ugr | Universidad de Granada

TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Desarrollo de una aplicación web para la gestión de finanzas personales

Autor

Noah Ramos González

Directores

Manuel Jesús Cobo Martín

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Noviembre de 2025

Desarrollo de una aplicación web para la gestión de finanzas personales

Autor

Noah Ramos González

Directores

Manuel Jesús Cobo Martín

Desarrollo de una aplicación web para la gestión de finanzas personales

Noah Ramos González

Palabras clave: Finanzas personales, Django, API Rest, React, aplicación web, bases de datos, estadísticas, Recharts, categorías, movimientos, Material UI, PostgreSQL.

Resumen

Muchas veces nos proponemos ahorrar o una mejor gestión del dinero, pero por lo general no sabemos en qué estamos gastando nuestro dinero y solemos recurrir a notas sueltas u hojas de cálculo que dejamos de usar porque se vuelven inmanejables con el tiempo.

Esta aplicación permite anotar ingresos y gastos de forma sencilla y rápida, crear tus propias categorías y consultar resúmenes que muestran de un vistazo cuánto entra, cuánto sale y cuál es el balance del periodo que interese consultar. Para facilitar el análisis, se pueden filtrar los movimientos por fechas, categoría o tipos, buscar y ordenar la información, y visualizarla en gráficos que nos ayudan a entender de manera más clara y concisa cuales son nuestras principales fuentes de ingresos y gastos, todo a través de una interfaz limpia y práctica, pensada para que cualquiera pueda usarla sin la necesidad de aprender algo nuevo.

El resultado es una herramienta útil para poner orden en la economía personal durante el día a día y tomar decisiones con más seguridad. Además, sienta una base sólida sobre la que poder añadir diversas mejoras en el futuro y hacer de esta una aplicación más completa y puntera.

Development of a Web Application for Personal Finance Management

Noah Ramos González

Keywords: Personal finances, Django, REST API, React, web application, databases, statistics, Recharts, categories, transactions, Material UI, PostgreSQL.

Abstract

We often set out to save or manage money better, but we don´t really know where our money goes. Many people rely on loose notes or spreadsheets that, over time, become hard to maintain and end up abandoned. This Final Degree Project proposes a simple solution focused on computer use: a web application to manage personal finances in a clear, everyday way.

The application lets users quickly record income and expenses, create custom categories, and view summaries that show at a glance how much comes in, how much goes out, and the balance for a given period. To make analysis easier, transactions can be filtered by date, category, or type, searched and sorted, and visualized with charts that help clarify where the money is earned and spent, all through a clean, practical interface designed to be used without any learning curve.

The result is a useful tool for bringing order to day-to-day finances and making decisions with greater confidence. It also lays a solid foundation for future enhancements that can make the application more complete and cutting-edge.

Yo, **Noah Ramos González**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 45331448N, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Noah Ramos González

Granada a 11 de noviembre de 2025.

Agradecimientos

Agradecerle a mi familia, amigos y pareja. A mis padres por darme la educación, valores y cariño que me han dado desde pequeño. También por darme la oportunidad de haber estudiado fuera de mi casa y haberme apoyado en todo momento. A mi hermana por siempre estar ahí y a pesar de haber pasado muchos años separados seguir queriéndonos mucho. A mis abuelos por haberme tratado como un hijo, sobre todo mi abuela que sé que está muy orgullosa de lo que estoy logrando y de mí como persona. A mis amigos de toda la vida, que a pesar de que tomamos rumbos diferentes siempre nos volvemos a encontrar y da igual el tiempo que haya pasado que cuando nos vemos es como si siguiéramos yendo todos los días a clase juntos. A mis amigos que he hecho durante la carrera con los que he compartido muchos momentos importantes e inolvidables de mi vida y con los que he forjado una amistad para el resto de los años. Y por último, pero no menos importante, a mi pareja que siempre ha estado a mi lado, tanto en los buenos como en los malos momentos, en etapas bonitas y en otras que no lo fueron tanto, pero que siempre estuvo, está y estará ahí. Ver como juntos hemos crecido, aprendido, querido y logrado muchas de nuestras metas no tiene precio. Y es por ello que les doy las gracias, porque sin ellos no sería lo que soy hoy, ni lo que seré mañana. Muchas gracias de corazón. Les quiero mucho.

ÍNDICE

Capítulo 1: Introducción

Este es el capítulo donde se aborda, desde una perspectiva general, el proyecto, la motivación de la realización de este, los objetivos que se pretenden cumplir y una pequeña muestra de herramientas que realicen un trabajo similar a esta propuesta en el mercado.

1.1. Motivación

Para la mayoría de las personas suele ser un quebradero de cabeza llevar el control exacto de sus finanzas, siempre hay facturas que se cobran directamente sin saber la cantidad que nos están cobrando, bizums que se envían y quedan en el olvido, etc. Por otro lado, las personas que optan por intentar llevar un control financiero de sus vidas suelen caer en el uso de hojas de cálculos que acaban siendo indescifrables o aplicaciones móviles genéricas que no terminan de satisfacer las necesidades reales del usuario, por no hablar de la poca oferta para dispositivos como los ordenadores. Es por ello por lo que este Trabajo de Fin de Grado surge para intentar satisfacer las necesidades mencionadas anteriormente, proponiendo una aplicación web ligera que permita registrar y analizar la economía personal de forma sencilla y para aquellas personas que prefieren gestionarla desde un dispositivo de sobremesa.

1.2. Objetivos

El objetivo principal que se persigue en la realización de este proyecto es desarrollar una aplicación web para la gestión de las finanzas personales que permita a los usuarios registrar ingresos y gastos, personalizar categorías a las que añadir movimientos y analizar resultados mediante KPIs y gráficos. Además del objetivo principal, se pretende lograr diferentes objetivos más específicos, que son los siguientes:

- Diseñar la arquitectura cliente-servidor, el modelo de datos y el contrato de la API REST.
- Diseñar una interfaz sencilla y accesible para todo tipo de usuarios utilizando React.
- Implementar el backend con Django y Django REST framework.
- Ofrecer un panel de estadísticas donde el usuario pueda visualizar claramente sus finanzas personales.
- Validar la aplicación mediante pruebas funcionales y no funcionales que verifique su rendimiento.

1.3. Herramientas similares

Tras revisar el mercado de aplicaciones con propósito parecido, destacamos varios referentes que cubren, con distintos enfoques, la gestión de finanzas personales:

- **YNAB (You Need A Budget)**[1]: YNAB es una plataforma centrada en presupuestación por sobres con sus reglas (“Give every dollar a job”), sincronización bancaria y análisis de hábitos de gasto. Su foco principal está en planificar el presupuesto futuro más que el registro histórico. Dispone de web y aplicación para iOS y Android, ambas son de pago.

- **Spendee**[2]: Spendee está orientada al registro de ingresos y gastos con carteras, presupuestos y gráficos. Dispone tanto de aplicación para iOS y Android como web y tiene planes gratuitos y otros de pago con funciones avanzadas como la sincronización bancaria.

- **Fintonic**[3]**:** Fintonic está centrada en agregación bancaria multientidad, categorización automática, alertas sobre comisiones y recibos y análisis financiero a través de FinScore. Además, integra comparadores de seguros, préstamos y otros productos. Dispone de aplicación para iOS y Android y de web. Es de uso gratuito.

Frente a estas alternativas, la propuesta desarrollada en este Trabajo de Fin de Grado prioriza una experiencia web de escritorio, evita la complejidad de la sincronización bancaria para favorecer el control consciente del usuario e incluye una interfaz simple y útil, filtros y ordenación de los propios ingresos y gastos y estadísticas concretas de estos con una visualización sencilla y clara.

1.4. Estructura de memoria

En este apartado se muestra cómo va a ser la estructura de la memoria del proyecto, detallando los capítulos y los contenidos que componen dichos capítulos. En cada uno de ellos se muestra un aspecto concreto del desarrollo, la implementación y evaluación del sistema, proporcionando una visión completa y organizada del trabajo realizado.

- **Capítulo 1: Introducción**
  Presenta el problema que motiva el proyecto, el propósito general y los objetivos que se persiguen. También se incorpora una breve revisión de soluciones existentes con las que se contrasta el enfoque adoptado.

[1] https://www.ynab.com/
[2] https://www.spendee.com/
[3] https://www.fintonic.com/es-ES/inicio/

- **Capítulo 2: Planificación**
  Recoge la planificación seguida, así como la metodología de gestión utilizada para controlar tiempos y objetivos.

- **Capítulo 3: Análisis del sistema**
  Especifica qué debe hacer el sistema y bajo qué condiciones. Se detallan requisitos funcionales, no funcionales, de información, las reglas de negocio y los actores implicados, además de sus casos de uso.

- **Capítulo 4: Diseño**
  Describe como se materializa la solución, es decir, la arquitectura software, modelo de datos y la interfaz. Incluye los diagramas necesarios para entender la estructura y el flujo de la aplicación.

- **Capítulo 5: Implementación**
  Explica cómo se construyó el sistema, incluyendo las tecnologías empleadas, componentes principales, decisiones técnicas relevantes y problemas encontrados durante el desarrollo, junto con su resolución.

- **Capítulo 6: Verificación y pruebas**
  Detalla las estrategias y casos de prueba utilizados para validar el sistema, cubriendo tanto requisitos funcionales como no funcionales, y los resultados obtenidos.

- **Capítulo 7: Despliegue y guía de uso**
  Ofrece una guía para desplegar la aplicación: prerrequisitos, pasos de instalación y configuración, y consideraciones para su ejecución en el entorno objetivo. También se orienta al usuario final en el manejo de la plataforma. Incluye ejemplos y capturas que facilitan la comprensión de las funcionalidades principales.

- **Capítulo 8: Conclusiones**
  Resume los logros alcanzados, evalúa el grado de cumplimiento de los objetivos y propone mejoras y extensiones que podrían abordarse en trabajos posteriores.

Capítulo 2: Planificación

En este capítulo se explica cómo se ha organizado el trabajo para desarrollar la aplicación de finanzas personales. El objetivo ha sido avanzar de forma incremental, asegurando que cada paso dejara algo funcional y estable. Para ello se ha definido una metodología sencilla, un plan con hitos claros y unos criterios de calidad que sirvieran como lista de verificación antes de dar por finalizada cada parte. También se incluye la gestión de las tareas, el control de las versiones y los principales riesgos que se han tenido en cuenta.

2.1. Metodología de trabajo

Para organizar el desarrollo se siguió Scrum en una versión ligera y adecuada a un proyecto individual. Se trabajó en iteraciones cortas, sprints de 1-2 semanas, con un objetivo claro y una versión funcional al finalizar cada iteración. Al tratarse de una única persona desarrolladora, las labores de priorización, coordinación y construcción recayeron en la misma persona.

2.1.1. Elementos que se gestionaron en cada sprint

- **Product backlog o lista general de funcionalidades**: En ella se reúne y prioriza lo que se requiere construir, nuevas pantallas, filtros, validaciones, pruebas, etc. Se revisó y ajustó de forma continua.
- **Sprint backlog o lista de trabajo del sprint:** Es la selección de tareas de la lista general que se abordó en ese sprint. Incluye el objetivo del propio sprint y las tareas desglosadas.
- **Versión funcional al cierre:** Cada sprint finalizó con una versión que corre en local y demuestra lo comprometido, por ejemplo: *API operativa, pantalla de movimientos con filtros, o panel de estadísticas* entre otros.

2.1.2. Reuniones y hábitos de trabajo

- **Planificación del sprint**: Al inicio de cada iteración se definió el objetivo y se eligieron las tareas que aportaban más valor.
- **Seguimiento breve diario (Auto-revisión):** Se registró el avance y los posibles bloqueos para ajustar el plan de cada día.
- **Revisión del sprint:** Se mostró la versión funcionando del backend y frontend en local y se recogió feedback para priorizar la siguiente iteración.
- **Retrospectiva:** Se analizaron mejoras de proceso, como, por ejemplo, limitar el trabajo simultáneo, reforzar pruebas o mensajes de error.

2.1.3. Criterios para dar por finalizada una tarea

Una tarea se consideró terminada cuando se cumplían los siguientes puntos:

- La función se encontraba operativa en local.
- Existía al menos una prueba representativa en backend o una verificación manual clara en frontend.
- Se registró un commit descriptivo.
- Se actualizó la configuración o documentación si hacía falta.

2.2. Plan de sprints y resultados

La siguiente tabla resume los sprints ejecutados y las tareas y entregables que se realizaron en cada uno de ellos.

Sprint	Objetivo principal	Tareas realizadas (resumen)	Entregables / Incremento	Observaciones
S1	Puesta en marcha del entorno	Configuración de git y repositorio, creación de entorno virtual, proyecto Django, base de datos PostgreSQL, superusuario y estructura inicial	Proyecto Django ejecutándose en local con BD PostgreSQL	Se documentó el arranque (runserver) y la activación del venv
S2	Dominio y API base	App movimientos (modelos Categoría y Movimiento), migraciones, Djando REST Framework (serializers, viewsets, router), autenticación por token y pruebas con Postman	Endpoints de categorías y movimientos con seguridad por token	Se añadió bloqueo de borrado de categoría si tiene movimientos asociados (lado backend)
S3	Frontend y login	Proyecto React, servicio de auth (almacenado de token), LoginPage, ProtectedRoute, Navbar, CategoryList (listar/crear/borrar)	Frontend mínimo funcionando contra la API (login + categorías)	Se pulió el flujo de login y redirecciones
S4	CRUD de movimientos y filtros	MovementList (listar/crear/borrar/editar), filtros por tipo, categoría y fechas, KPIs básicos (ingresos, gastos, balance), signos y colores, validaciones de cantidades positivas y	Gestión completa de movimientos desde la interfaz de usuario, con validaciones coherentes en	Se impidió borrar categorías con movimientos y se mejoraron mensajes de error

		unicidad de categoría por usuario	backend y frontend	
S5	Estadísticas y gráficos	Resumen mensual en la API, panel de estadísticas con barras (ingresos vs gastos por mes) y gráficos circulares por categoría, formato y colores consistentes (ingresos en verde y gastos en rojo)	Panel de estadísticas útil y legible	Se ajustó la leyenda y el orden visual a la convención de la aplicación (primero ingresos y luego gastos)
S6	Experiencia de uso, rendimiento y búsqueda	Responsive en filtros y formularios, paginación y ordenación en listados, búsqueda por descripción y KPIs desde el backend con los mismos filtros	Listas fluidas, estables y rápidas en diferentes tamaños de pantalla	Se actualizaron tests de API para filtros y agregados
S7	Registro y políticas de contraseña	Endpoint de registro en backend (confirmación de contraseña y política de seguridad de Django), RegisterPage, mensajes orientativos de contraseña, mejoras en navegación (HashRouter) y accesos del Navbar	Flujo completo: registro, login y uso	Se evitó diferenciar demasiado mensajes de error de login por seguridad
S8	Documentación y memoria	Redacción del documento explicativo del proyecto	Borrador consolidado de la memoria con estructura clara	No se realizaron observaciones

Tabla 2.1: Planificación de sprints

2.3. Priorización de tareas

La priorización de tareas fue iterativa, primero se abordó lo imprescindible para que el sistema fuese utilizable, como la autenticación, las categorías y los movimientos, después se añadieron mejoras de valor, como los filtros, KPIs, las estadísticas, y finalmente, el acabado y la calidad del diseño, haciendo de este un producto más completo con un diseño adaptable, incluyendo paginación, ordenación y búsqueda y haciéndolo más robusto con las validaciones y pruebas pertinentes. Cabe destacar que la tarea de despliegue del producto se reservó como una posible mejora, es por ello por lo que todo se ejecuta de manera local.

