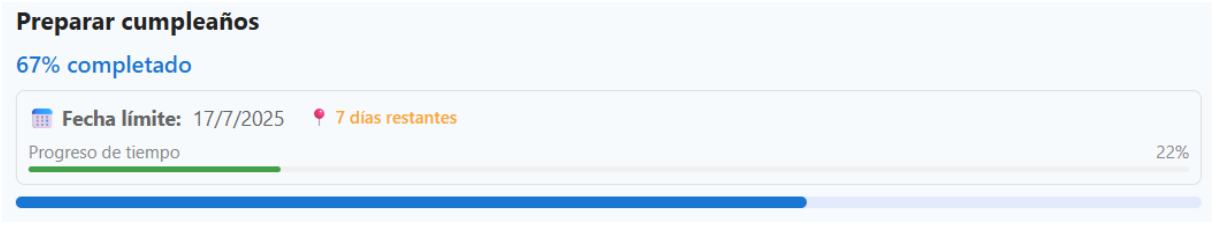

Figura 21: Fecha límite lejana

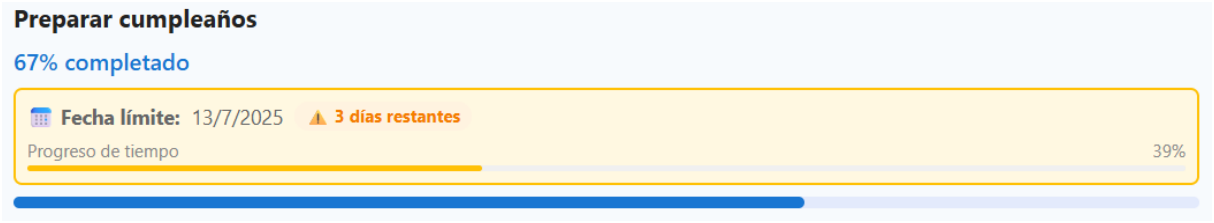

Figura 22: Fecha límite cercana

Figura 23: Fecha límite mañana

Figura 24: Fecha límite hoy

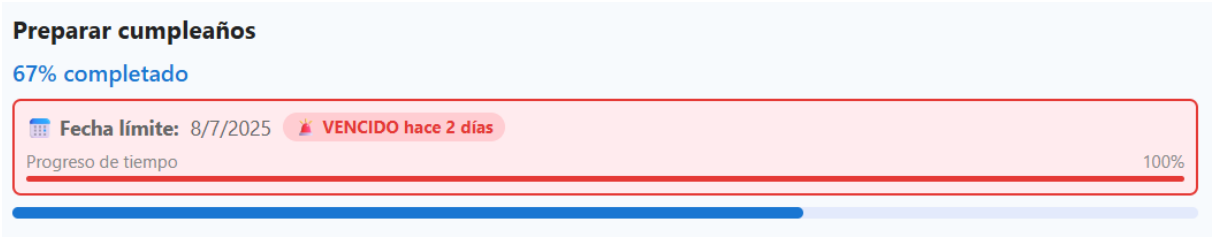

Figura 25: Fecha límite pasada

3.5.5. Responsividad y Accesibilidad

La aplicación utiliza CSS para layouts responsivos, tipografía escalable y contraste de colores apropiados. El diseño se adapta automáticamente a diferentes dispositivos manteniendo la funcionalidad.

El resultado es una interfaz coherente, intuitiva y eficiente que facilita a los usuarios la gestión de objetivos manteniendo la motivación.

4. Implementación y Desarrollo

En este punto se detalla el proceso de desarrollo del sistema, abarcando desde la configuración inicial del entorno hasta la integración completa entre frontend y backend. Se presenta la implementación práctica de los diseños y arquitecturas definidos en el punto anterior.

4.1. Configuración del Entorno de Desarrollo

La configuración del entorno de desarrollo se estableció utilizando tecnologías modernas de desarrollo web, priorizando la eficiencia, la escalabilidad y el mantenimiento.

El proyecto se inicializó utilizando **Vite**. Esto se deber a que ofrece un tiempo de inicio significativamente menor y recarga instantánea durante el desarrollo:

npm create vite@latest objetivos-web -- --template react

cd objetivos-web

npm install

4.1.1. Dependencias

Se encuentran en el fichero package.json

```
"dependencies": {
    "firebase": "^10.14.1",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.22.3",
    "recharts": "^2.15.4"
  }
```

Firebase: Kit de desarrallo para integración con servicios de Firebase

React: Librería principal para la construcción de interfaces de usuario

React DOM: Renderizado de componentes React en el DOM

React Router DOM: Gestión de rutas y navegación

Recharts: Librería para generación de gráficos

4.1.2. Configuración de Firebase

Firebase CLI

Se debe instalar y configurar:

npm install -g firebase-tools

firebase login

firebase init

Configuración de Firebase en el Cliente:

Firebase te da un fichero automático con las claves necesarias. Si el código lo puede consultar otras personas, es totalmente recomendable poner estas claves en .env . Este fichero debe estar en el .gitignore para que no se suba al repositorio. Así quedaría firebase.jsx usando las variables de entorno.

```
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: import.meta.env.VITE_FIREBASE_API_KEY,
  authDomain: import.meta.env.VITE_FIREBASE_AUTH_DOMAIN,
  projectId: import.meta.env.VITE_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.VITE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: import.meta.env.VITE_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.VITE_FIREBASE_APP_ID
};

const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export const db = getFirestore(app);
```

4.1.3. Flujo de Desarrollo

Desarrollo local

Instalar dependencias:

npm install

Iniciar servidor de desarrollo:

npm run dev

Construcción y despliegue

Constuir aplicación para producción:

npm run build

Desplegar en Firebase Hosting:

firebase deploy

Esta configuración proporciona un entorno de desarrollo moderno y eficiente, optimizado tanto para desarrollo como para producción.

4.2. Desarrollo del Frontend

El desarrollo del frontend se realizó con React.js con una arquitectura basada en componentes modulares y hooks personalizados para la gestión del estado. La aplicación implementa un patrón de Single Page Application (SPA) con navegación controlada y rutas protegidas según el usuario.

4.2.1. Componentes Principales

La aplicación se organiza en cuatro páginas principales:

Login.jsx – Autenticación de usuarios

Es el encargado del proceso de inicio de sesión con validación obligatoria de email verificado:

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setLoading(true);

  try {
    const userCredential = await signInWithEmailAndPassword(auth, email,
password);
    const user = userCredential.user;

    // Verificar email antes de permitir acceso
    if (!user.emailVerified) {
      setError("📧 Debes verificar tu email antes de continuar.");
      setShowResendButton(true);
      await auth.signOut(); // Cerrar sesión inmediatamente
      return;
    }
  } catch (err) {
    setError("Credenciales incorrectas.");
  }
  setLoading(false);
};
```

El código completo incluye gestión de estados, validación de formularios y la funcionalidad de reenviar la verificación. Se ha omitido por brevedad.

Register.jsx – Registro de nuevos usuarios

Se encarga del flujo de registro: creación del usuario, almacenamiento en Firestore y envío de verificación:

```
const handleSubmit = async (e) => {
  e.preventDefault();

  try {
    // 1. Crear usuario en Firebase Auth
    const userCredential = await createUserWithEmailAndPassword(auth, email,
password);
    const user = userCredential.user;
```

```
    f
    // 2. Guardar información adicional en Firestore
    await setDoc(doc(db, "users", user.uid), {
      email: user.email,
      name: fullName,
      createdAt: new Date(),
      dashboardWidgets: { 0: "chart", 1: "objectives", 2: "progress" }
    });

    // 3. Enviar email de verificación
    await sendEmailVerification(user);
    await auth.signOut(); // Forzar verificación antes del acceso