Con este enfoque, cada iteración entregó una versión utilizable y permitió validar decisiones de forma temprana, reduciendo retrabajo y asegurando que el resultado final cumpliera con los objetivos del proyecto,

Capítulo 3: Análisis del sistema

A lo largo de este capítulo se describe qué debe hacer el sistema y en qué condiciones. Para ello analizaremos los requisitos que se deben cumplir, detallando el funcionamiento de la aplicación y las reglas que aseguran su coherencia. Con ello, se pretende ofrecer una visión clara y ordenada de lo que el sistema necesita para satisfacer las necesidades del usuario.

3.1. Requisitos del sistema

Los requisitos del sistema son fundamentales para mostrar una descripción completa y detallada del comportamiento del sistema que se pretende desarrollar. Con ellos definimos el alcance, las características, las funcionalidades y las limitaciones del software para satisfacer las necesidades del usuario y los objetivos del negocio.

3.1.1. Requisitos funcionales

Los requisitos funcionales muestran de forma detallada qué es lo que debe hacer el sistema, definiendo así las funcionalidades y comportamiento de este.

RF-1	**Registro de un usuario**
Descripción	Permitir que los nuevos usuarios se registren en el sistema. Para ello, deben de aportar un usuario y contraseña válidas, pudiendo además añadir el correo electrónico de manera opcional. El usuario debe ser único en el sistema.
Prioridad	Alta.
Observaciones	El usuario debe cumplimentar correctamente los datos para formalizar el registro.

Tabla 3.1: RF-1 Registro de un usuario

RF-2	**Inicio de sesión de un usuario**
Descripción	Autenticar a un usuario mediante el nombre de usuario y contraseña.
Prioridad	Alta.
Observaciones	El usuario previamente debe de haberse registrado en el sistema.

Tabla 3.2: RF-2 Inicio de sesión de un usuario

RF-3	Cerrar sesión
Descripción	El sistema facilitará la posibilidad al usuario de que una vez este lo desee podrá finalizar la sesión. Cuando esto suceda la sesión del usuario quedará finalizada.
Prioridad	Alta.
Observaciones	El usuario previamente debe haber iniciado sesión.

Tabla 3.3: RF-3 Cerrar sesión

RF-4	Crear categoría
Descripción	Permitir al usuario crear categorías con nombre y tipo (ingreso/gasto)
Prioridad	Alta.
Observaciones	El nombre + tipo debe ser único por usuario, por lo que si se intenta crear una categoría del mismo tipo con el mismo nombre se mostrará un error.

Tabla 3.4: RF-4 Crear categoría

RF-5	Listar categorías
Descripción	Mostrar el listado de categorías, permitiendo al usuario la posibilidad de filtrar por tipo (ingresos/gastos)
Prioridad	Media.
Observaciones	El usuario debe de haber creado alguna categoría para poder filtrar.

Tabla 3.5: RF-5 Listar categoría

RF-6	Borrar categoría
Descripción	El sistema debe permitir a los usuarios borrar categorías solo si no tienen movimientos asociados. Se mostrará un mensaje para solicitar la confirmación de borrado al usuario.

Prioridad	Alta.
Observaciones	Si una categoría tiene movimientos asociados se mostrará un mensaje de error si se desea borrar indicando esto mismo.

Tabla 3.6: RF-6 Borrar categoría

RF-7	**Crear movimiento**
Descripción	El sistema debe permitir a los usuarios registrar un movimiento con categoría, fecha, cantidad y descripción de manera opcional. La cantidad debe ser positiva.
Prioridad	Alta.
Observaciones	El usuario debe de haber creado previamente alguna categoría a la que añadir el movimiento. La fecha por defecto es el día actual en la que se registra el movimiento, aunque por supuesto se puede modificar.

Tabla 3.7: RF-7 Crear movimiento

RF-8	**Listar movimientos**
Descripción	El sistema debe permitir a los usuarios la posibilidad de consultar los movimientos que ha creado
Prioridad	Alta.
Observaciones	El usuario debe tener previamente movimientos creados.

Tabla 3.8: RF-8 Listar movimientos

RF-9	**Filtrar movimientos**
Descripción	El sistema debe permitir a los usuarios filtrar los movimientos por rango de fechas (desde/hasta), tipo (ingreso/gasto) y categoría.
Prioridad	Alta.

Observaciones	El usuario debe de disponer de movimientos creados previamente para que el filtrado funcione correctamente.

Tabla 3.9: RF-9 Filtrar movimientos

RF-10	**Buscar movimiento**
Descripción	Los usuarios deben poder buscar movimientos a través de una búsqueda por texto de la descripción de dichos movimientos.
Prioridad	Media.
Observaciones	Para que la búsqueda sea exitosa es necesario que el usuario haya añadido una descripción en el momento de la creación del movimiento.

Tabla 3.10: RF-10 Buscar movimientos

RF-11	**Ordenar movimientos**
Descripción	El sistema debe permitir a los usuarios ordenar los movimientos por fecha (recientes/antiguos), cantidades (mayores/menores) o por orden alfabético (A-Z/Z-A).
Prioridad	Media.
Observaciones	El usuario debe tener creados previamente los movimientos para poder ordenarlos. Además, la ordenación se puede combinar con la búsqueda y filtrado de movimientos.

Tabla 3.11: RF-11 Ordenar movimientos

RF-12	**Editar movimiento**
Descripción	El sistema debe permitir al usuario la posibilidad de editar los campos de un movimiento ya existente.
Prioridad	Media.
Observaciones	El usuario debe haber creado previamente el movimiento que desea editar. También se aplican las mismas reglas que en Crear Movimiento.

Tabla 3.12: RF-12 Editar movimiento

RF-13	Eliminar movimiento
Descripción	El sistema debe permitir al usuario la posibilidad de eliminar un movimiento de la lista de sus movimientos.
Prioridad	Media.
Observaciones	El usuario debe haber creado previamente el movimiento que desea eliminar. Además, se le solicitará la confirmación de la eliminación de este.

Tabla 3.13: RF-13 Eliminar movimiento

RF-14	Visualizar estadísticas
Descripción	El sistema debe permitir al usuario la posibilidad de observar las estadísticas y gráficas de sus categorías y movimientos.
Prioridad	Alta.
Observaciones	El usuario debe de disponer de categorías y movimientos previamente creados para poder visualizar las estadísticas.

Tabla 3.14: RF-14 Visualizar estadísticas

RF-15	Filtrar estadísticas
Descripción	El sistema debe permitir al usuario la posibilidad de filtrar las estadísticas para los periodos de tiempo concretos que desee (mes anterior, desde/hasta, año en curso, este mes)
Prioridad	Alta.
Observaciones	El usuario debe disponer de diversos movimientos y categorías para que el filtrado sea útil.

Tabla 3.15: RF-15 Filtrar estadísticas

3.1.2. Requisitos no funcionales

Los requisitos no funcionales, también conocidos como requisitos de calidad, muestran las características y cualidades del sistema, definiendo cómo debe funcionar.

RNF-1	Seguridad
Descripción	El sistema debe proteger la información del usuario y evitar accesos indebidos, para ello: ● Autenticación: el acceso a la API se realiza mediante token, por lo que solo los usuarios identificados podrán operar con sus datos. ● Autorización: cada petición queda limitada a los recursos del usuario autentificado, no permitiendo así lecturas ni escrituras cruzadas entre cuentas. ● Protección: las contraseñas se almacenan con hash y nunca en texto plano.

Tabla 3.16: RNF-1 Seguridad

RNF-2	Usabilidad
Descripción	Se busca que la interfaz pueda utilizarse con facilidad por perfiles no técnicos, a través de: ● Interfaz clara: una navegación simple, mensajes de errores comprensibles y la consistencia visual en todos los módulos ofrecen al usuario una experiencia sencilla. ● Accesibilidad: el desarrollo de la aplicación ha sido pensado para que todo tipo de usuarios sean capaces de poder hacer un correcto uso de esta.

Tabla 3.17: RNF-2 Usabilidad

RNF-3	Rendimiento
Descripción	El sistema debe ofrecer tiempos de respuesta ágiles en operaciones comunes: ● Paginación en listados para evitar cargas innecesarias. ● Consultas eficientes en base de datos, con posibilidad de índices cuando sea pertinente. ● Tiempo de respuesta y carga óptimos para no perjudicar el rendimiento.

Tabla 3.18: RNF-3 Rendimiento

RNF-4	Mantenibilidad
Descripción	Se prioriza que el sistema sea fácil de entender y extender con: ● Estructura modular (apps Django, componentes React) para facilitar el mantenimiento y facilitar las posibles mejoras. ● Correcto y periódico uso de pruebas para comprobar el correcto funcionamiento de la aplicación. ● Documentación detallada para que cualquier persona pueda comprender el funcionamiento y desarrollo de la aplicación.

Tabla 3.19: RNF-4 Mantenibilidad

RNF-5	Fiabilidad
Descripción	Se persigue mantener la integridad de la información y la continuidad del servicio a pesar de la posible existencia de fallos que puedan surgir.

Tabla 3.20: RNF-5 Fiabilidad

RNF-6	Portabilidad
Descripción	La aplicación debe funcionar en diferentes entornos y dispositivos. Para ello, el servidor debe ser compatible para los diferentes sistemas, asegurar un correcto funcionamiento en los diferentes navegadores y asegurar un despliegue óptimo en los diferentes dispositivos.

Tabla 3.21: RNF-6 Portabilidad

3.1.3. Requisitos de información

Los requisitos de información son aquellos que nos indican la información que se va a almacenar en el sistema para asegurar el correcto funcionamiento de este.

RI-1	Usuario
Contenido	El sistema con respecto al usuario debe almacenar el identificador, nombre de usuario y contraseña.

Tabla 3.22: RI-1 Usuario

RI-2	Categoría
Contenido	El sistema almacena el identificador, usuario propietario, nombre y tipo. No se permiten duplicados por usuario, nombre y tipo.

Tabla 3.23: RI-2 Categoría

RI-3	Movimiento
Contenido	El sistema almacena el identificador, usuario propietario, categoría, fecha, cantidad y la descripción (opcional). La cantidad del movimiento obligatoriamente debe ser positiva y no es posible asociar movimientos a categorías de otros usuarios

Tabla 3.24: RI-3 Movimiento

3.1.4. Reglas de negocio

Las reglas de negocio nos indican las políticas y restricciones que gobiernan el funcionamiento de la aplicación. Con ellas se garantiza la coherencia e integridad de los datos y se evitan usos no válidos del sistema.

RN-1	Propiedad individual de categoría
Descripción	Una categoría pertenece exclusivamente a una única persona.

Tabla 3.25: RN-1 Propiedad individual de categoría

RN-2	Unicidad de categoría
Descripción	Para un mismo usuario no pueden existir dos categorías con el mismo nombre y tipo, independientemente de si está escrito en mayúsculas o minúsculas.

Tabla 3.26: RN-2 Unicidad de categoría

RN-3	**Importe positivo obligatorio**
Descripción	Todo movimiento debe tener una cantidad estrictamente mayor que cero, independientemente si se trata de un ingreso o un gasto.

Tabla 3.27: RN-3 Importe positivo obligatorio

RN-4	**Protección de borrado con dependencias**
Descripción	No se puede eliminar una categoría que disponga de movimientos asociados. En el momento del borrado el sistema informará mediante mensaje que la categoría contiene movimientos asociados.

Tabla 3.28: RN-4 Protección de borrado con dependencias

RN-5	**Estadísticas dependientes de filtros**
Descripción	Los totales y gráficos se calculan según los filtros activos, por lo que, al cambiar los filtros cambian los resultados

Tabla 3.29: RN-5 Estadísticas dependientes de filtros

RN-6	**Aislamiento por usuario**
Descripción	Solo se muestran y operan los datos del usuario que ha iniciado sesión

Tabla 3.30: RN-6 Aislamiento por usuario

3.2. Casos de uso

En esta sección se describe como interactúa el usuario, único actor del sistema, con la aplicación para alcanzar determinados objetivos a través de los casos de uso. Estos recogen, desde la perspectiva del usuario, las principales acciones que realiza y que a su vez están directamente relacionados con los requisitos funcionales definidos previamente. Al tratarse de un único actor que interactúa con el sistema, en el siguiente diagrama veremos como todas las interacciones con los casos de uso son por parte del usuario, mientras que el sistema es el encargado de proporcionar los mecanismos y una experiencia de uso adecuada para que dichas tareas sean completadas de forma clara y eficiente.

3.2.1. Diagrama de casos de uso

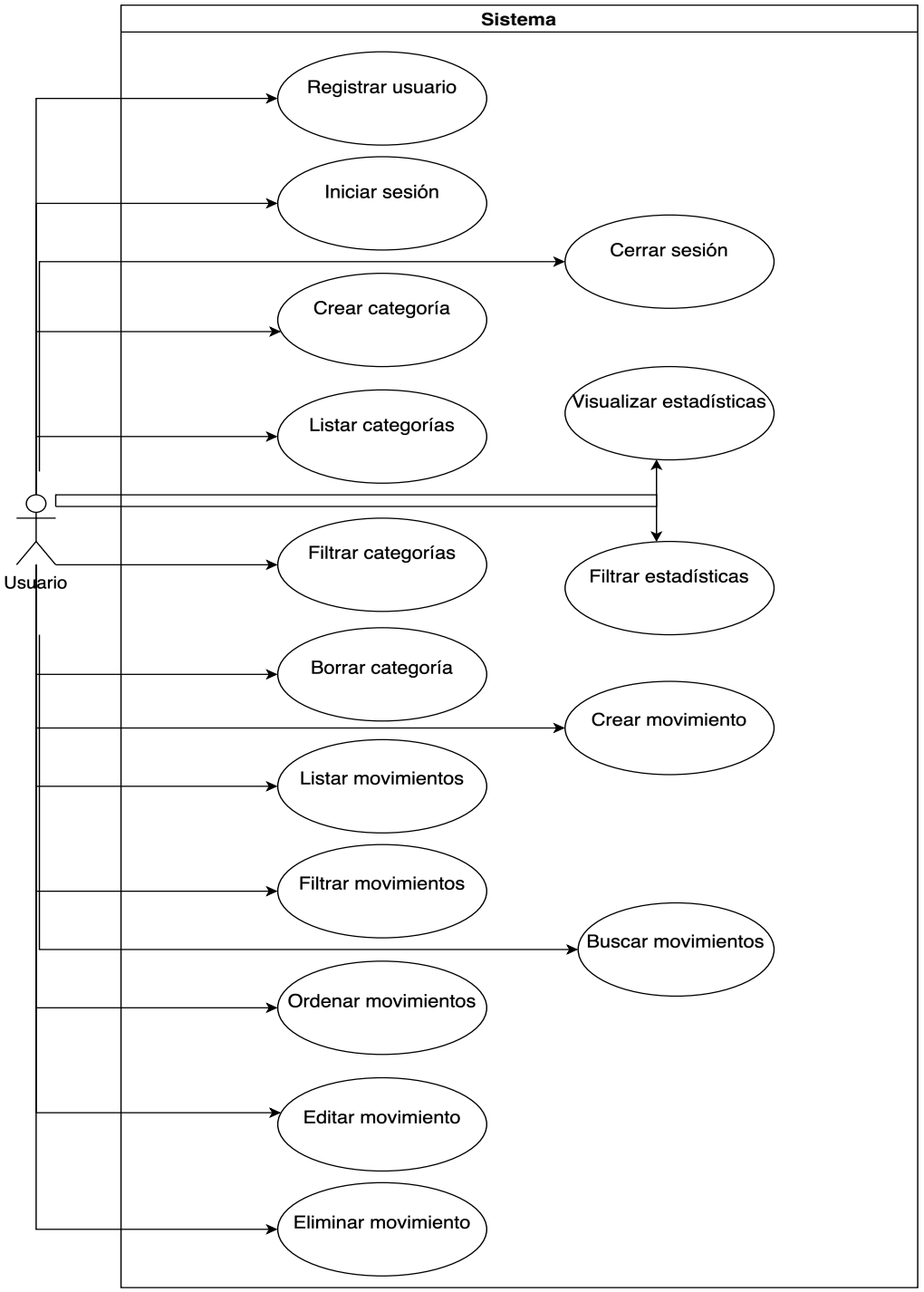


Figura 3.1: Diagrama de casos de uso

3.2.2. Descripción de los casos de uso

CU-1	Registrar usuario
Descripción	Dar de alta a un nuevo usuario en el sistema.
Actores	Usuario
Referencias	RF-1
Precondiciones	No hay precondiciones.
Postcondiciones	El usuario queda registrado en el sistema.
Flujo normal	1. Usuario: selecciona crear cuenta. 2. Usuario: introduce el usuario, contraseña y de manera opcional el email. 3. Sistema: valida los datos y la unicidad de usuario. 4. Sistema: crea el usuario.
Flujo alternativo	4.b. Datos inválidos o usuario ya existente → Sistema: informa del error y no realiza el registro

Tabla 3.31: CU-1 Registrar usuario

CU-2	Iniciar sesión
Descripción	Autenticar a un usuario existente.
Actores	Usuario
Referencias	RF-2
Precondiciones	El usuario debe existir y conocer sus credenciales.
Postcondiciones	Sesión iniciada (token disponible en el cliente).
Flujo normal	1. Usuario: introduce usuario y contraseña. 2. Sistema: valida las credenciales. 3. Sistema: genera token y habilita el acceso al usuario.
Flujo alternativo	3.b. Credenciales incorrectas → Sistema: informa del error y no inicia sesión.

Tabla 3.32: CU-2 Iniciar sesión

CU-3	Cerrar sesión
Descripción	Finalizar la sesión de un usuario autenticado.
Actores	Usuario
Referencias	RF-3
Precondiciones	El usuario debe haber iniciado sesión.
Postcondiciones	Se invalida el token almacenado en el cliente y se finaliza la sesión.
Flujo normal	1. Usuario: solicita cerrar sesión. 2. Sistema: elimina token del cliente y redirige al usuario a inicio de sesión.
Flujo alternativo	

Tabla 3.33: CU-3 Cerrar sesión

CU-4	Crear categoría
Descripción	Crear una categoría con nombre y tipo (ingreso/gasto).
Actores	Usuario
Referencias	RF-4
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	La categoría queda registrada y visible en el listado.
Flujo normal	1. Usuario: introduce nombre y tipo. 2. Sistema: comprueba unicidad de la categoría (nombre y tipo). 3. Sistema: guarda la categoría.
Flujo alternativo	3.b. Categoría inválida o duplicada → Sistema: informa del error y no crea la categoría.

Tabla 3.34: CU-4 Crear categoría

CU-5	Listar categorías
Descripción	Mostrar el listado de categorías al usuario.
Actores	Usuario

Referencias	RF-5
Precondiciones	El usuario debe estar autenticado.
Postcondiciones	Las categorías se muestran en pantalla.
Flujo normal	1. Usuario: accede al apartado de *Categorías*. 2. Sistema: recupera y muestra la lista de las categorías.
Flujo alternativo	2.b. No hay categorías → Sistema: muestra la lista vacía y opción de crear categorías.

Tabla 3.35: CU-5 Listar Categorías

CU-6	**Filtrar categoría**
Descripción	Filtrar categoría por tipo (ingreso/gasto).
Actores	Usuario
Referencias	RF-5
Precondiciones	El usuario debe haber iniciado sesión y tener categorías disponibles.
Postcondiciones	Se muestra el listado filtrado por la categoría seleccionada.
Flujo normal	1. Usuario: selecciona el tipo por el que desea filtrar. 2. Sistema: aplica el filtro y muestra la lista filtrada.
Flujo alternativo	2.b. No hay coincidencias → Sistema: muestra la lista vacía.

Tabla 3.36: CU-6 Filtrar categoría

CU-7	**Borrar categoría**
Descripción	Eliminar una categoría que no contenga movimientos.
Actores	Usuario
Referencias	RF-6
Precondiciones	El usuario debe estar autenticado y la categoría no debe tener movimientos.

Postcondiciones	La categoría eliminada y desaparece del listado.
Flujo normal	1. Usuario: pulsa el botón de eliminar la categoría. 2. Sistema: solicita confirmación para eliminar. 3. Usuario: confirma la eliminación. 4. Sistema: verifica que la categoría no contenga movimientos. 5. Sistema: elimina la categoría del listado.
Flujo alternativo	5.b. Categoría con movimientos asociados → Sistema: informa del error y no borra la categoría.

Tabla 3.37: CU-7 Borrar categoría

CU-8	**Crear movimiento**
Descripción	Registrar un movimiento con categoría, fecha, cantidad (>0) y descripción (opcional).
Actores	Usuario
Referencias	RF-7, CU-4
Precondiciones	El usuario debe estar autenticado y debe existir al menos una categoría.
Postcondiciones	El movimiento queda guardado y visible en el listado. Los KPIs se actualizan.
Flujo normal	1. Usuario: introduce datos para registrar el movimiento. 2. Sistema: valida que los datos introducidos son correctos. 3. Sistema: guarda el movimiento, actualiza la lista y los KPIs.
Flujo alternativo	3.b. Datos inválidos → Sistema: informa del error y no crea el movimiento.

Tabla 3.38: CU-8 Crear movimiento

CU-9	**Listar movimientos**
Descripción	Mostrar los movimientos del usuario (con paginación).
Actores	Usuario
Referencias	RF-8
Precondiciones	El usuario debe estar autenticado y debe existir al menos un movimiento.

Postcondiciones	Se muestran los movimientos de la página seleccionada
Flujo normal	1. Usuario: accede al apartado de *Movimientos*. 2. Sistema: obtiene y muestra la lista paginada.
Flujo alternativo	2.b. No hay movimientos → Sistema: muestra la lista de movimientos vacía.

Tabla 3.39: CU-9 Listar movimientos

CU-10	**Filtrar movimientos**
Descripción	Filtrar movimientos por rango de fechas, tipo y/o categoría.
Actores	Usuario
Referencias	RF-9
Precondiciones	El usuario debe estar autenticado y existir movimientos para filtrar.
Postcondiciones	Se muestra la lista que cumple con los filtros aplicados.
Flujo normal	1. Usuario: define los filtros a aplicar. 2. Sistema: aplica los filtros y actualiza el listado y los KPIs. 3. Sistema: muestra la lista filtrada y los KPIs actualizados.
Flujo alternativo	3.b. Sin resultados → Sistema: muestra la lista vacía.

Tabla 3.40: CU-10 Filtrar movimientos

CU-11	**Buscar movimientos**
Descripción	Localizar movimientos por texto en la descripción.
Actores	Usuario
Referencias	RF-10
Precondiciones	El usuario debe estar autenticado y existir movimientos con descripciones.
Postcondiciones	Se muestran los movimientos que coinciden con la búsqueda.

Flujo normal	1. Usuario: escribe término de búsqueda. 2. Sistema: filtra por descripción. 3. Sistema: muestra los resultados que coinciden con la búsqueda.
Flujo alternativo	3.b. Sin coincidencias → Sistema: muestra la lista vacía.

Tabla 3.41: CU-11 Buscar movimiento

CU-12	**Ordenar movimientos**
Descripción	Ordenar movimientos por fechas, cantidades o descripción
Actores	Usuario
Referencias	RF-11
Precondiciones	El usuario debe estar autenticado y existir movimientos para ordenar.
Postcondiciones	Se muestra el listado reordenado.
Flujo normal	1. Usuario: selecciona el criterio de ordenación. 2. Sistema: aplica el orden y actualiza la vista.
Flujo alternativo	

Tabla 3.42: CU-12 Ordenar movimientos

CU-13	**Editar movimiento**
Descripción	Modificar los campos de un movimiento ya registrado.
Actores	Usuario
Referencias	RF-10
Precondiciones	El usuario debe estar autenticado y existir el movimiento a editar.
Postcondiciones	Se actualiza el movimiento y los KPIs.
Flujo normal	1. Usuario: pulsa sobre el botón de editar movimiento. 2. Usuario: modifica el movimiento. 3. Sistema: valida que los datos introducidos son correctos. 4. Sistema: actualiza el movimiento y los KPIs.
Flujo alternativo	4.b. Datos inválidos → Sistema: muestra error y no actualiza el movimiento.

Tabla 3.43: CU-13 Editar movimiento

CU-14	Eliminar movimiento
Descripción	Borrar un movimiento existente.
Actores	Usuario
Referencias	RF-13
Precondiciones	El usuario debe estar autenticado y existir movimientos.
Postcondiciones	Se borra el movimiento y se recalculan los KPIs.
Flujo normal	1. Usuario: pulsa sobre el botón de eliminar movimiento. 2. Sistema: solicita al usuario que confirme la operación de borrado. 3. Usuario: confirma la eliminación del movimiento. 4. Sistema: elimina el movimiento y actualiza el listado y los KPIs.
Flujo alternativo	3.b. Usuario cancela la eliminación → Sistema: no elimina el movimiento.

Tabla 3.44: CU-14 Eliminar movimiento

CU-15	Visualizar estadísticas
Descripción	Consultar gráficos y resúmenes estadísticos.
Actores	Usuario
Referencias	RF-14, CU-9
Precondiciones	El usuario debe haber iniciado sesión y disponer de categorías y movimientos.
Postcondiciones	Se muestran los gráficos y resúmenes de las diferentes categorías y movimientos.
Flujo normal	1. Usuario: accede al apartado de *Estadísticas*. 2. Sistema: carga los datos y renderiza los gráficos.
Flujo alternativo	2.b. No existen categorías/movimientos → Sistema: no muestra los gráficos.

Tabla 3.45: CU-15 Visualizar estadísticas

CU-16	Filtrar estadísticas
Descripción	Ajustar el periodo que se desea visualizar
Actores	Usuario
Referencias	RF-15, CU-15
Precondiciones	El usuario debe estar autenticado y existir datos para el periodo seleccionado.
Postcondiciones	Se muestran los gráficos y KPIs del periodo seleccionado
Flujo normal	1. Usuario: selecciona atajo o define las fechas por las que desea filtrar. 2. Sistema: aplica el filtro y muestra los gráficos para el periodo seleccionado.
Flujo alternativo	2.b. Sin datos en el periodo → Sistema: aplica el filtro y muestra los gráficos vacíos.

Tabla 3.46: CU-16 Filtrar estadísticas

3.3. Modelo conceptual de datos

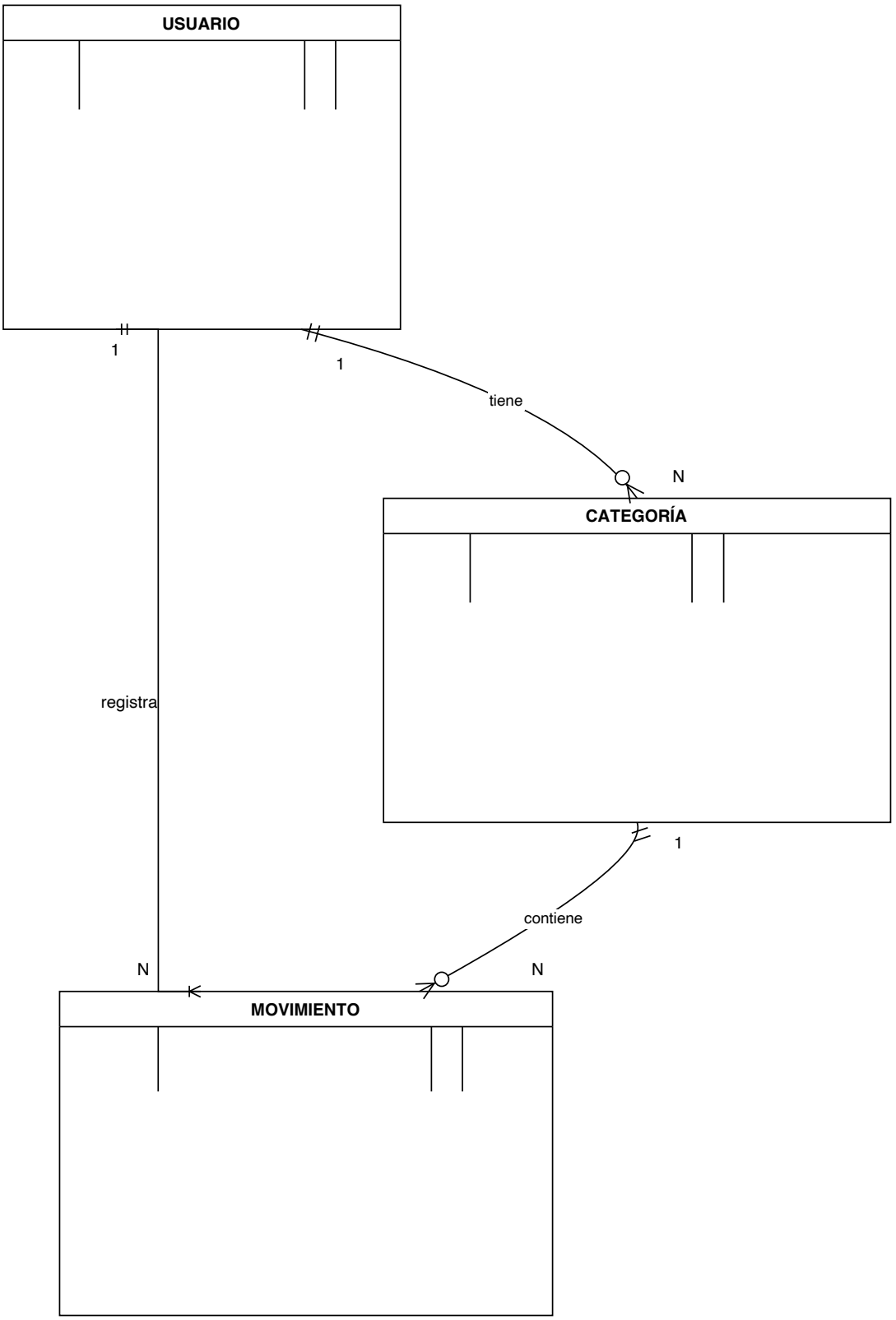


Figura 3.2: Modelo conceptual de datos

Tal y como se observa en la figura anterior, se muestra el modelo conceptual de datos, el cual nos muestra las principales entidades y sus respectivas relaciones. Las tres entidades definidas son **Usuario**, **Categoría** y **Movimiento**. Además de profundizar en estas entidades y como se relacionan veremos un conjunto reducido de reglas de integridad que garantizan coherencia y calidad de los datos.

Usuario representa a la persona que interactúa con el sistema. A nivel conceptual esta entidad incluye un identificador y los datos básicos que el usuario introduce a la hora de darse de alta en el sistema, siendo el email un dato opcional. En la implementación, esta entidad se apoya en el sistema de usuarios que nos facilita Django. Un usuario puede crear varías categorías y diversos movimientos.