  } catch (err) {
    setError(getErrorMessage(err.code));
  }
};
```

El código completo incluye validación de contraseñas, manejo de errores específicos de Firebase y estados de UI.

Dashboard.jsx – Panel principal del usuario

Es el componente con más funcionalidades. La función de creación de objetivos muestra la integración con el hook personalizado useObjectives:

```
const Dashboard = () => {
  const { objectives, addObjective, updateObjective, deleteObjective } =
useObjectives(auth.currentUser?.uid);
  const [widgets, setWidgets] = useState(["progress", "objectives", "chart"]);

  const handleAddObjective = async (e) => {
    e.preventDefault();

    // Validación de hitos obligatorios
    if (milestones.length === 0) {
      setError("Debes añadir al menos un hito.");
      return;
    }

    const deadline = newObjectiveDeadline ? Timestamp.fromDate(new
Date(newObjectiveDeadline)) : null;

    await addObjective({
      uid: auth.currentUser.uid,
```

```
      text: newObjective,
      milestones: milestones.map(title => ({ title, completed: false })),
      deadline: deadline,
      comments: []
    });

    // Limpiar formulario
    setNewObjective("");
    setMilestones([]);
  };
```

El código de Dashboard también incluye la gestión de widgets personalizables, sistema de comentarios, gestión completa (CRUD) de objetivos, cálculo matemático del progreso y gestión de fechas límite con alertas, entre otras funcionalidades.

Admin – Panel de Administración

Interfaz que permite a los administradores gestionar objetivos de cualquier usuario mediante el email:

```
const handleAddObjective = async (e) => {
  e.preventDefault();

  // Buscar UID por email
  const uid = await getUidByEmail(newEmail);
  if (!uid) {
    setError("No se encontró ningún usuario con ese correo.");
    return;
  }

  await addObjective({
    uid, // UID del usuario objetivo, no del admin
    text: newObjective,
    milestones: milestones.map(title => ({ title, completed: false })),
    deadline: deadline,
  });
};
```

El panel incluye listado de todos los objetivos del sistema y tiene las mismas funcionalidades que el dashboard normal.

MainContainer.jsx proporciona una distribución consistente para todas las páginas:

```
const MainContainer = ({ children, maxWidth = 500 }) => (
  <div style={{
    minHeight: "100vh",
    display: "flex",
    alignItems: "center",
    justifyContent: "center",
    background: "linear-gradient(135deg, #667eea 0%, #764ba2 100%)",
  }}>
    <div style={{
      background: "#fff",
      borderRadius: 16,
      padding: 32,
      maxWidth: maxWidth,
      boxShadow: "0 8px 32px rgba(0, 0, 0, 0.1)",
    }}>
      {children}
    </div>
  </div>
);
```

Centraliza el diseño visual y asegura consistencia en toda la aplicación.

4.2.2. Gestión del estado y hooks personalizados

Hook useObjectives

El hook más importante del proyecto encapsula toda la lógica de gestión de objetivos, proporcionando así una interfaz limpia para realizar operaciones CRUD:

```
export function useObjectives(uid = null) {
  const [objectives, setObjectives] = useState([]);

  // Función para obtener objetivos con filtrado opcional por usuario
  const fetchObjectives = useCallback(async () => {
    const q = uid
      ? query(collection(db, "objectives"), where("uid", "==", uid)) //
Usuario específico
      : collection(db, "objectives"); // Todos los objetivos (admin)

    const querySnapshot = await getDocs(q);
    const objs = [];
    querySnapshot.forEach((docu) => {
```

```
      objs.push({ id: docu.id, ...docu.data() });
    });
    setObjectives(objs);
  }, [uid]);

  const addObjective = async (data) => {
    await addDoc(collection(db, "objectives"), {
      ...data,
      createdAt: Timestamp.now()
    });
    fetchObjectives(); // Refrescar lista
  };

  return {
    objectives,
    fetchObjectives,
    addObjective,
    updateObjective,
    deleteObjective,
    addComment,
    deleteComment
  };
}
```

El hook también incluye funciones para la actualización, eliminación y gestión de comentarios. Este hook se usará para usuarios y administradores, permitiendo su reutilización.

4.2.3. Autenticación y Rutas Protegidas

App.jsx – Sistema de enrutamiento

Es el componente raíz. Implementa un sistema de autenticación con redirección automática según el rol:

```
function App() {
  const [user, setUser] = useState(undefined); // undefined = cargando

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (firebaseUser) => {
      // Solo permitir acceso con email verificado
      if (firebaseUser && !firebaseUser.emailVerified) {
        setUser(null); // Tratar como no autenticado
```

```
      } else {
        setUser(firebaseUser || null);
      }
    });
    return () => unsubscribe();
  }, []);

  // Estado de carga inicial
  if (user === undefined) {
    return <div style={{ textAlign: 'center', padding: '50px'
}}>Cargando...</div>;
  }

  return (
    <Router>
      <Routes>
        <Route path="/login" element={
          !user ? <Login /> :
          <Navigate to={user.email === "conquistalogros@gmail.com" ? "/admin"
: "/dashboard"} />
        } />
        <Route path="/dashboard" element={
          user ? <Dashboard /> : <Navigate to="/login" />
        } />
        <Route path="/admin" element={
          user && user.email === "conquistalogros@gmail.com" ?
          <Admin /> : <Navigate to="/login" />
        } />
      </Routes>
    </Router>
  );
}
```

El enrutador también incluye rutas catch-all (todas las rutas que no coincidan con las definidas) y redirecciones automáticas.

Caracteríticas:

- Bloqueo total hasta la verificación del correo
- Funcionalidad de reenvío
- Detección automática de administrador por email
- Persistencia automática de sesión con Firebase

4.3. Desarrollo del Backend

El backend del sistema se implementó utilizando Firebase como plataforma Backend-as-a-Service (BaaS). Utilizamos esta tecnología por sus servicios integrados para la autenticación, base de datos, funciones serverless y hosting. Esta arquitectura ofrece escalabilidad automática y reduce la complejidad en infraestructura.

La aplicación integra cuatro servicios de Firebase:

- **Firebase Authentication:** Gestión de usuarios y autenticación
- **Cloud Firestore:** Base de datos NoSQL en tiempo real
- **Firebase Functions:** Funciones serverless para la lógica del backend
- **Firebase Hosting:** Alojamiento web para la aplicación

Anteriormente se explicó cómo configurar el firebase.jsx para soportar la autenticación. También se creó el diseño de la base de datos con las colecciones users y objectives. Además, se vio cómo construir la aplicación para producción y cómo desplegarla.

Este punto se centra en **Firebase Functions**, donde se definió unas funciones serverless para enviar correos electrónicos a los usuarios.

Configuración de Gmail para las notificaciones

El sistema de notificaciones se configuró utilizando Gmail SMTP con autenticación por contraseña de la aplicación:

firebase functions:config:set gmail.email="conquistalogros@gmail.com"

firebase functions:config:set gmail.password="contraseña-de-aplicacion"

Esta configuración permite que las Firebase Functions envíen emails automatizados utilizando una cuenta de Google.

Función principal: Verificación diaria automática

La función dailyCheck se ejecuta cada día a las 9:00 AM UTC para verificar las fechas límite de los objetivos y enviar los emails que correspondan:

```
exports.dailyCheck = onSchedule('0 9 * * *', async (event) => {
  console.log('Ejecutando verificación automática diaria...');

  const db = getFirestore();
  const now = new Date();
  now.setHours(0, 0, 0, 0);

  try {
    // Obtener todos los objetivos y usuarios simultáneamente
    const [objectivesSnapshot, usersSnapshot] = await Promise.all([
      db.collection('objectives').get(),
      db.collection('users').get()
    ]);