La entidad **Categoría** es la encargada de agrupar los movimientos bajo un nombre y un tipo (ingreso/gasto). Cada categoría pertenece exactamente a un usuario, tal y como se observa en la relación entre ambas entidades en el diagrama, siendo la relación 1-N desde el usuario. Para que esto se cumpla se impone una restricción de unicidad por usuario sobre el par nombre y tipo, lo que provoca que un usuario no pueda crear dos categorías con el mismo nombre y que estas a su vez sean del mismo tipo. Esto no evita que otro usuario sí pueda tener su propio conjunto de categoría con los mismos nombres.

Movimiento recoge cada transacción que el usuario realiza en su día a día. Cada movimiento es registrado con su respectiva fecha, cantidad y descripción (opcional) y siendo referenciado a una categoría concreta. Un movimiento siempre pertenece a un usuario y a una categoría, siendo ambas relaciones 1-N. Además, sobre la cantidad del movimiento se establece una regla estricta, la cual establece que esta debe ser positiva, el signo lo aporta la categoría a la que se asocia el movimiento (ingreso/gasto). Esta decisión evita errores con los signos, simplifica la interfaz y hace más eficiente el cálculo de los balances.

Desde el punto de vista de integridad referencial, se ha optado por proteger el borrado de categorías que tengan movimientos asociados, teniendo que reubicar o eliminar los movimientos de esa categoría antes de proceder con el borrado de esta. Los datos como balance o totales por mes no están asociados a ninguna entidad ya que estos se calculan bajo demanda en la capa de servicios para evitar inconsistencias y minimizar errores. Como consecuencia, esto ha permitido que se haya construido una interfaz sencilla y una API coherente, manteniendo la posibilidad al mismo tiempo de evolución futura sin necesidad de reestructurar el núcleo de datos.

Capítulo 4: Diseño

Este capítulo describe el desarrollo del sistema diseñado. En primer lugar, se presenta la arquitectura general y la justificación de las decisiones tomadas. Posteriormente, se detalla la organización por capas, el modelo lógico de datos derivado del modelo conceptual y el contrato de la API expuesta. Finalmente, en esta sección se recoge el diseño de la interfaz de usuario, la gestión del estado y los aspectos referidos a la seguridad y validación que garantizan el buen funcionamiento y la coherencia en el sistema.

4.1. Visión general de la arquitectura

El sistema por el que se ha optado ha sido por una arquitectura cliente-servidor con una aplicación de una sola página (SPA) en la parte del cliente y un servidor web que expone una API REST. En la parte del navegador se ejecuta el cliente, la cual ha sido desarrollada con React, el cual es responsable de la parte de la navegación, la presentación y la interacción con el propio usuario. Por la parte del servidor se ejecuta Django con Django REST Framework, encargado de la lógica de negocio, el acceso a datos y la autenticación mediante tokens. Además, la persistencia se realiza con PostgreSQL.

Esta separación permite modificar y evolucionar la interfaz sin afectar al backend y viceversa, lo cual permite reutilizar la API desde otros clientes en el futuro (por ejemplo, una aplicación móvil) y mantener las responsabilidades claras, por una parte, teniendo el cliente que muestra la parte de la interfaz con la que el usuario interactúa, y por la otra, el servidor que se encarga de validar, calcular y garantizar la consistencia de los datos.

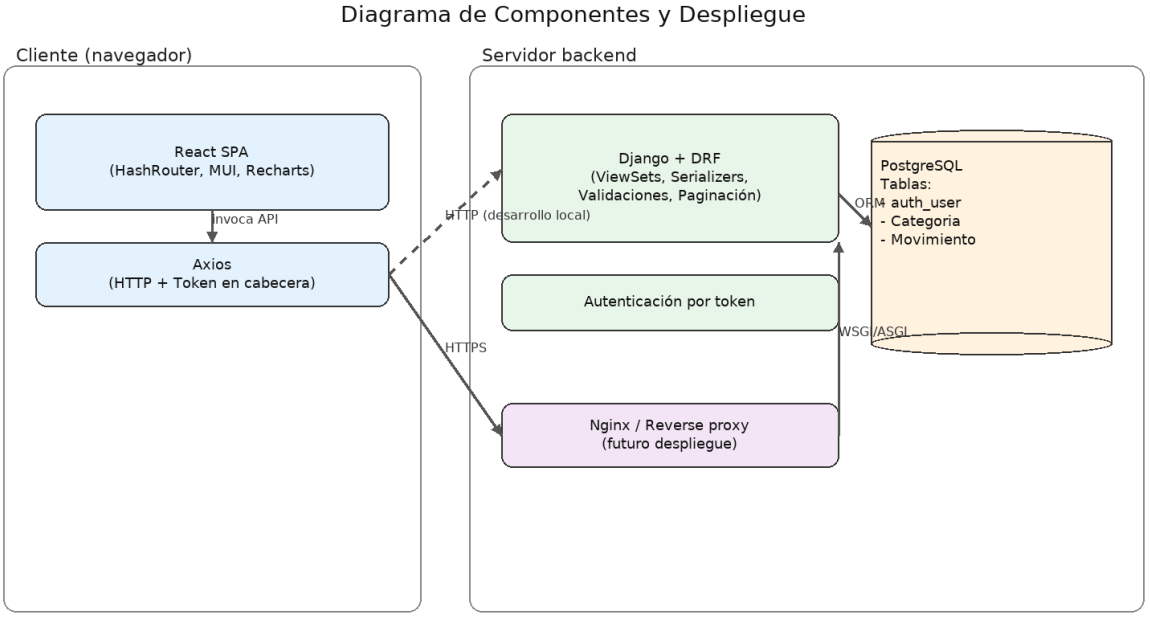


Figura 4.1: Diagrama de componentes

4.2. Organización por capas

A pesar de que el backend ha sido desarrollado como un proyecto "monolítico" en Django, el sistema se estructura internamente en capas con responsabilidades claras, lo que simplifica su mantenimiento y crecimiento.

En la capa de presentación, el cliente web creado con React se encarga de toda la interacción con el usuario, administra la navegación entre diferentes vistas (inicio de sesión, registro, categorías, movimientos y estadísticas), presenta formularios y listas, y realiza validaciones iniciales para optimizar la experiencia del usuario. Se utiliza Material UI para la interfaz, que proporciona componentes accesibles y responsivos, y Recharts para la visualización gráfica de datos en el panel de estadísticas. Ambas son librerías de JavaScript bastante utilizadas en aplicaciones web con React. La navegación se gestiona directamente en el cliente utilizando HashRouter (react-router-dom), lo que elimina la necesidad de dependencias del servidor para las rutas, mientras que la interacción con el backend se lleva a cabo con axios, incorporando el token de autenticación en la cabecera de cada petición.

La capa de servicios de aplicación se desarrolla con Django REST Framework, que presenta una API REST utilizada por el cliente. En esta capa se encuentran los ViewSets responsables de las funciones de listado, creación, modificación y eliminación de categorías y movimientos, junto con los endpoints particulares de agregación (resumen por categorías y series mensuales). Los Serializers establecen el acuerdo de datos entre el cliente y el servidor y encierran algunas de las reglas de negocio (como que la cantidad de un movimiento debe ser positiva o que no se puede repetir una categoría por nombre y tipo para un mismo usuario). Asimismo, los permisos y los querysets seleccionan sistemáticamente la información por el usuario autenticado, asegurando el aislamiento lógico de los datos.

Finalmente, la capa de dominio y el acceso a datos se fundamenta en el ORM de Django. Se modelan las entidades clave (Categoría y Movimiento) y sus relaciones, apoyadas por restricciones a nivel de base de datos (unicidad compuesta usuario+nombre+tipo en categorías, integridad referencial con movimientos, entre otros). Esta capa también agrupa las consultas y agregaciones que respaldan la funcionalidad de filtrado por fecha, tipo o categoría, junto con la paginación y la ordenación, además de los cálculos requeridos para el tablero. La distribución de funciones entre capas hace posible que las variaciones en la presentación no afecten a la lógica de negocio y que las reglas del dominio se mantengan unidas y comprobables tanto en el servidor como, cuando sea necesario, en el cliente.

4.3. Modelo lógico y físico de datos

4.3.1. Modelo lógico de datos

El diseño del modelo de datos se ha realizado con un enfoque relacional y normalizado, manteniendo las entidades que representan conceptos diferentes del dominio separadas: usuarios, categorías y movimientos.

La identificación del usuario se basa en la tabla estándar de autenticación de Django (auth_user), la cual el sistema utiliza para iniciar sesión y vincular la información generada por cada individuo.

En esa línea, la entidad Categoría almacena el nombre y el tipo (ingreso/gasto). Cada categoría está asignada a un único usuario, permitiendo que dos personas empleen nombres y formatos parecidos sin que se afecten mutuamente. Para prevenir duplicados indeseados, se establece una restricción de unicidad compuesta por (usuario, nombre en minúsculas, tipo), de esta manera, un mismo usuario no puede registrar dos categorías “Salario” de tipo ingreso, pero podría tener “Salario” como ingreso y “Salario” como gasto si lo requiere. Esta elección conserva el catálogo organizado y agiliza el filtrado posterior.

La entidad Movimiento representa cada registro económico y se relaciona con una única categoría. Sus características incluyen: fecha (tipo fecha, empleado para ordenación y filtros), cantidad (decimal con dos cifras decimales) y descripción (texto opcional para búsquedas). La aplicación verifica que la cantidad sea estrictamente mayor que cero, de tal forma que el signo contable se determina según el tipo de la categoría (ingreso suma, gasto resta). Esta convención simplifica la interfaz y previene inconsistencias (por ejemplo, “ingresos” negativos). Asimismo, la vinculación con Categoría se ha establecido de tal manera que no se puede eliminar una categoría que tenga movimientos asociados: primero debe ser vaciada o reasignada, asegurando la integridad referencial desde la lógica de negocio y los puntos endpoints.

Desde la perspectiva del rendimiento, el modelo obtiene ventajas de los índices automáticos que Django implementa sobre las claves primarias y foráneas (como categoria_id en movimientos y el campo de usuario en categorías). Esto, junto con la paginación, la ordenación y los filtros por intervalo de fechas, tipo y categoría implementados en la API, facilita la consulta de listas extensas sin afectar el tiempo de respuesta. Las métricas agregadas (totales por categoría y series mensuales) no se guardan, se generan bajo pedido mediante agregaciones SQL, previniendo duplicados de datos derivados y conservando el modelo en tercera forma normal.

Gracias a esto, el diseño conserva una estructura básica pero adecuada para abordar el caso de uso: el usuario gestiona su propio catálogo de categorías (ingresos/gastos) y registra movimientos con fecha, cantidad y una descripción opcional. Las reglas de integridad y las restricciones de unicidad (evitar la eliminación de categorías en uso, mantener cantidades positivas) se implementan de forma coherente, lo que simplifica tanto la utilización diaria como el desarrollo futuro del sistema sin poner en riesgo la calidad de los datos.

4.3.2. Modelo físico de datos

La base de datos se implementa en PostgreSQL y se accede vía Django ORM. La autenticación reutiliza la tabla estándar auth_user, siendo las tablas propias del dominio categorias y movimientos. La evolución del esquema se gestiona con migraciones de Django

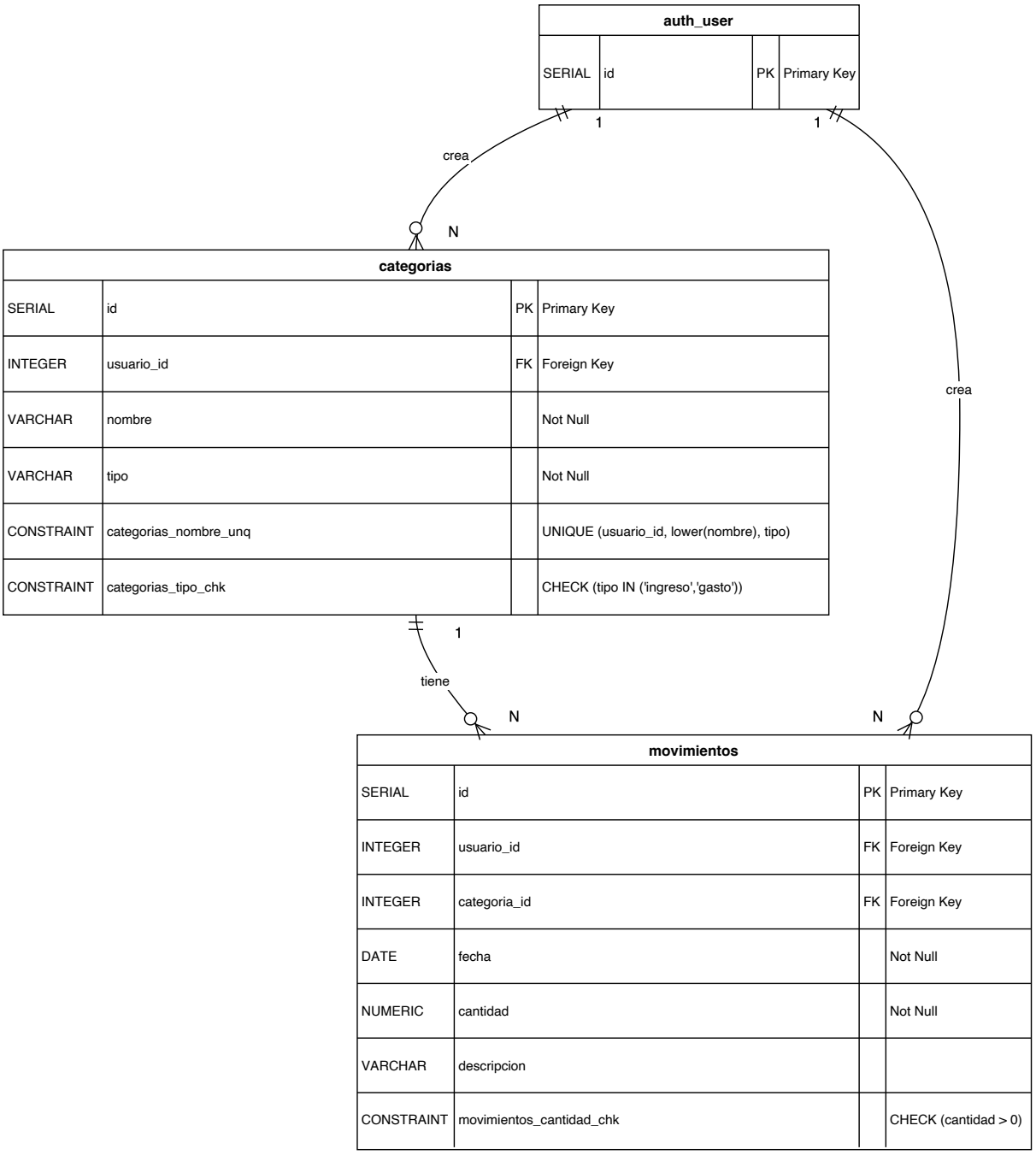


Figura 4.2: Diagrama físico de datos

Cabe destacar que, como ya ha sido mencionado, se ha tomado algunas decisiones relevantes para el correcto funcionamiento de la aplicación. Se ha optado por la unicidad de usuario+nombre+tipo en categorías para evitar duplicados, incluyendo lower(nombre) para no haces distinciones entre mayúsculas y minúsculas. Por otra parte, se ha optado por validar tipo con CHECK para evitar valores fuera del dominio. Que la cantidad deba ser mayor a cero se garantiza en la base de datos y en el backend con el CHECK.

En la parte de categorías, se ha puesto ON DELETE RESTRICT para que el usuario primero deba eliminar o reasignar los movimientos antes de eliminar una categoría. Se añade

también índices en (usuario_id, fecha), categoría_id y descripción (para búsquedas simples) además de paginación y ordenación desde la API para mejorar el rendimiento.

Respecto a la seguridad, las contraseñas no se almacenan en tablas propias, como ya se mostraba antes, sino que residen en auth_user gestionado por Django con hash seguro y el acceso a filas se aísla por usuario_id desde las vistas/serializers para evitar lecturas cruzadas.

En resumen, el modelo físico se origina desde la base del modelo lógico en PostgreSQL con tipos, restricciones e índices que hacen posible mantener consistencia, seguridad y buen rendimiento para los patrones de uso de la aplicación.

4.4. Contrato de la API REST

La aplicación presenta una API REST centrada en recursos con el prefijo /api/, utilizada por el frontend React. El contrato está diseñado para ser predecible, coherente y seguro, permitiendo que cualquier cliente (no solo el frontend actual) se integre sin dificultad. Las API REST utilizan endpoints como método de respuesta a las peticiones del servicio.

4.4.1. Principios y convenciones generales

La API, como se mencionan anteriormente, se estructura con el prefijo común /api/, incorporando una versión implícita en el desarrollo de este proyecto. Este enfoque posibilita, en su momento, migrar a rutas como /api/v2/ sin afectar la compatibilidad con integraciones previas. Cada una de las peticiones y respuestas se transmiten en formato JSON y se ofrecen con el encabezado Content-Type: application/json; charset=utf-8; la codificación utilizada es UTF-8 y las fechas se muestran en el formato ISO YYYY-MM-DD, considerando la zona horaria del servidor como punto de referencia.

El acceso está protegido, excepto para el registro de usuario, todas las acciones precisan de un token de autenticación válido. Asimismo, cada consulta se limita de forma automática a los datos del usuario que ha iniciado sesión, por lo que no se puede acceder ni alterar información de otras personas. Para el entorno de desarrollo (y considerando futuros despliegues) se activa CORS solo para el origen del frontend. Finalmente, se mantienen las nociones fundamentales de idempotencia: las acciones de lectura y eliminación (GET y DELETE) no cambian el estado más allá de su efecto previsto y pueden repetirse sin resultados inesperados, mientras que POST se utiliza para crear recursos y PATCH para cambios parciales y específicos.

4.4.2. Autenticación y registro

- **POST /api/registro/**

- **Descripción:** alta de usuario y entrega de token de sesión.

- **Parámetros:** como parte del body JSON se requiere el nombre de usuario, la contraseña dos veces, para verificar coincidencia y el email (de manera opcional).
- **Salida:** el servidor verifica que el alta del usuario se ha realizado correctamente y entrega un token, el cual se usará en las demás peticiones como credencial de acceso.

- **POST /api-token-auth/**

- **Descripción:** obtiene un token de sesión a partir del usuario y la contraseña.
- **Parámetros:** como parte del body JSON se requiere el nombre de usuario y la contraseña.
- **Salida:** el usuario ha iniciado sesión de manera correcta y se le concede un token de autenticación. Este token funciona igual que el anterior.

4.4.3. Categorías

- **GET /api/categorias/**

- **Descripción:** lista de categorías del usuario.
- **Parámetros:** como parámetros de consulta se puede añadir el tipo (ingreso/gasto) para filtrar por tipo.
- **Salida:** se proporciona la lista total de categorías del usuario (filtrada por tipo, si se desea). Cada componente señala su identificación, el nombre que se muestra y si corresponde a ingresos o gastos.

- **POST /api/categorias/**

- **Descripción:** crea una nueva categoría para el usuario.
- **Parámetros:** como parte del body JSON se debe añadir el nombre de la categoría a añadir y el tipo (ingreso/gasto).
- **Salida:** se crea la categoría y se le asigna su id correspondiente. A partir de ese momento se puede utilizar dicha categoría para adicionarle movimientos.

- **GET /api/categorias/{id}/**

- **Descripción:** detalle de una categoría del usuario.
- **Parámetros:** como parte de la ruta se debe añadir el id de la categoría.
- **Salida:** se obtiene el detalle de la categoría concreta {id}. Esto es útil para preparar o mostrar una edición.

- **PATCH /api/categorias/{id}/**

- **Descripción:** modifica parcialmente una categoría.
- **Parámetros:** como parte de la ruta se debe añadir el id de la categoría y como parte del body JSON o el nombre o el tipo de la categoría.
- **Salida:** la categoría se actualiza parcialmente y se devuelve el estado final tras la modificación.

- **DELETE /api/categorias/{id}/**

- **Descripción:** elimina una categoría solo si no contiene movimientos asociados.
- **Parámetros:** como parte de la ruta se debe añadir el id de la categoría.
- **Salida:** no hay salida, pero se elimina la categoría.

4.4.4. Movimientos

- **GET /api/categorias/**

- **Descripción:** lista paginada de movimientos del usuario con filtros, búsqueda y ordenación.
- **Parámetros:** vamos a diferenciar los parámetros según el caso:
    - ***Filtrado:*** como parámetros de consulta podemos filtrar por la categoría, el tipo y rangos de fechas.
    - ***Búsqueda:*** como parámetro de consulta podemos buscar por el texto de la descripción del movimiento.
    - ***Ordenación:*** como parámetros de consulta podemos ordenar por fecha (más reciente/más antiguo), cantidades (mayores/menores) o por descripción(A-Z/Z-A).
    - ***Paginación:*** como parámetros de consulta podemos añadir la página o el tamaño de página.
- **Salida:** se devuelve una lista de movimientos aplicando los filtros y la ordenación solicitada, pero paginada. Dicha lista nos dice cuántos elementos hay en total, nos muestra solo la página actual, pero nos permite navegar por las demás páginas si estas existieran. Además, cada movimiento incluye su categoría (por id), fecha, descripción e importe.

- **POST /api/movimientos/**

- **Descripción:** crea un movimiento.
- **Parámetros:** como parte del body JSON se debe añadir la categoría, la fecha, la cantidad (mayor que cero) y la descripción (opcional, aunque recomendada).
- **Salida:** se devuelve el registro del movimiento creado y como este queda creado.

- **GET /api/movimientos/{id}/**

- **Descripción:** detalle de un movimiento del usuario.
- **Parámetros:** como parte de la ruta se debe añadir el id del movimiento.
- **Salida:** se obtiene el detalle del movimiento concreto {id}. Esto es útil para mostrar información o prefijar un formulario de edición.

- **PATCH /api/movimientos/{id}/**

- **Descripción:** modifica parcialmente un movimiento.

- **Parámetros:** como parte de la ruta se debe añadir el id del movimiento y como parte del body JSON alguno de los parámetros referentes a la categoría, fecha, cantidad o descripción.
- **Salida:** se muestra el estado final guardado en la base de datos del movimiento que ha sido modificado.

- **DELETE /api/movimientos/{id}/**

- **Descripción:** elimina un movimiento del usuario.
- **Parámetros:** como parte de la ruta se debe añadir el id del movimiento.
- **Salida:** no hay salida, pero se elimina el movimiento.

- **GET /api/movimientos/resumen**

- **Descripción:** movimientos totales del periodo (según filtros) y desglose por categoría.
- **Parámetros:** mismos parámetros que GET /api/movimientos/ pero sin paginación.
- **Salida:** un resumen consolidado para el rango y filtros especificados. Se muestra las cifras totales de ingresos, gastos y balance (ingreso – gastos), además de un desglose por categoría con el total acumulado por cada una. Esta respuesta es la base de los KPIs y de los gráficos por categorías del apartado de estadísticas.

- **GET /api/movimientos/resumen-mensual/**

- **Descripción:** serie temporal mensual, con los ingresos, gastos y balance por mes, respetando los mismos filtros.
- **Parámetros:** mismos parámetros que GET /api/movimientos/ pero sin paginación.
- **Salida:** devuelve una serie temporal agrupada por mes, con los importes de ingresos, gastos y balance de cada mes. Es la fuente de datos para el gráfico “Ingresos vs Gastos por mes” del apartado de estadísticas.

4.5. Diseño de la interfaz de usuario

La interfaz se ha diseñado con un objetivo definido, que no es otro que permitir que el usuario registre, consulte y comprenda sus finanzas con la menor problemática posible. La capa de presentación en React estructura la aplicación en vistas simples, interrelacionadas y respaldadas por componentes de Material UI, brindando consistencia visual, accesibilidad predeterminada y un comportamiento responsivo sin requerir estilos complicados.

4.5.1. Diseño inicial

Antes de realizar la interfaz gráfica final con la que el usuario pueda gestionar sus finanzas de manera óptima se realizó una serie de diseños que servirían como base para la implementación de esta.