    // Crear mapa de usuarios para optimizar búsquedas
    const usersMap = {};
    usersSnapshot.forEach(doc => {
      usersMap[doc.id] = doc.data().email;
    });

    let emailsSent = 0;

    for (const doc of objectivesSnapshot.docs) {
      const obj = doc.data();
      const userEmail = usersMap[obj.uid];

      if (!obj.deadline || !userEmail) continue;

      // Calcular progreso automáticamente
      let progress = 0;
      if (obj.milestones && obj.milestones.length > 0) {
        const completed = obj.milestones.filter(m => m.completed).length;
        progress = (completed / obj.milestones.length) * 100;
      }

      // Saltar objetivos ya completados
      if (progress >= 100) continue;
```

```
      const deadline = new Date(obj.deadline.toDate());
      const daysLeft = Math.ceil((deadline - now) / (1000 * 60 * 60 * 24));

      // Enviar notificaciones según proximidad
      if (daysLeft === 0) {
        await sendNotificationEmail(userEmail, 'urgente', obj, progress,
daysLeft);
        emailsSent++;
      } else if (daysLeft === 1) {
        await sendNotificationEmail(userEmail, 'mañana', obj, progress,
daysLeft);
        emailsSent++;
      } else if (daysLeft === 3) {
        await sendNotificationEmail(userEmail, 'recordatorio', obj, progress,
daysLeft);
        emailsSent++;
      }
    }

    console.log(`Verificación completada. Emails enviados: ${emailsSent}`);
    return { success: true, emailsSent, timestamp: new Date().toISOString() };

  } catch (error) {
    console.error('ERROR:', error);
    throw error;
  }
});
```

La función completa también incluye manejo de objetivos vencidos con notificaciones semanales.

Procesamiento eficiente:

- Las consultas se realizan paralelamente con Promise.all() para obtener objetivos y usuarios simultáneamente
- Se crea un mapa de usuarios para evitar búsquedas repetitivas

Función de felicitación por completar objetivos

La función completationEmail se ejecutaba automáticamente cuando se completaban todos los hitos de un objetivo:

```
exports.completationEmail = onRequest(async (req, res) => {
```

```
  const email = req.query.email;
  const objetivo = req.query.objetivo;

  try {
    await sendNotificationEmail(email, 'completado', { text: objetivo }, 100,
0);
    res.json({
      success: true,
      message: `🎉 Email de felicitación enviado a ${email}`,
      objetivo: objetivo
    });
  } catch (error) {
    res.status(500).json({ success: false, error: error.message });
  }
});
```

Configuración de Nodemailer

```
const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransporter({
  service: 'gmail',
  auth: {
    user: gmailEmail.value(),
    pass: gmailPassword.value()
  }
});
```

Tipos de notificaciones enviadas

El sistema envía cinco tipos de notificaciones con distintos diseños HTML:

Urgente: Cuando el objetivo vence el mismo día

Mañana: Cuando queda 1 día para el vencimiento

Recordatorio: Cuando quedan 3 días

Vencido: Cada semana después del vencimiento

Completado: Felicitación al completar el objetivo

Ejemplo de la plantilla HTML para la notificación urgente:

```
case 'urgente':
  subject = '🚨 ¡Tu objetivo vence HOY!';
```

```
  html = `
    <div style="font-family: 'Segoe UI', Arial, sans-serif; max-width: 600px;
margin: 0 auto; border-radius: 12px; overflow: hidden;">
      <div style="background: linear-gradient(135deg, #e53935, #d32f2f);
color: white; padding: 30px; text-align: center;">
        <h1 style="margin: 0; font-size: 28px;">🚨 ¡URGENTE!</h1>
        <h2 style="margin: 15px 0 0 0; font-weight: 300;">Tu objetivo vence
HOY</h2>
      </div>
      <div style="background: white; padding: 30px;">
        <h3 style="color: #333; margin-top: 0;">📋 ${objetivo.text}</h3>
        <div style="background: #f5f5f5; padding: 15px; border-radius: 8px;">
          <p style="margin: 0; color: #666;">
            <strong>Progreso actual:</strong> ${Math.round(progress)}%
completado
          </p>
        </div>
        <p style="color: #e53935; font-weight: bold; text-align: center;">
          ⏰ ¡Es tu última oportunidad para terminarlo!
        </p>
      </div>
    </div>
  `;
  break;
```

Cada notificación utiliza colores y mensajes distintos para maximizar el impacto visual y la motivación del usuario.

Figura 26: Correo ejemplo Urgente

4.4. Integración y comunicación Frontend - Backend

La integración entre el frontend React y el backend Firebase se implementó usando patrones de comunicación que aprovechan las capacidades nativas de Firebase. Hay dos tipos de comunicación: comunicación directa con Firestore para operaciones CRUD y llamadas HTTP a Firebase Functions.

Comunicación directa con Firestore

El patrón principal utiliza el SDK de Firebase para la comunicación directa entre React y Firestore, elminando así la necesidad de APIs REST intermedias. Esta implementación se encapsula en el hook useObjectives:

```
import { collection, query, where, getDocs, addDoc, updateDoc, deleteDoc, doc,
arrayUnion, arrayRemove, Timestamp } from "firebase/firestore";

export function useObjectives(uid = null) {
  const [objectives, setObjectives] = useState([]);

  // Consulta directa a Firestore con filtrado opcional
  const fetchObjectives = useCallback(async () => {
    const q = uid
      ? query(collection(db, "objectives"), where("uid", "==", uid))
```

```
      : collection(db, "objectives");
    const querySnapshot = await getDocs(q);
    const objs = [];
    querySnapshot.forEach((docu) => {
      objs.push({ id: docu.id, ...docu.data() });
    });
    setObjectives(objs);
  }, [uid]);

  // Operaciones CRUD con actualización automática
  const addObjective = async (data) => {
    await addDoc(collection(db, "objectives"), {
      ...data,
      comments: [],
      createdAt: Timestamp.now()
    });
    fetchObjectives(); // Refresco inmediato del estado local
  };

  const updateObjective = async (id, data) => {
    await updateDoc(doc(db, "objectives", id), data);
    fetchObjectives();
  };
```

El hook completo incluye operaciones para eliminación y gestión de comentarios.

Una de las ventajas de la comunicación directa es la **latencia mínima**, ya que se eliminan las capas intermedias.

Llamadas HTTP a Firebase Functions

Para funcionalidades específicas que requieren procesamiento del lado del servidor, se realizan llamadas HTTP directas. Un ejemplo representativo es el envío de emails de felicitación:

```
const sendCompletionEmail = async (objectiveText) => {
  try {
    const response = await fetch(
      `https://us-central1-objetivos-
779ed.cloudfunctions.net/completationEmail?objetivo=${encodeURIComponent(objec
tiveText)}&email=${encodeURIComponent(auth.currentUser.email)}`
    );
    const result = await response.json();
```