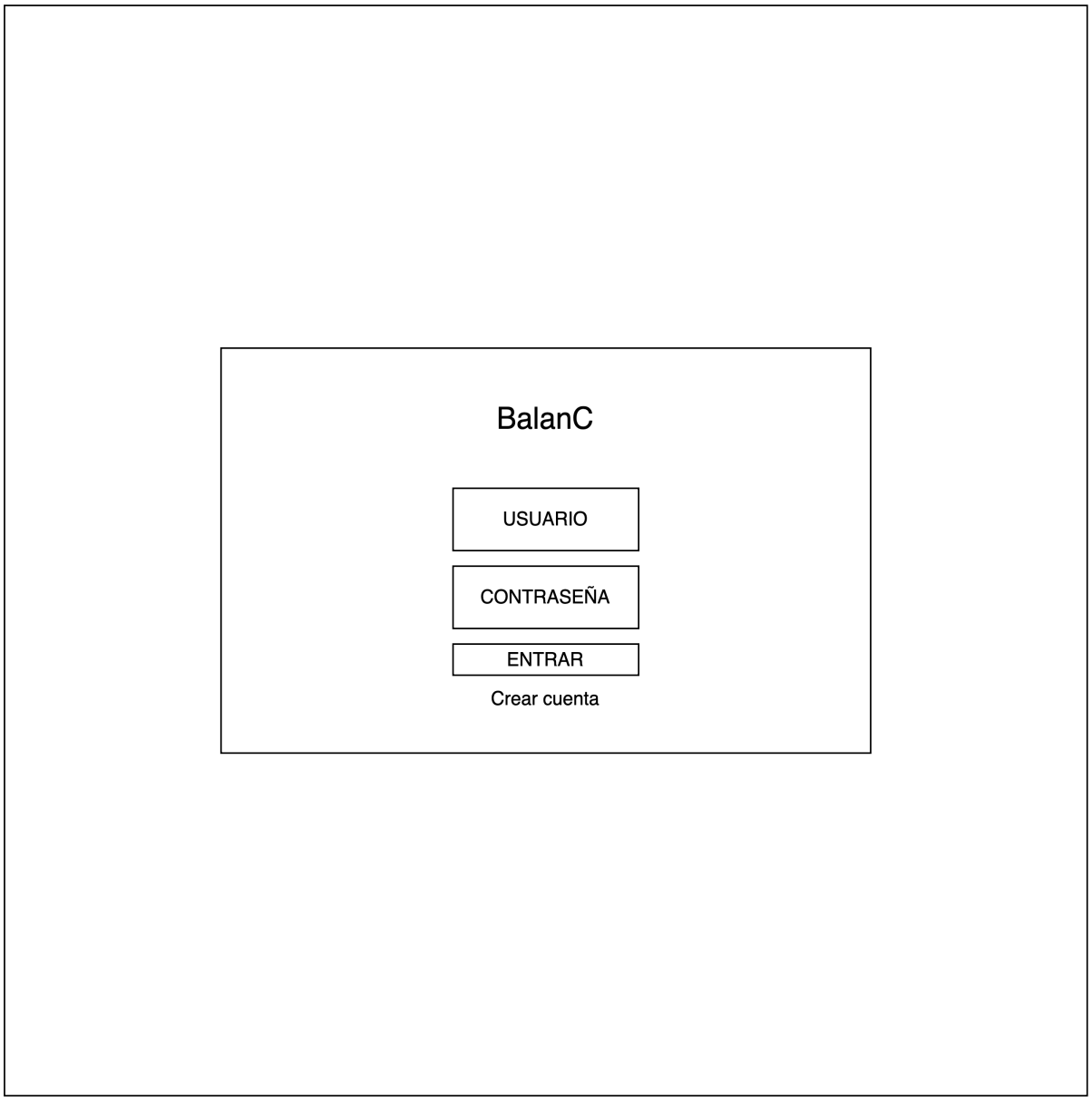


Figura 4.3: Diseño de pantalla inicial

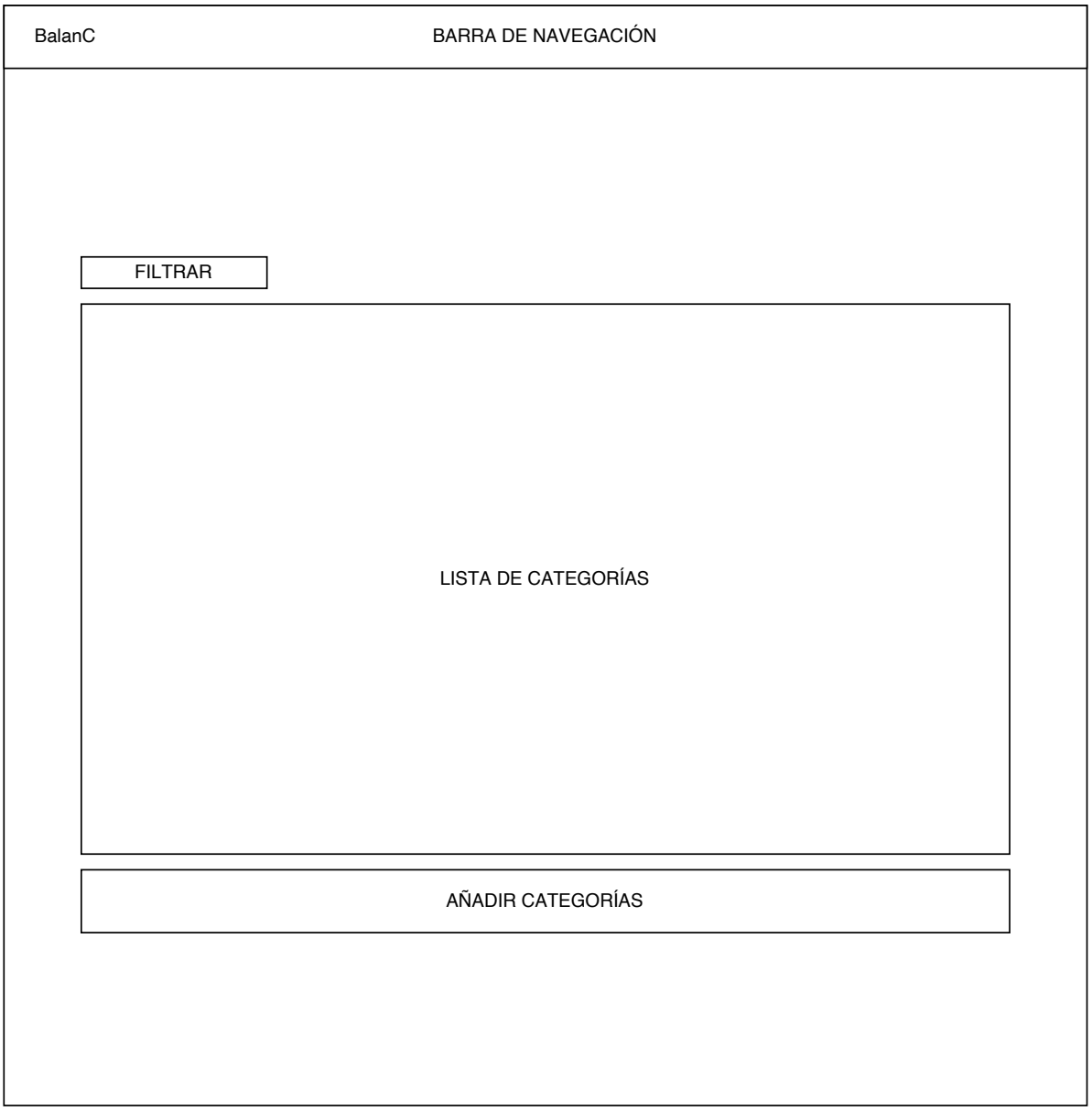


Figura 4.4: Diseño de vista categorías

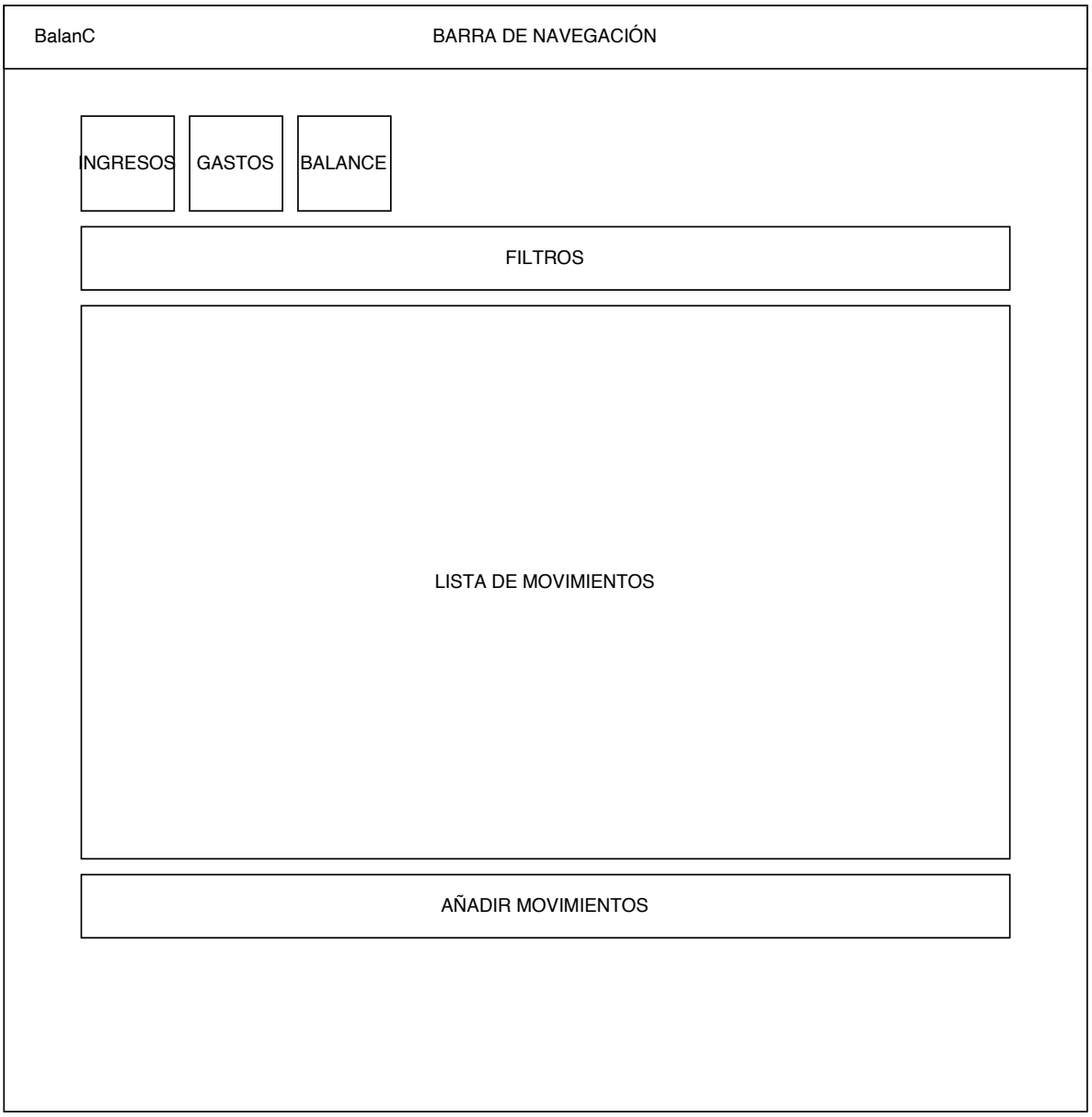


Figura 4.5: Diseño de vista movimientos

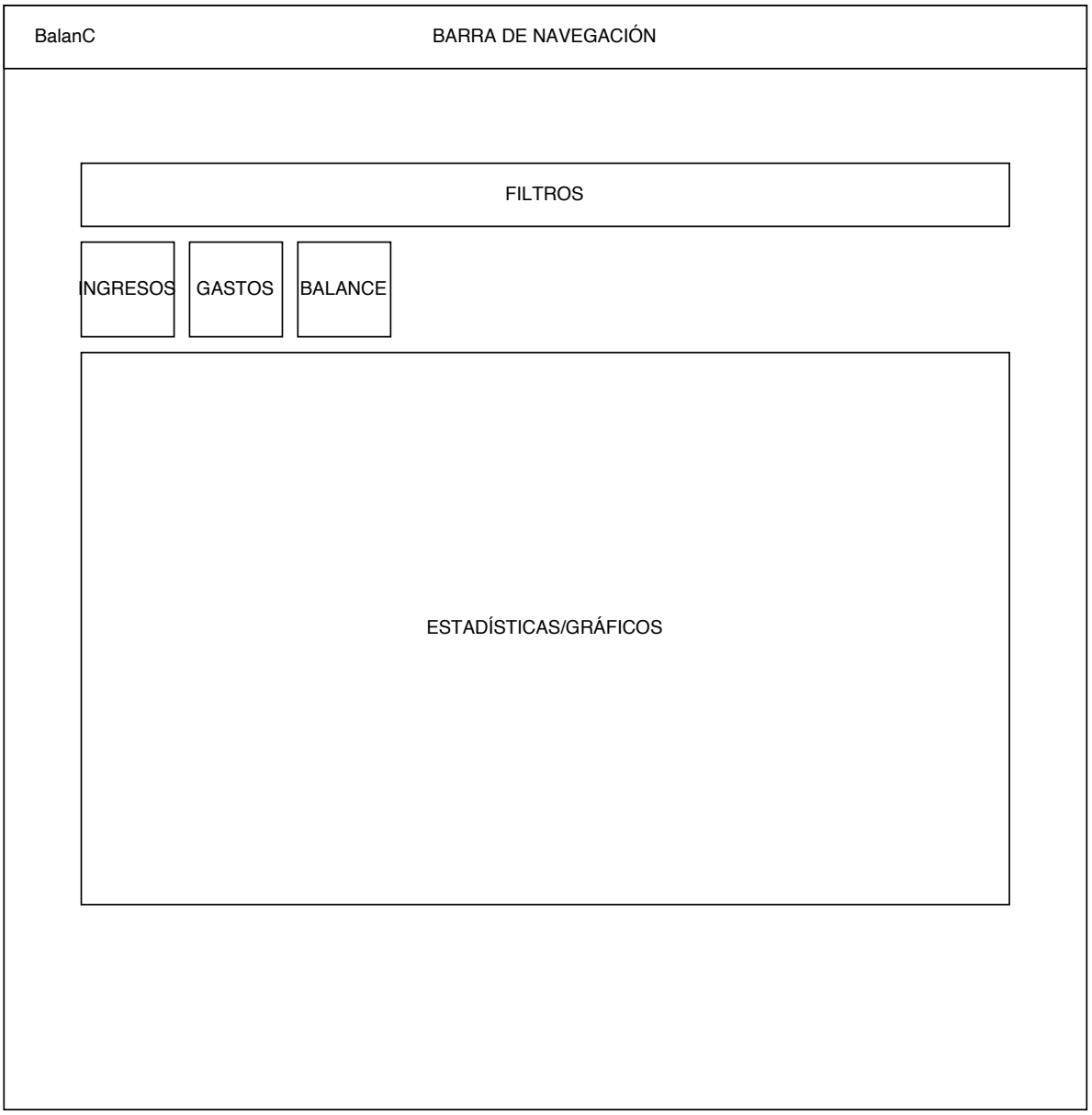


Figura 4.6: Diseño de vista estadísticas

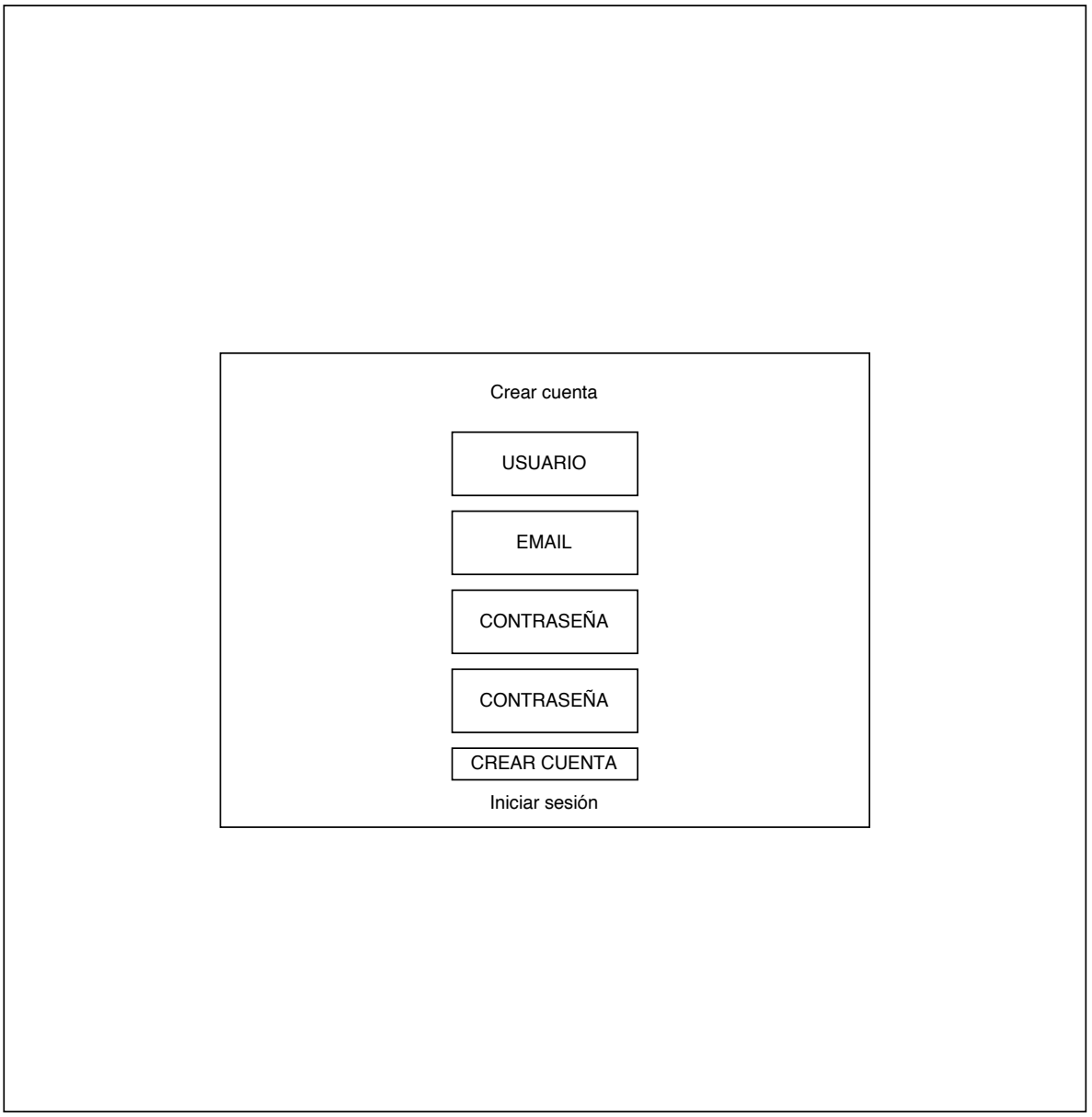


Figura 4.7: Diseño de registrar usuario

4.5.2. Estructura general y navegación

La aplicación utiliza una navegación de página única (SPA). Después de la autenticación, el usuario puede acceder a un menú superior (Navbar) que facilita la navegación entre “Categorías”, “Movimientos” y “Estadísticas”. El elemento activo se destaca, por lo que la ubicación siempre es clara. Las rutas sensibles están aseguradas, por lo que, si no hay una sesión válida, el sistema dirige al inicio de sesión. Para minimizar problemas al trabajar en local (y facilitar la corrección), se utiliza hash routing, lo que previene conflictos con el backend.

4.5.3. Registro e inicio de sesión

El registro pide un nombre de usuario y una contraseña (también se puede añadir de forma opcional el correo electrónico). Se han añadido asistencias en línea y mensajes de error claros (por ejemplo, si la contraseña no cumple con los requisitos básicos). Al iniciar sesión, los errores de credenciales se informan de manera clara y sin detalles innecesarios. Una vez autenticado, el navegador guarda un token que se utiliza de manera automática en cada petición a la API.

4.5.4. Gestión de categorías

La pantalla de categorías tiene dos objetivos claros, visualizar de manera rápida el catálogo personal y formar nuevas categorías con la menor cantidad de pasos. La lista presenta el nombre y una etiqueta de tipo (ingreso/gasto) con códigos de color (verde y rojo) para facilitar el reconocimiento instantáneo. Se añade un filtro por categoría (todos/ingresos/gastos) que no actualiza la página y preserva el contexto del usuario. Al realizar la creación, la validación se lleva a cabo en el cliente (campos requeridos) y en el servidor (unicidad por usuario + nombre + tipo). Si se desea borrar una categoría con movimientos vinculados, el sistema comunica claramente la razón para evitar confusiones.

4.5.5. Gestión de movimientos

La vista de movimientos integra en una misma pantalla los siguientes elementos:

- KPIs del periodo (ingresos, gastos y balance) con color ajustable en el balance (verde si es positivo y rojo si es negativo).
- Filtros y herramientas: intervalo de fechas (con calendario), tipo (ingreso/gasto), categoría (dependiente del tipo elegido), búsqueda por descripción, clasificación (fecha, cantidad, alfabético) y tamaño de página. Al modificar cualquier criterio, la lista se refresca sin perder la atención del usuario.
- Lista de acciones rápidas: cada entrada presenta fecha, descripción, categoría y un indicador de monto con símbolo y color determinados por el tipo. Las funciones de edición y eliminación están disponibles en línea; al modificar, los campos se convierten en controles de formulario en la misma fila, evitando cambios de pantalla.
- Creación asistida: el formulario “Nuevo movimiento” configura por defecto la fecha actual, requiere cantidades positivas y proporciona una ayuda textual (“Ingresa un importe positivo”). Se ha decidido deducir el signo a partir del tipo de categoría para evitar incongruencias (por ejemplo, ingresos en negativo).

4.5.6. Estadísticas y visualización

La vista de estadísticas combina filtros temporales, como los botones creados para cambiar entre el año en curso, este mes y el mes anterior, con una actualización inmediata de los gráficos. Los gráficos que se han elegido para expresar de forma clara y concisa las finanzas del usuario son:

- **Barras agrupadas por mes (Ingresos vs Gastos)** para facilitar la comparación temporal de forma intuitiva. La leyenda además replica el orden de aparición de cada barra, estando a la izquierda los ingresos y a la derecha los gastos y la leyenda muestra el paso de los meses de forma ordenada.

- **Gráficos circulares por categoría** (ingresos y gastos por separado) donde se muestran las proporciones de las categorías, con su porcentaje y referente a los periodos seleccionados. Estas se ajustan dinámicamente al tamaño disponible y utilizan paletas contrastadas para distinguir las diferentes categorías manteniendo a su vez las etiquetas legibles.

4.5.7. Mensajes, estados y accesibilidad

Los errores y las confirmaciones se muestran a través de alertas sutiles que se desvanecen después de unos segundos, evitando así abrumar al usuario. Cuando un listado no obtiene resultados debido a filtros demasiado restrictivos o a la falta de datos, se presenta un estado vacío evidente. En lo que respecta a la accesibilidad, se han utilizado elementos de Material UI, que es una biblioteca que facilita el desarrollo de interfaces de usuarios moderna, proporcionando elementos ya estilizados y listos para usar. Estos elementos de MUI proporcionan roles, etiquetas y un contraste adecuado. Además, el color no es el único medio de información, ya que también se incluyen textos y símbolos.

En su totalidad, la aplicación se ha construido con la capacidad de ser responsive, es decir, que tiene la capacidad para adaptarse a los diferentes tamaños de página que puedan surgir sin que el contenido pierda su forma o no aparezca en ese tamaño concreto, ya se haga esta más pequeña o grande. Además, se ha procurado que las pequeñas interacciones, como resaltar el botón activo o conservar el estado de los filtros al paginar, sirvan de gran ayuda a los usuarios para que su experiencia sea más fluida y que no requiera de gran complejidad a este.

4.5.8. Decisiones de diseño

Durante el diseño de la interfaz se ha buscado, ante todo, reducir la complejidad innecesaria. Para ello se ha priorizado que el usuario pueda comprender y utilizar la aplicación sin una curva de aprendizaje elevada. Menos pantallas, menos pasos y más acciones disponibles donde son realmente necesarias. Esta idea se traduce en flujos más directos, como la posibilidad de crear movimientos desde el propio listado, y en formularios con campos imprescindibles, facilitando así que la interacción del usuario con la interfaz sea lo más sencilla y concreta posible.

La aplicación brinda retroalimentación instantánea y limitada en duración para cada acción significativa. Al crear, modificar, borrar o aplicar filtros, el sistema confirma el resultado con mensajes claros que se desvanecen después de unos segundos, evitando saturar la vista y permitiendo conservar la concentración en la tarea principal. Este modelo de

retroalimentación constante permite al usuario desarrollar confianza en el sistema, dado que cada acción genera una respuesta clara y esperada.

Desde el punto de vista visual, se ha optado por mantener una línea coherente en tipografías, tamaños, espaciados y jerarquías, de modo que queda bastante claro y a la vista los elementos más importantes, los cuales se distinguen por su posición y su peso. También se ha optado por usar el color con moderación y con un significado concreto, resaltar lo verdaderamente importante que son los ingresos y los gastos, quienes aparecen de color verde y rojo respectivamente. Esta coherencia facilita al usuario la lectura e interpretación de las diferentes listas, formularios y gráficas, reduciendo así el esfuerzo cognitivo.

Por último, se ha optado por optimizar las tareas más frecuentes para facilitar al usuario la utilización de los diferentes elementos de la aplicación. Entre esto se destaca que la fecha a la hora de añadir un movimiento por defecto aparece en el día actual, la validación de cantidades de realiza de forma temprana para prevenir errores y los filtros mantienen su estado para que el usuario no tenga que estar configurándolos repetidas veces. En conjunto, estas decisiones permiten que el usuario se centre en lo primordial, que es la gestión de sus finanzas personales, sin tener que invertir tiempo en aprender cómo funciona la aplicación, como se navega por ella o como se configura.

Capítulo 5: Implementación

En este capítulo se aborda como se ha construido el sistema y las tecnologías o herramientas utilizadas para ello.

5.1. Backend: Python, Django y Django REST Framework

El lado del servidor ha sido creado en Python empleando Django como marco principal y Django REST Framework (DRF) para proporcionar una API limpia y coherente. Esta elección tiene dos objetivos, por un lado, beneficiarse de la solidez del ecosistema Django (autenticación, ORM, migraciones, administración) y, por el otro, establecer un contrato HTTP bien definido que el cliente pueda utilizar sin dependencias.

La lógica del dominio se enfoca en las aplicaciones de Django,en los que los modelos contienen las reglas esenciales (por ejemplo, que la cantidad de un movimiento sea positiva y que el nombre y tipo de una categoría no se puedan repetir por usuario), mientras que los serializers establecen el formato de intercambio en JSON y validan las entradas antes de ser almacenadas en la base de datos. Los ViewSets de DRF ofrecen funciones de listado, creación, modificación y eliminación, junto con dos agregaciones particulares para el área de estadísticas (resumen por categorías y serie mensual). Cualquier solicitud, excepto el alta, requiere un token de autenticación, y los querysets se filtran de manera sistemática según el usuario autenticado para prevenir accesos indebidos.

El ORM de Django permite un desarrollo seguro y expresivo, pues las consultas se realizan en Python y se convierten a un SQL eficiente, con funcionalidades para filtros, ordenación, paginación y anotaciones que nutren los gráficos sin replicar datos calculados.

5.2. Frontend: React, Material UI, Recharts y axios

La interfaz ha sido desarrollada con React utilizando una estructura de componentes básica y enfocada en tareas. Material UI ofrece una base visual uniforme y accesible (tipografías, tamaños, espaciados y estados de interacción consistentes), permitiendo que el énfasis se posicione en el flujo funcional y no en estilos únicos. Se ha elegido Recharts para la visualización de datos, ya que se integra fácilmente en React y posibilita la actualización de gráficos al cambiar filtros o intervalos temporales.

El cliente se comunica con la API usando axios, que agrega automáticamente el encabezado Authorization: Token … cuando hay una sesión activa. La navegación se gestiona con react-router-dom en modo HashRouter, lo que facilita el trabajo local y previene conflictos con el servidor. La sección de Movimientos integra filtros, búsqueda, ordenamiento y paginación en un único espacio, con edición en línea y validaciones al instante, Estadísticas presenta barras agrupadas (ingresos frente a gastos por mes) y gráficos circulares por categorías y finalmente, Categorías permite un alta directa con verificaciones de unicidad y avisos claros cuando no se puede eliminar una categoría por tener movimientos vinculados.

5.3. Base de datos: PostgreSQL

Para el almacenamiento de datos se utiliza PostgreSQL. Se ha elegido utilizar PostgreSQL debido a su solidez, alto rendimiento y soporte de restricciones avanzadas. Las migraciones de Django regulan el desarrollo del esquema y garantizan la reproducibilidad entre diferentes entornos. Aparte de las claves primarias y foráneas, se han establecido restricciones de unicidad (usuario+nombre+tipo) y checks para asegurar que las cantidades sean estrictamente positivas. Esta mezcla disminuye errores lógicos, preserva la coherencia y simplifica el código del servidor.

5.4. Entorno de desarrollo

El desarrollo se ha llevado a cabo en macOS utilizando Python 3 en un entorno virtual para aislar dependencias, y con Node.js y npm para el cliente. El editor principal utilizado ha sido Visual Studio Code, aprovechando extensiones de formato, detección de errores y soporte nativo para Git.

El proyecto ha sido estructurado en dos directorios: backend (Django) y frontend (React), además de un repositorio Git que se encuentra en GitHub, lo que permite versionar los cambios y facilita la revisión de diferencias, la creación de ramas temporales o la restauración a versiones estables. Para validar la API durante el desarrollo se utilizó Postman, lo que facilitó la verificación de encabezados, paginación, filtros y respuestas antes de incorporar cada vista en el cliente.

La puesta en marcha local es bastante sencilla: python manage.py runserver para el backend y npm start para el frontend. De este modo, el navegador es capaz de cargar la SPA y utilizar la API sin configuraciones extras.

5.5. Herramientas de apoyo

Además de lo mencionado anteriormente, se ha utilizado herramientas que han facilitado la construcción y documentación de la aplicación. Postman, ya mencionado antes, ha sido clave para depurar los endpoints y validar el ciclo de autenticación por token. Github, también mencionado antes, ha servido para almacenar el código y registrar la correcta evolución del proyecto. Como herramientas no mencionadas, se ha utilizado draw io para diseñar diagramas que se han incluido en este documento. Google Drive ha sido clave para poder almacenar en la nube el desarrollo de la memoria y poder acceder a ella desde diferentes dispositivos.

5.6. Consideraciones de calidad

Durante el desarrollo del proyecto se han tomado diferentes decisiones que afectaban directamente a la experiencia y mantenibilidad de este. La validación temprana en cliente y servidor ha servido para reducir errores de entrada y evitar estados inválidos. La paginación

y ordenación garantizan respuestas ágiles, aunque exista un crecimiento notorio en el volumen de los movimientos. La separación entre la interfaz y la API limita el acoplamiento y facilita futuras extensiones (por ejemplo, un cliente móvil).

El uso de restricciones en la base de datos, a través de las unicidades y checks, junto con pruebas de la API para casos más sensibles, aporta confianza y muestra que el sistema se comporta de la forma esperada y que el modelo de datos se mantiene íntegro a medida que va evolucionando.

5.7. Incidencias y soluciones

Muchas de las incidencias que surgieron durante el desarrollo del proyecto se solucionaron con diversas situaciones que han sido mencionadas a lo largo del documento, pero a continuación vamos a recopilar las más relevantes y las soluciones por las que se optaron en el momento en que se encontró dicha incidencia.

Al principio, algunas peticiones a la API devolvían 401 o Unauthorized debido a tokens caducados o mal enviados. Esto fue resuelto centralizando el envío del encabezado Authorization: Token <valor> en axios y documentando en Postamn la obtención o renovación del token antes de probar la API.

Más adelante, el frontend a pesar de paginar los movimientos, los mostraba todos. El origen de dicha incidencia fue una discrepancia entre los parámetros esperados por la API y los que enviaba el cliente. Para la resolución de esta incidencia, se unificó el contrato (page, page_size, ordering) y se actualizó para leer count/results cuando la paginación estuviera activa.

Un tema ya bastante mencionado ha sido la unicidad de las categorías duplicadas, y es que, en su momento, realizando pruebas en el sistema, se detectó que un mismo usuario podía crear una categoría con el mismo nombre y tipo cambiando mayúsculas y minúsculas. Para ello se añadió una restricción compuesta (usuario, lower(nombre), tipo) y la validación correspondiente en serializer, devolviendo un mensaje claro al usuario sobre la ya existencia de dicha categoría.

Con respecto a los movimientos, en su momento se averiguó que un usuario podía adicionar movimientos con signos contrarios a los del tipo del propio movimiento, distorsionando los valores reales. Para ello, se implementó una validación en modelo y serializer de la cantidad, que debía ser mayor que cero, y una ayuda al usuario en el formulario a la hora de introducir la cantidad de nuevos movimientos.

Con la detección de estas incidencias y sus correspondientes correcciones, el sistema se transformó en uno mucho más robusto y confiable, mejorando así la experiencia de uso, especialmente en los flujos de mayor afluencia, como el alta y consulta de movimientos.

Capítulo 6: Verificación y pruebas

Este apartado se ha construido para realizar diversas pruebas que garanticen el correcto funcionamiento del sistema, haciendo especial incidencia en los requisitos planteados previamente en el capítulo 2.

6.1. Entorno de verificación

Para la realización de las pruebas se ha utilizado un dispositivo Macbook Air con las siguientes características:

- Chip: Apple M3.
- Memoria RAM: 16 GB.
- Almacenamiento: 512 GB SSD.
- Tarjeta gráfica: GPU integrada Apple M3.
- Sistema operativo: MacOS Sonoma 15.5.

6.2. Estrategia

La estrategia seguida para probar el correcto funcionamiento de la aplicación ha sido un conjunto de pruebas manuales encargadas de verificar el rendimiento frente a las acciones esperadas de los requisitos del sistema.

6.3. Pruebas ejecutadas

En esta sección se muestran las pruebas realizadas.

6.3.1. Pruebas de los requisitos funcionales.

P-1	**Prueba de registro de usuario**
Descripción	Comprobar el correcto registro de un usuario en el sistema.
Referencia	RF-1
Datos de entrada	● Usuario: noah ● Email (opcional): noahrg@gmail.com ● Contraseña: PruebaTFG ● Repite la contraseña: PruebaTFG

Salida esperada	El usuario queda registrado en el sistema y se inicia sesión.
Resultados	El usuario se registra correctamente y se accede directamente con su usuario a la sesión.

Tabla 6.1: P-1 Prueba de registro de usuario

P-2	**Prueba de inicio de sesión**
Descripción	Comprobar que el usuario inicia sesión de manera satisfactoria en el sistema.
Referencia	RF-2
Datos de entrada	● Usuario: noah ● Contraseña: PruebaTFG
Salida esperada	El usuario inicia sesión en el sistema.
Resultados	El usuario inicia sesión correctamente.

Tabla 6.2: P-2 Prueba de inicio de sesión

P-3	**Prueba de cierre de sesión**
Descripción	Comprobar que el usuario cierra sesión de manera satisfactoria en el sistema.
Referencia	RF-3
Datos de entrada	No hay datos de entrada
Salida esperada	El usuario cierra sesión y se redirige a la página de inicio.
Resultados	El usuario cierra sesión correctamente y se redirige a la página de inicio de sesión.

Tabla 6.3: P-3 Prueba de cierre de sesión

P-4	**Prueba de crear categoría**
Descripción	Comprobar la correcta creación de la categoría.
Referencia	RF-4

Datos de entrada	• Nombre: Luz • Tipo: Gasto
Salida esperada	Categoría creada correctamente.
Resultados	Categoría creada correctamente y mensaje del sistema que lo indica.

Tabla 6.4: P-4 Prueba de crear categoría

P-5	**Prueba de listar categorías**
Descripción	Comprobar que las categorías se muestran correctamente, incluyendo filtrado.
Referencia	RF-5
Observaciones	Esta prueba se ha realizado existiendo únicamente la categoría Luz de tipo gasto creada previamente.
Datos de entrada	Filtrar por ingresos.
Salida esperada	Listado vacío sin categorías que mostrar.
Resultados	No hay categorías para mostrar.

Tabla 6.5: P-5 Prueba de listar categorías

P-6	**Prueba de borrar categoría**
Descripción	Comprobar la correcta eliminación de la categoría.
Referencia	RF-6
Datos de entrada	No hay datos de entrada.
Salida esperada	La categoría queda eliminada.
Resultados	La categoría queda eliminada correctamente y mensaje del sistema que lo indica.

Tabla 6.6: P-6 Prueba de borrar categoría

P-7	**Prueba de crear movimiento**

Descripción	Comprobar la correcta creación de un movimiento.
Referencia	RF-7
Observaciones	Debido a que no hay ninguna categoría actualmente, se crea la categoría Luz de nuevo de tipo gasto.
Datos de entrada	● Categoría: Luz(gasto) ● Fecha: 30/10/2025 ● Cantidad: 150 euros ● Descripción: Factura de luz de octubre
Salida esperada	Movimiento creado correctamente. Se modifican los valores de gastos y balance.
Resultados	El movimiento se crea correctamente y se actualizan los valores de gastos y balance.