```
    console.log('🎉 Email de felicitación enviado:', result);
    return result;
  } catch (error) {
    console.error('Error enviando email de felicitación:', error);
  }
};
```

Al completar un objetivo, la notificación vía email se realiza de forma instantánea gracias a sendCompletationEmail.

Nota: La función dailyCheck se ejecuta automáticamente cada día, luego el frontend no necesita comunicarse con ella ya que operan de forma independiente.

5. Resultados y pruebas

5.1. Resultados

La evaluación de las funcionalidades implementadas se realizó mediante una verificación de todos los requisitos definidos. La aplicación ha logrado implementar exitosamente el 100% de los requisitos funcionales establecidos.

Categoría	Implementados	Total	Porcentaje	Estado
RF-01: Gestión de Autenticación	5	5	100%	Completo
RF-02: Gestión de Objetivos	7	7	100%	Completo
RF-03: Visualización y Progreso	5	5	100%	Completo
RF-04: Sistema de Notificaciones	6	6	100%	Completo
RF-05: Funcionalidades de Administración	5	5	100%	Completo
Total Requisitos Funcionales	28	28	100%	Completo

Tabla 35: Verificación de Requisitos Funcionales

También se ha logrado cumplir el **100%** de los **requisitos no funcionales.** Lo que hace que el sistema tenga las características de calidad que se definieron.

Categoría	Implementados	Total	Porcentaje	Estado

RNF-01: Rendimiento	4	4	100%	Completo
RNF-02: Seguridad	4	4	100%	Completo
RNF-03: Usabilidad	5	5	100%	Completo
RNF-04: Disponibilidad	1	1	100%	Completo
RNF-05: Escalabilidad	3	3	100%	Completo
RNF-06: Mantenibilidad	3	3	100%	Completo
RNF-07: Compatibilidad	2	2	100%	Completo
Total Requisitos No Funcionales	22	22	100%	Completo

Tabla 36: Verificación de Requisitos No Funcionales

5.2. Demostración visual

Se van a incluir algunas capturas de pantalla para demostrar visualmente algunas de las funcionalidades:

RF-01: Sistema de Autenticación

Las capturas de pantalla muestran las páginas de iniciar sesión y registrar, además del mensaje de error cuando se intenta iniciar sesión sin verificar el correo electrónico.

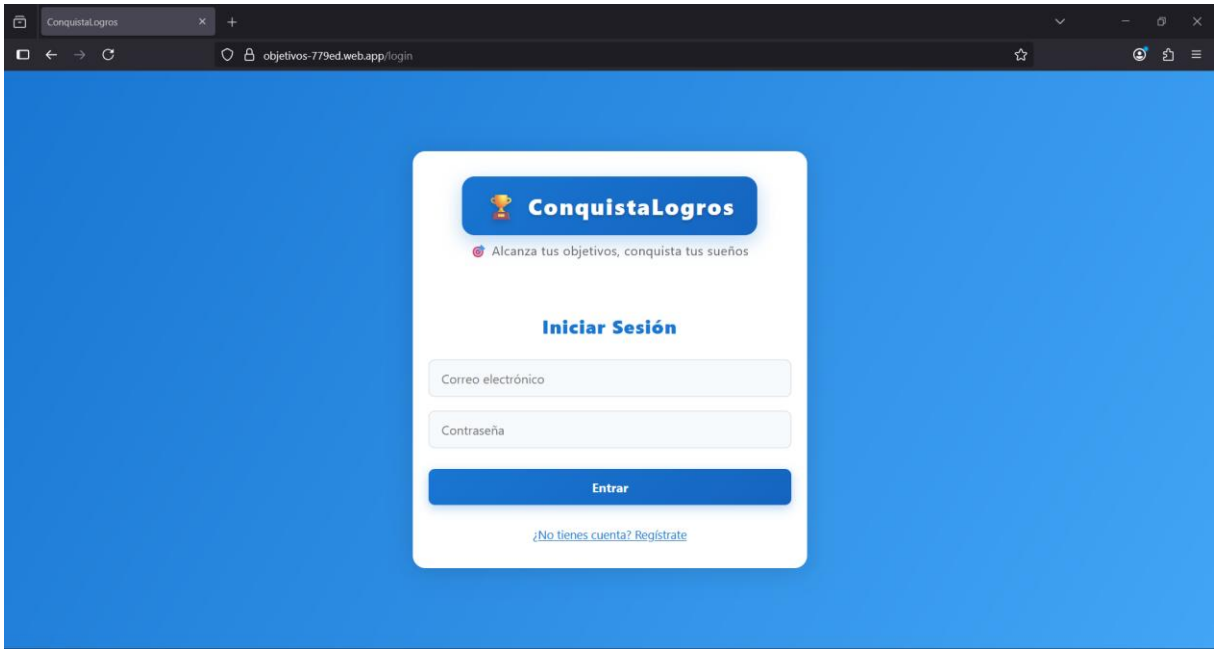

Figura 27: Pantalla de Login

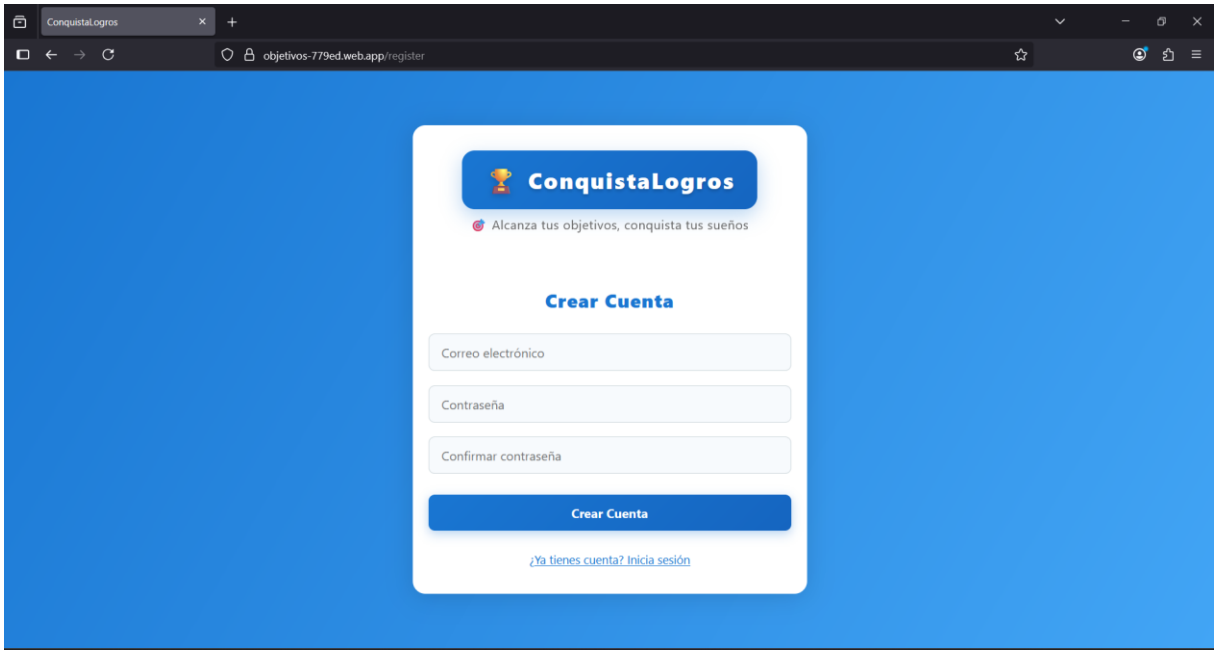

Figura 28: Pantalla de Registro

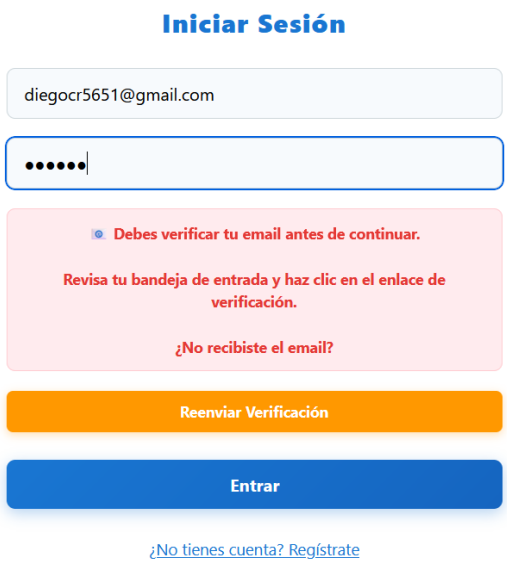

Figura 29: Mensaje de verificación de email

RF-02 y RF-03: Gestión de Objetivos y Visualización

Las capturas de pantalla muestran el dashboard general, el formulario para crear objetivos, los datos detallados de un objetivo (incluyendo la barra de progreso de hitos, la barra de progreso temporal, los hitos individuales marcados o desmarcados y los comentarios), así como un ejemplo de la personalización del dashboard, donde se muestra únicamente el gráfico de objetivos completados.

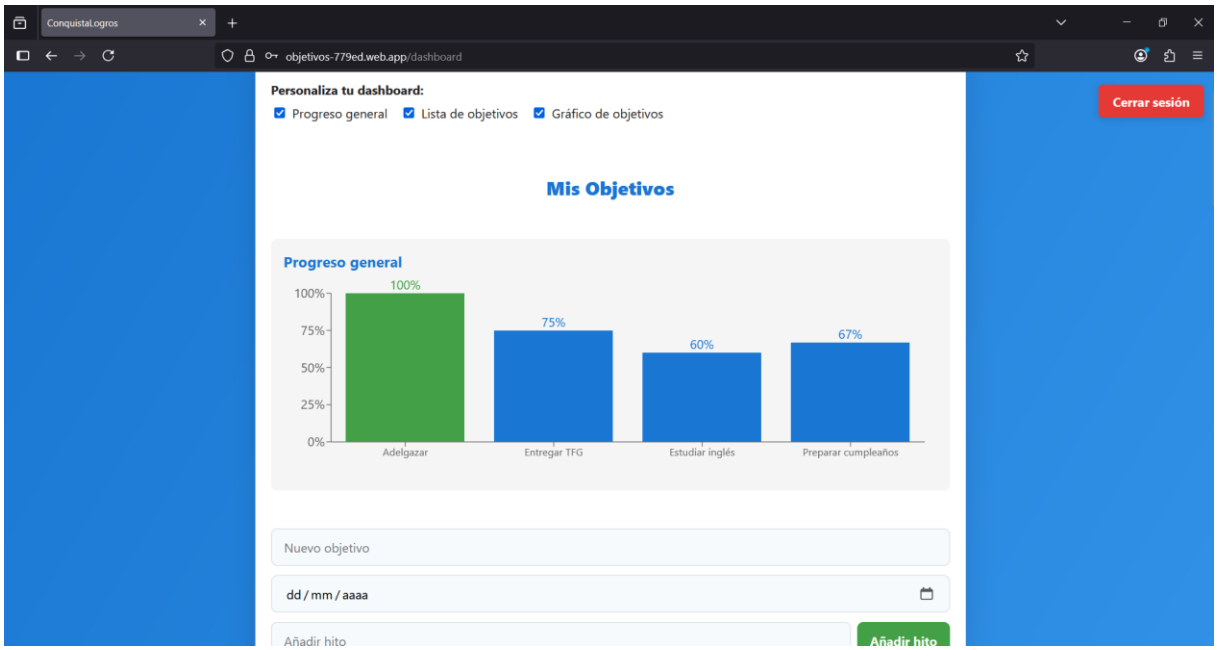

Figura 30: Dashboard principal

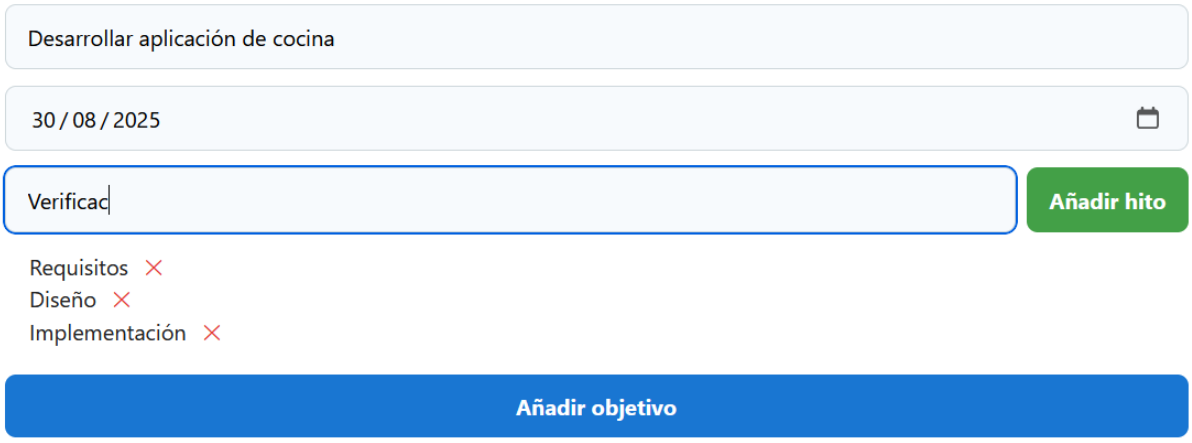

Figura 31: Formulario de creación de objetivos

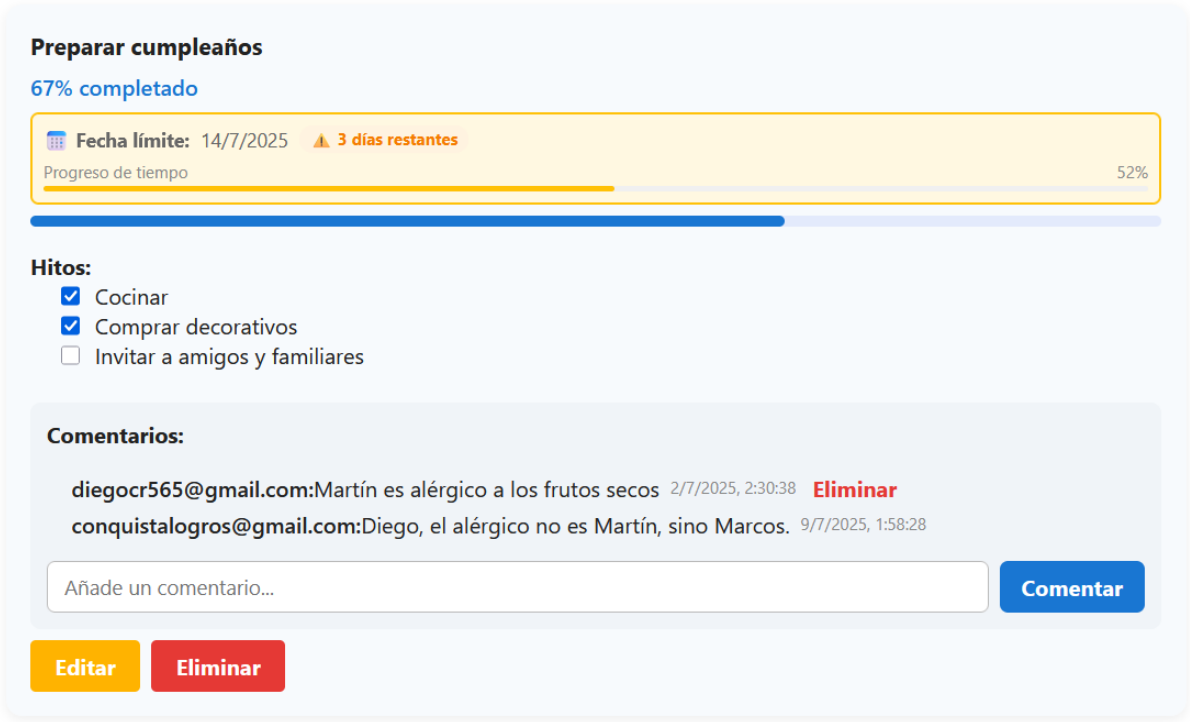

Figura 32: Objetivo

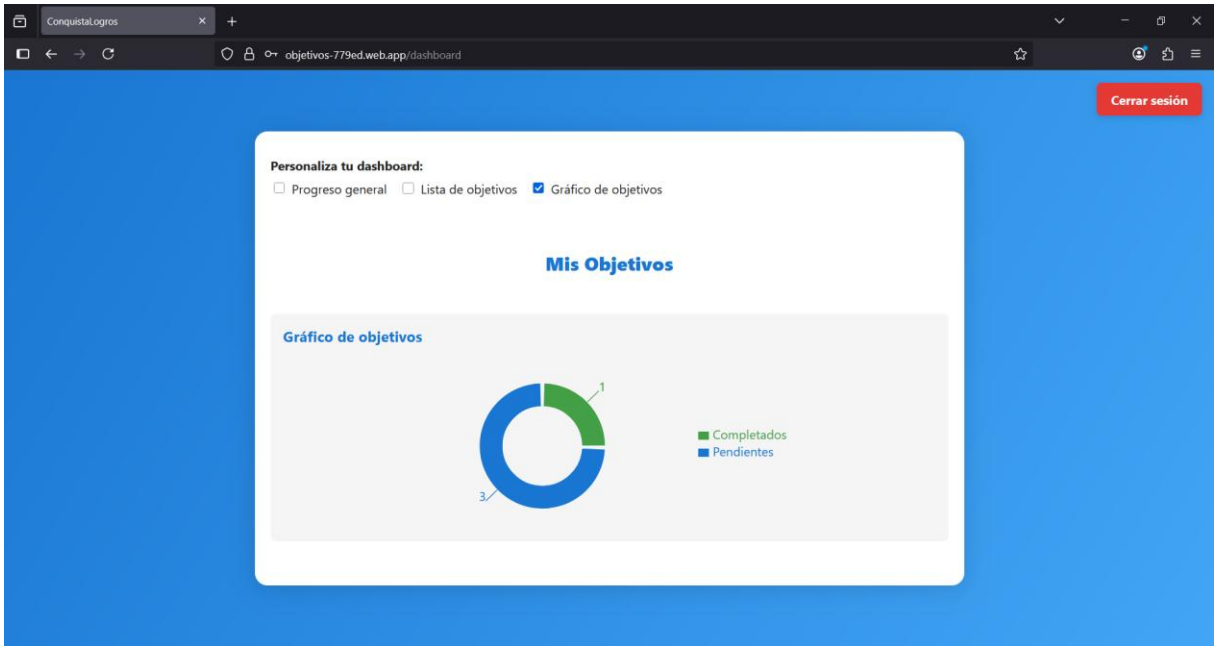

Figura 33: Personalización del dashboard (solo mostrar gráfico de objetivos)

RF-04: Sistema de Notificaciones

Las capturas de pantalla muestran algunas notificaciones vía email automáticas.

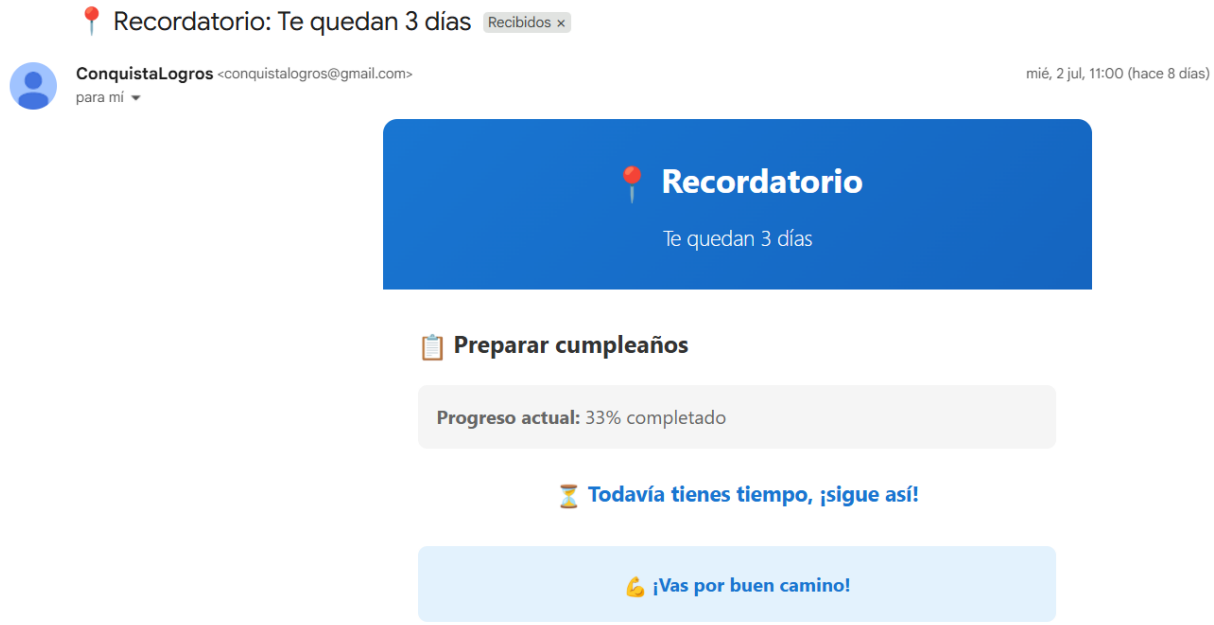

Figura 34: Correo de recordatorio

Figura 35: Correo de felicitación

RF-05: Funcionalidades de Administración

La captura de pantalla muestra el panel de administración, donde se puede encontrar a cualquier usuario del sistema y realizar operaciones CRUD sobre sus objetivos.

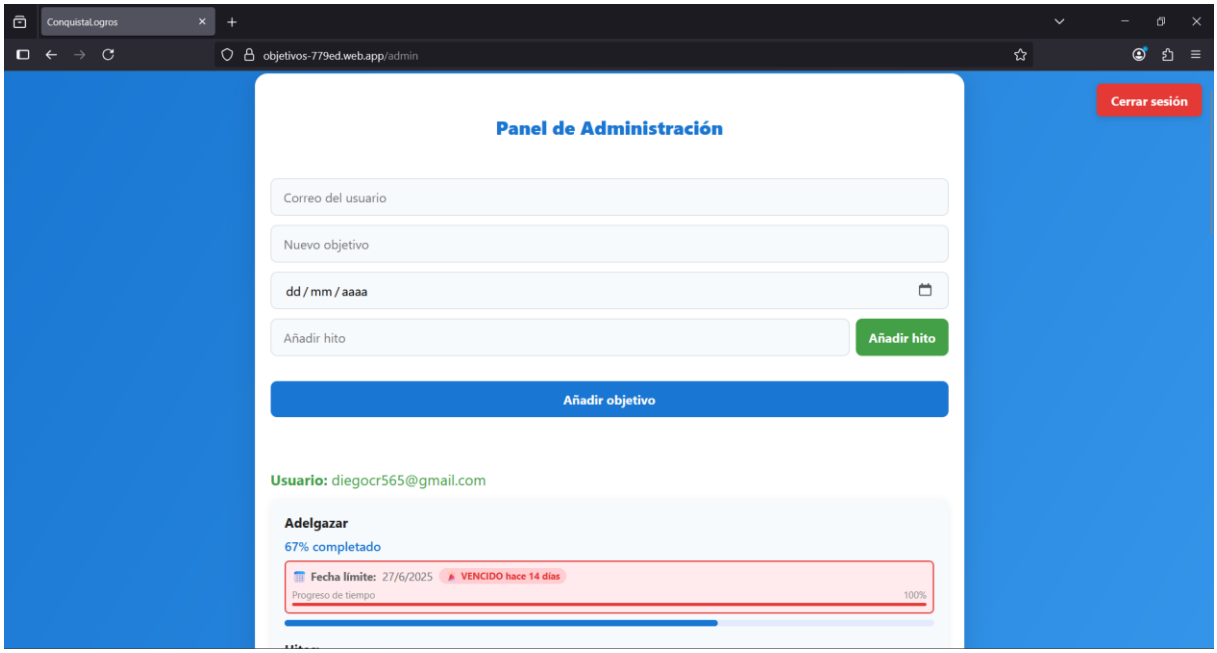

Figura 36: Panel de administración

RNF-03: Usabilidad (Responsive Design)

Las capturas de pantalla se realizaron tanto en un escritorio como en un móvil, demostrando así que el diseño es responsive y se adapta al dispositivo y tamaño de la pantalla.

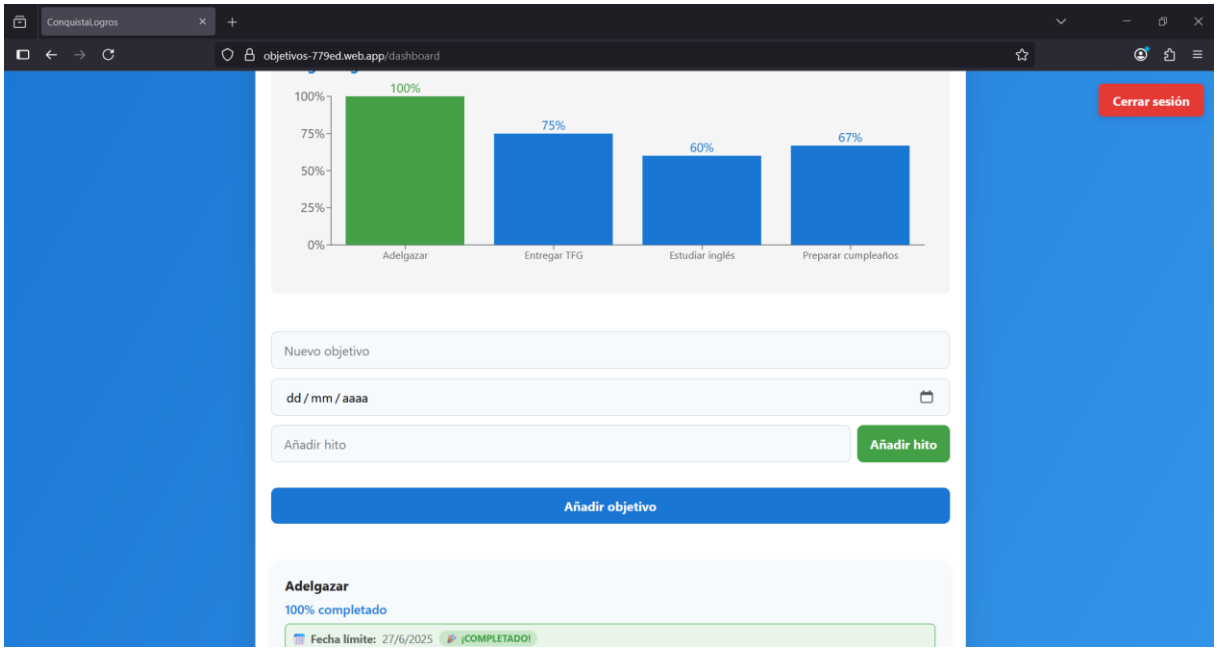

Figura 37: Vista escritorio

Figura 38: Vista móvil

6. Conclusiones y trabajo futuro

En este último apartado de la memoria se exponen las conclusiones académicas, técnicas y personales obtenidas a lo largo del desarrollo de este proyecto. Además, se analiza el impacto ético, legal y medioambiental del mismo. Por último, se proponen algunas posibles mejoras futuras en caso de que se decida extender la funcionalidad de la página web.

6.1. Conclusiones

El desarrollo de esta aplicación web ha cumplido exitosamente con el propósito fundamental de crear una plataforma web integral para la gestión de objetivos personales. La aplicación resultante ofrece una solución completa que combina funcionalidades de seguimiento, visualización y motivación, ofreciendo a los usuarios una herramienta efectiva para alcanzar sus metas.

Los objetivos específicos planteados inicialmente se han materializado de manera satisfactoria. El sistema implementa un conjunto de funcionalidades que incluyen autenticación segura con verificación por email, gestión completa de objetivos, visualización mediante gráficos interactivos, notificaciones automatizadas por correo electrónico y funcionalidades administrativas básicas. La arquitectura elegida, basada en React.js para frontend y Firebase como backend, ha demostrado ser una decisión acertada y moderna.

6.1.1. Conclusiones académicas

Desde el punto de vista académico, este proyecto ha sido una oportunidad excepcional para integrar y aplicar de manera práctica los conocimientos adquiridos a lo largo del Grado en Ingeniería del Software, en concreto, en asignaturas como Fundamentos de Ingeniería del Software, Fundamentos de Programación, Ingeniería de Requisitos y Modelado, Fundamentos de Seguridad, Bases de Datos Avanzadas y Construcción y Diseño de Interfaces Gráficas de usuario.

También se aplicaron conocimientos adquiridos en el curso MOOC (Massive Online Open Courses) sobre React de la UPM. Se aplicaron patrones de diseño modernos, como la arquitectura basada en componentes y el uso de hooks personalizados, lo que profundiza la comprensión de principios de desarrollo de software avanzados.

La aplicación de una metodología en cascada ha proporcionado estructura y claridad al proceso de desarrollo, permitiendo una progresión ordenada desde el análisis de requisitos hasta la implementación final.

6.1.2. Conclusiones técnicas

La elección de Firebase como plataforma backend ha sido una decisión técnica acertada. Su ecosistema integrado de servicios ha permitido implementar funcionalidades complejas como autenticación, base de datos en tiempo real y computación serverless. Las Firebase Functions han demostrado ser especialmente útiles para automatizar procesos como el sistema de notificaciones.

React.js, combinado con Vite como herramienta de desarrollo, ha facilitado la creación de una interfaz de usuario moderna y reactiva. La implementación de hooks personalizados permitió tener un código más mantenible y reutilizable. La integración de librerías especializadas como Recharts ha permitido crear gráficos con un esfuerzo mínimo de desarrollo.

El diseño de la base de datos en Firestore, orientado a documentos NoSQL, ha proporcionado flexibilidad para evolucionar el modelo de datos durante el desarrollo.

6.1.3. Conclusiones personales

El desarrollo de esta aplicación ha constituido un reto personal significativo que ha impulsado el crecimiento tanto técnico como profesional. El enfrentarse por primera vez a tecnologías como React.js y Firebase ha supuesto una curva de aprendizaje empinada pero muy enriquecedora, requiriendo desarrollar competencias de aprendizaje autodidacta y resolución de problemas.

La experiencia de partir desde cero con React ha sido especialmente formativa, desde comprender conceptos fundamentales como componentes y estado, hasta dominar aspectos avanzados como hooks personalizados y gestión de efectos. Similarmente, Firebase fue un paradigma completamente nuevo, desde conceptos tradicionales de backend hacia una arquitectura serverless que integra diversos servicios.

La necesidad de consultar documentación oficial, tutoriales y foros ha fortalecido la capacidad de aprendizaje continuo, una competencia vital en el campo de la ingeniería del software.

6.2. Aspectos éticos, legales y medioambientales

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

6.2.1. Aspectos éticos

1.- Inclusión digital y accesibilidad universal

La aplicación ha sido diseñada priorizando la inclusión y facilidad de uso para usuarios con diferentes niveles de competencia digital. La interfaz desarrollada con React.js implementa principios de diseño universal, garantizando que las funcionalidades sean accesibles sin necesidad de tener conocimientos técnicos especializados.

El sistema de notificaciones por email y las gráficas intuitivas aseguran que todos los usuarios puedan beneficiarse de la herramienta sin necesidad de experiencia previa en tecnologías digitales. El diseño responsive garantiza accesibilidad desde múltiples dispositivos, democratizando el acceso la gestión de objetivos.

2.- Confidencialidad y protección de datos

La protección de datos personales es un pilar fundamental del diseño ético de la aplicación. Firebase Authentication gestiona las credenciales de usuario utilizando protocolos de cifrado avanzados, eliminando la posibilidad de que las contraseñan se almacenasen en texto en claro.

La implementación de verificación obligatoria por email añade una capa adicional de seguridad, evitando que se puedan crear usuarios con correos ajenos.

3.- Transparencia

Los usuarios mantienen control total sobre sus objetivos y comentarios, con capacidad de visualización, edición y eliminación en tiempo real.

El sistema no implementa procesamiento de datos oculto ni análisis de comportamiento, manteniendo la confianza del usuario.

6.2.2. Aspectos legales

1.- Cumplimiento normativo

El proyecto respeta las licencias de software libre de todas las tecnologías empleadas, incluyendo React.js, Firebase, Recharts y dependencias del ecosistema npm. Todas las librerías operan bajo licencias permisivas que autorizan su uso.

El código fuente desarrollado es completamente original y está almacenado en un repositorio en GitHub. Las referencias bibliográficas y recursos consultados se encuentran debidamente citados en el apartado de Referencias.

2.- Derechos de propiedad intelectual

Todos los componentes, algoritmos y diseños implementados en el proyecto constituyen creación intelectual original. Los diagramas, esquemas y documentación técnica han sido desarrollados específicamente para este proyecto, asegurando su originalidad.

Diferentes logos o imágenes de aplicaciones que se han utilizado en el proyecto se han extraído de internet y no han sido modificadas.

3.- Gestión de datos y privacidad digital

El desarrollo de la aplicación ha seguido principios compatibles con normativas como el RGPD (Reglamento General de Protección de Datos).

El sistema no utiliza cookies de seguimiento ni tecnologías de rastreo, operando exclusivamente con datos esenciales para la funcionalidad de gestión de objetivos.

6.2.3. Aspectos medioambientales

1.- Optimización de recursos computacionales

La arquitectura serverless de Firebase elimina la necesidad de mantener servidores dedicados en funcionamiento continuo, reduciendo el consumo energético comparado con soluciones de hosting tradicional. Las Firebase Functions se ejecutan únicamente cuando son necesarias, optimizando el uso de recursos.

Vite proporciona builds optimizados que minimizan el tamaño de los archivos transmitidos, reduciendo el ancho de banda requerido y, en consecuencia, el consumo energético.

2.- Desarrollo sostenible y eficiente

El desarrollo local durante la fase de implementación minimizó el uso de recursos de red. La elección de tecnologías modernas como React y Firebase resulta en una aplicación más eficiente que alternativas más pesadas.

3.- Escalabilidad responsable

Firebase proporciona escalabilidad automática que ajusta los recursos según la demanda real, evitando el desperdicio energético.

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

La aplicación contribuye directamente a varios Objetivos de Desarrollo Sostenible establecidos por la Agenda 2030 de la Organización de las Naciones Unidas (ONU):

ODS 3: Salud y Bienestar

La aplicación promueve el bienestar personal mediante herramientas de gestión de objetivos que fomentan el crecimiento personal, la organización y el logro de metas. El sistema de notificaciones contribuye al desarrollo de hábitos saludables y productivos.

Figura 39: Salud y Bienestar – ODS 3

ODS 4: Educación de Calidad

Como herramienta de autogestión y desarrollo personal, la aplicación facilita el aprendizaje de competencias de planificación y organización, puede utilizarse en entornos educativos para enseñar gestión de proyectos y establecimiento de objetivos.

Figura 40: Educación de Calidad – ODS 4

ODS 9: Industria, Innovación e Infraestructura

El proyecto utiliza tecnologías y arquitecturas modernas que representan innovación en el desarrollo de aplicaciones web. La implementación de soluciones serverless contribuye al avance de infraestructuras eficientes.

Figura 41: Industria, Innovación e Infraestructura – ODS 9

ODS 12: Producción y Consumo Responsables

La eficiencia energética de la arquitectura escogida refleja un desarrollo responsable que minimiza el impacto ambiental. El diseño centrado en la funcionalidad esencial evita el aumento de consumo de recursos.

Figura 42: Producción y Consumo Responsables – ODS 12

6.3. Líneas futuras

El desarrollo de la aplicación establece una base sólida que permite múltiples mejoras. Si bien la aplicación actual cumple satisfactoriamente con los objetivos inicialmente planteados, existen oportunidades de mejorar la funcionalidad y la experiencia del usuario. A continuación, se proponen algunas mejoras:

Funcionalidad	Descripción
Inteligencia artificial para recomendaciones	Implementar algoritmos de machine learning que analicen patrones de comportamiento del usuario para sugerir

	objetivos personalizados, fechas límite optimizadas y estrategias de cumplimiento.
Colaboración social y equipos	Desarrollar funcionalidades que permitan crear grupos de trabajo donde varios usuarios puedan colaborar en objetivos compartidos.
Gamificación avanzada	Introducir elementos de juego como sistemas de puntos, logros desbloqueables y niveles de progreso para incrementar la motivación a largo plazo.
Integración con ecosistemas externos	Conectar la aplicación con herramientas como Google Calendar o Notion.
Aplicación móvil nativa	Crear aplicaciones nativas para iOS y Android utilizando React Native. Puede incluir notificaciones push, sincronización offline y widgets para la pantalla de inicio.
Asistente virtual	Integrar un chatbot que guíe a los usuarios en la definición de objetivos y responda consultas sobre mejores prácticas de gestión de metas.
Exportación y backup	Desarrollar funcionalidades de exportación de datos en múltiples formatos (PDF, Excel, JSON) y sistemas de backup automático con control de versiones.
Métricas avanzadas	Implementar análisis que muestren tendencias de productividad y correlaciones entre diferentes tipos de objetivos.
Personalización visual	Permitir temas personalizables y configuraciones de accesibilidad avanzadas para usuarios con necesidades específicas.

Tabla 37: Posibles ampliaciones futuras

Estas ideas representan oportunidades muy interesantes para transformar la aplicación en el futuro. El trabajo realizado en este proyecto establece una base tecnológica sólida que facilita la implementación progresiva de mejoras, asegurando que la aplicación sea mantenible y evolucione de forma continua.

Referencias

[1] ConquistaLogros - Aplicación web desarrollada. https://objetivos-779ed.web.app/

[2] Documentación oficial de React. https://react.dev/

[3] Vite - Herramienta de construcción frontend. https://vitejs.dev/

[4] Recharts - Librería de gráficos para React. https://recharts.org/

[5] Firebase Authentication Documentation. https://firebase.google.com/docs/auth

[6] Cloud Firestore Documentation. https://firebase.google.com/docs/firestore

[7] Firebase Functions Documentation. https://firebase.google.com/docs/functions

[8] React Router Documentation. https://reactrouter.com/

[9] Hooks en React - Documentación oficial. https://react.dev/reference/react

[10] Proyecto Fin de Grado, Computadores, ETSISI (UPM). https://computadores.etsisi.upm.es/proyecto-fin-grado/

[11] 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/

[12] Firebase Documentation. https://firebase.google.com/docs

[13] Creando una Aplicación SPA con React
https://medium.com/@diegoguevaraco/creando-una-aplicaci%C3%B3n-spa-con-react-9e570625d15f

[14] Envío de Email con Firebase Functions y Nodemailer.
https://dev.to/napsterh/envio-de-email-con-firebase-functions-y-nodemailer-79p

[15] MOOC sobre REACT en la Plataforma UPM en abierto.
https://moodle.upm.es/en-abierto/enrol/index.php?id=139

[16] Cómo integrar Firebase con React.
https://codigofacilito.com/articulos/firebase-react-reactfire

[17] Apps responsive con React.
https://programacionfacil.org/cursos/react/capitulo-8-react-responsive.html

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

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

[20] Objetivos de Desarrollo Sostenible - Naciones Unidas.

https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

[21] Repositorio GitHub del proyecto.

https://github.com/diegocasero/objetivos-web