Tabla 6.7: P-7 Prueba de crear movimiento

P-8	**Prueba de listar movimientos**
Descripción	Comprobar que el sistema muestra los movimientos de manera correcta.
Referencia	RF-8
Datos de entrada	No hay datos de entrada.
Salida esperada	El usuario observa los movimientos existentes.
Resultados	El usuario visualiza correctamente los movimientos existentes.

Tabla 6.8: P-8 Prueba de listar movimientos

P-9	**Prueba de filtrar movimientos**
Descripción	Comprobar el sistema filtra los movimientos de manera correcta.
Referencia	RF-9
Observaciones	Esta prueba se ha realizado exclusivamente con la categoría Luz y el movimiento asociado a ella creado anteriormente.
Datos de entrada	Filtrar por movimientos de tipo ingreso

Salida esperada	El sistema muestra un listado vacío de movimientos y actualiza los valores de los KPIs para el filtro aplicado.
Resultados	El sistema muestra correctamente el listado vacío de movimientos y actualiza los valores de los KPIs para el filtro aplicado.

Tabla 6.9: P-1 Prueba de filtrar movimientos

P-10	**Prueba de buscar movimientos**
Descripción	Comprobar que el sistema muestra los movimientos buscados por su descripción.
Referencia	RF-10
Datos de entrada	Texto introducido: factura.
Salida esperada	El sistema muestra el movimiento que coincide con la búsqueda.
Resultados	Se muestra en la lista de movimientos los que coinciden con la búsqueda por descripción de factura.

Tabla 6.10: P-10 Prueba de buscar movimientos

P-11	**Prueba de ordenar movimientos**
Descripción	Comprobar que el sistema ordena los movimientos de manera correcta.
Referencia	RF-11
Observaciones	Para realizar una ordenación se necesitan mínimo dos movimientos, por lo que se ha creado un movimiento idéntico al ya existente, pero con una cantidad inferior, concretamente de 100 euros.
Datos de entrada	Filtro: Ordenar por cantidad (menor primero).
Salida esperada	El usuario observa los movimientos ordenados por cantidades, de menos a mayor.
Resultados	El usuario visualiza la lista de los movimientos correctamente ordenados.

Tabla 6.11: P-11 Prueba de ordenar movimientos

P-12	Prueba de editar movimiento
Descripción	Comprobar que el movimiento se edita correctamente.
Referencia	RF-12
Datos de entrada	• Cantidad: 200 euros.
Salida esperada	El movimiento actualiza la cantidad a la introducida.
Resultados	El movimiento se muestra de nuevo en el listado, pero se actualiza la cantidad a la introducida y por tanto se actualizan los KPIs.

Tabla 6.12: P-12 Prueba de editar movimiento

P-13	Prueba de eliminar movimiento
Descripción	Comprobar que el sistema elimina el movimiento correctamente.
Referencia	RF-13
Datos de entrada	No hay datos de entrada.
Salida esperada	El movimiento se elimina y se actualiza el listado de movimientos y los KPIs.
Resultados	El movimiento desaparece del listado y se actualizan los KPIs.

Tabla 6.13: P-13 Prueba de eliminar movimiento

P-14	Prueba de visualizar estadísticas
Descripción	Comprobar que el sistema muestra los gráficos y los valores correctamente.
Referencia	RF-14
Datos de entrada	No hay datos de entrada.
Salida esperada	El usuario observa las estadísticas de las categorías y los movimientos existentes.
Resultados	El usuario visualiza correctamente las estadísticas y los valores.

Tabla 6.14: P-14 Prueba de visualizar estadísticas

P-15	Prueba de filtrar estadísticas
Descripción	Comprobar que el sistema filtra las estadísticas de forma correcta.
Referencia	RF-15
Observaciones	No existen movimientos para el rango de fechas que se va a filtrar.
Datos de entrada	● Filtro de fecha: Desde (01/01/2025), Hasta (01/01/2025).
Salida esperada	El usuario observa que no hay estadísticas para el filtro seleccionado.
Resultados	El sistema no muestra ningún gráfico ni valor para las fechas seleccionadas.

Tabla 6.15: P-15 Prueba de filtrar estadísticas

6.3.2. Pruebas de los requisitos no funcionales

Para probar el correcto funcionamiento del sistema también debemos de realizar diferentes pruebas que muestren la calidad, características y cualidades del sistema. Para ello se han realizado diferentes validaciones referentes a los requisitos no funcionales propuestos en el capítulo 2:

- **RNF-1 Seguridad**

  Para verificar que los usuarios no pueden acceder al sistema sin haberse registrado previamente, se ha intentado iniciar sesión con unas credenciales que no han sido registradas previamente, lo cual ha sido imposible. Además, se ha introducido el usuario o la contraseña incorrecta de usuarios registrados, cambiando mayúsculas por minúsculas, pero aun así el acceso ha sido denegado.Por otra parte, se ha intentado conseguir la contraseña a través de la API, pero se valida que esta nunca se devuelve en respuesta de la API, sino que queda almacenada en hash seguro.

- **RNF-2 Usabilidad**

  Para la comprobación de este requisito se ha pedido a diferentes familiares y amigos la utilización de la interfaz de la aplicación sin el previo conocimiento de uso del sistema. Por lo general, la inmensa mayoría se desenvolvió de manera perfecta sin ningún tipo de inconveniente. Además, se les pidió que realizaran varios casos de uso sin explicación o ayuda y como antes, fueron resueltos con bastante sencillez. Finalmente, se les hizo algunas cuestiones sobre la interfaz y esta fue del agrado del

cómputo general, casi mayorotario, de personas que utilizaron la aplicación por primera vez.

- **RNF-3 Rendimiento**

El objetivo del rendimiento es asegurar tiempos de respuesta ágiles en operaciones comunes, para ello se registraron numerosos movimientos hasta obtener un listado bastante extenso, pero aun así la aplicación muestra tiempos de respuesta breves. Además, se ha utilizado de manera conjunta filtros de fecha, categoría y texto, junto a ordenación, lo cual no afectaba en absoluto a los tiempos de respuesta del sistema.

- **RNF-4 Mantenibilidad**

Para comprobar la mantenibilidad se realizó una revisión ágil del código para confirmar que el sistema es sencillo de comprender y ampliar. En el backend (Django/DRF) se verificó la distinción nítida entre modelos, serializers (validaciones de dominio), viewsets (endpoints y filtros) y permisos, previniendo dependencias circulares y lógica entrelazada. En el frontend (React) se verificó la separación entre páginas y componentes, la aplicación uniforme de Material UI/Recharts y la centralización de solicitudes a la API para evitar duplicaciones.

- **RNF-5 Fiabilidad**

Se verificó la fiabilidad del sistema a través de pruebas que ponían en compromiso las restricciones del sistema. Para ello se intentó crear categorías duplicadas o añadir cantidades negativas, pero todos los intentos fueron rechazados por parte del sistema que nos informaba de lo que estaba sucediendo. Se intentó borrar categorías con movimientos asociados pero el resultado fue el mismo.

- **RNF-6 Portabilidad**

Para llevar a cabo la verificación de la portabilidad, se probó la aplicación en diferentes navegadores y diferentes sistemas operativos. Para ello, se realizaron diferentes casos de uso en las diferentes plataformas, asegurando que la interfaz fuese de forma fluida y que las interacciones fuesen prácticamente idénticas en todas ellas.

Capítulo 7: Despliegue y guía de uso

Este capítulo está centrado en mostrar lo necesario para poder desplegar la aplicación en cualquier dispositivo. Además, se muestra una guía detallada para que el usuario final sepa exactamente cómo funciona y como debe relacionarse con el sistema para sacar el mayor rendimiento posible a la gestión de sus finanzas personales.

7.1. Requisitos básicos

Para que el sistema funcione de manera óptima, es necesario que el usuario disponga de unos requisitos mínimos indispensables, tanto a nivel hardware como a nivel software.

Para el adecuado funcionamiento y despliegue del sistema es necesario que se cumplan una serie de requisitos previos. A continuación, se detallan y enumeran los mismos:

Requisitos hardware:

- Conexión a internet de manera estable.
- 4GB de memoria RAM.

Requisitos de software:

- Sistema operativo: Windows, MacOS o GNU/Linux.
- Python 3.11+ o 3.12(recomendado).
- Node.js 18+ y npm 9+.
- Sistema operativo: Windows, Linux o MacOS.
- PostgreSQL 14+ (solo si se quiere usar Postgre, aunque para probar no es necesario).

7.2. Procedimiento de instalación

A continuación, se detalla paso a paso la implementación y configuración del sistema. Para ello se presupone la utilización de los requisitos anteriores:

Paso a paso detallado para la instalación y configuración del sistema. Este proceso presupone la instalación de los elementos detallados en la sección de requisitos previos.

1. **Obtención del proyecto**: Clonar el repositorio desde el siguiente enlace: https://github.com/noahramoss/TFG
   La estructura relevante es:

- *tfg_finanzas/:* proyecto Django, backend.
- movimientos/: app principal con modelos, serializers y vistas.
- *frontend/:* parte de React, frontend.

- requirements.txt: dependencias Python.
- *package.json:* dependencias frontend.

2. **Configuración del Backend con Django + DRF con SQLite**: Para ello vamos a optar por una ruta más sencilla utilizando SQLite en vez de PostgreSQL ya que es más sencillo para la instalación y prueba.

   Primero se crea y activa un entorno virtual
   python -m venv venv
   En macOS y Linux
   source venv/bin/activate
   En Windows
   .\venv\Scripts\Activate.ps1

   Se instalan las dependencias
   pip install --upgrade pip
   pip install -r requirements.txt

   Posteriormente las migraciones
   python manage.py migrate

   De manera opcional, aunque recomendada, se crea un super usuario para /admin
   python manage.py createsuperuser

   Finalmente, se arranca
   python manage.py runserver

   SQLite no necesita configurar nada, por lo que el archivo db.sqlite3 se crea automáticamente. El backend quedará disponible en http://127.0.0.1:8000/ y la administración de Django en http://127.0.0.1:8000/admin/ .

3. **Configuración del Frontend con React**:

   Se cambia al directorio del frontend
   cd frontend

   Se instala
   npm install

   De manera opcional, se configura la base de la API en .env
   REACT_APP_API_BASE=http://127.0.0.1:8000

   Finalmente, se ejecuta
   npm start

La aplicación se ejecuta en http://localhost:3000/ . Al iniciar sesión, el frontend guarda el token de sesión y lo reutiliza automáticamente en las llamadas a la API.

Como apunte final, cabe destacar que también se puede levantar el proyecto con PostgreSQL con una configuración similar creando las bases de datos, pero se ha optado por facilitar el proceso de implementación y prueba de la aplicación a los usuarios utilizando SQLite.

7.3. Guía detallada de uso

En este apartado se aporta al usuario final de la aplicación un manual para que una vez realizada la instalación del sistema disponga de una guía de ayuda para entender como es el funcionamiento de la propia aplicación destinada a la gestión de las finanzas personales.

Este capítulo proporciona una guía completa para el usuario final del sistema, detallando las características principales, los requisitos previos y las instrucciones para el uso del sistema. Este manual está diseñado para ayudar a los usuarios a entender y utilizar el gestor de gastos e ingresos personales y grupales de manera efectiva.

7.3.1. Características principales

La aplicación para la gestión de las finanzas personales tiene como propósito general cumplir con las siguientes premisas:

- Permitir el registro e inicio de sesión de los diferentes usuarios finales.
- Crear categorías personalizadas para la gestión de sus finanzas.
- Crear los movimientos deseados por los usuarios referentes a los ingresos o gastos propios y asignarse a la categoría que pertenezcan estos.
- Editar y eliminar un movimiento con plena libertad.
- Eliminar las categorías sin uso o libre de movimientos.
- Filtrar el listado de categorías.
- Filtrar, ordenar y paginar los diferentes movimientos según las opciones proporcionadas por la aplicación.
- Visualizar de forma sencilla los ingresos, gastos y balance (KPIs) de forma clara y según los filtros aplicados.
- Visualización de estadísticas por intervalos de tiempo y por categorías.

7.3.2. Registro e inicio de sesión en la aplicación

Al ejecutar la aplicación, como primera pantalla se muestra el inicio de sesión al sistema con la posibilidad de acceder a la zona de registro a través de un clic.

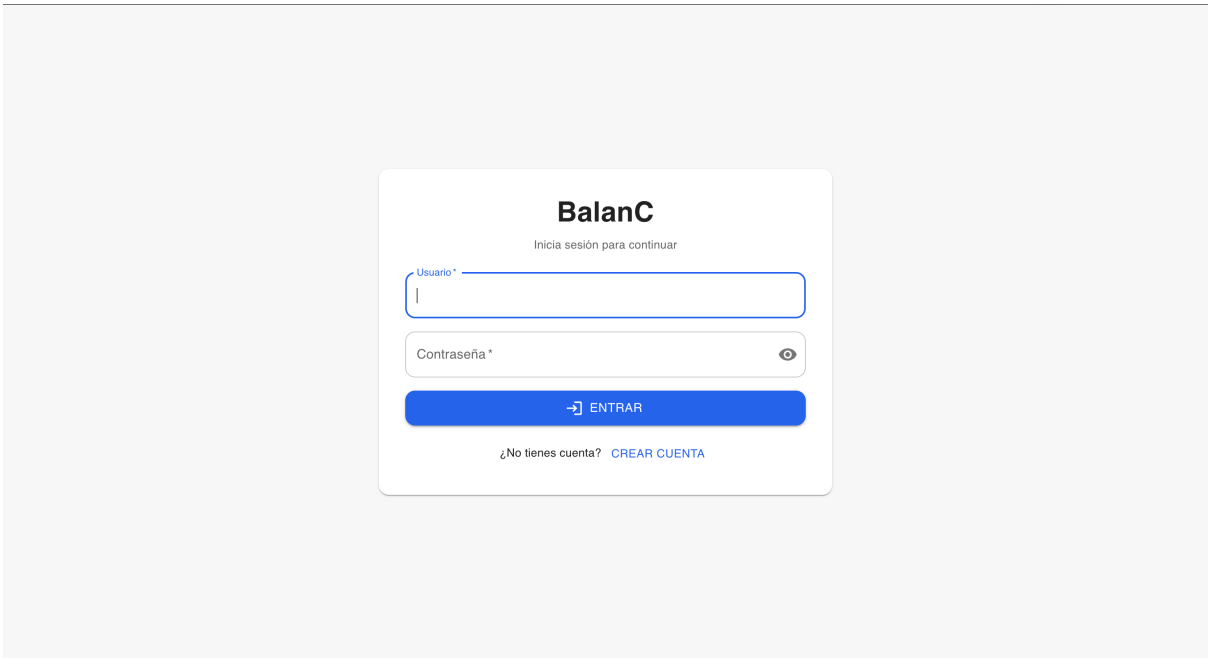


Figura 7.1: Pantalla de inicio de sesión

Para proceder con el registro es necesario clicar en *CREAR CUENTA*. Esto le enviará a la zona de registro para poder tener acceso a la aplicación. Para proceder con el registro es necesario que rellene de forma correcta los campos del formulario y una vez finalizado seleccione *CREAR CUENTA.*

Esto hará que inmediatamente pase a tener un usuario registrado en el sistema y se le redirija directamente a la página de Categorías, ya dentro de la aplicación.

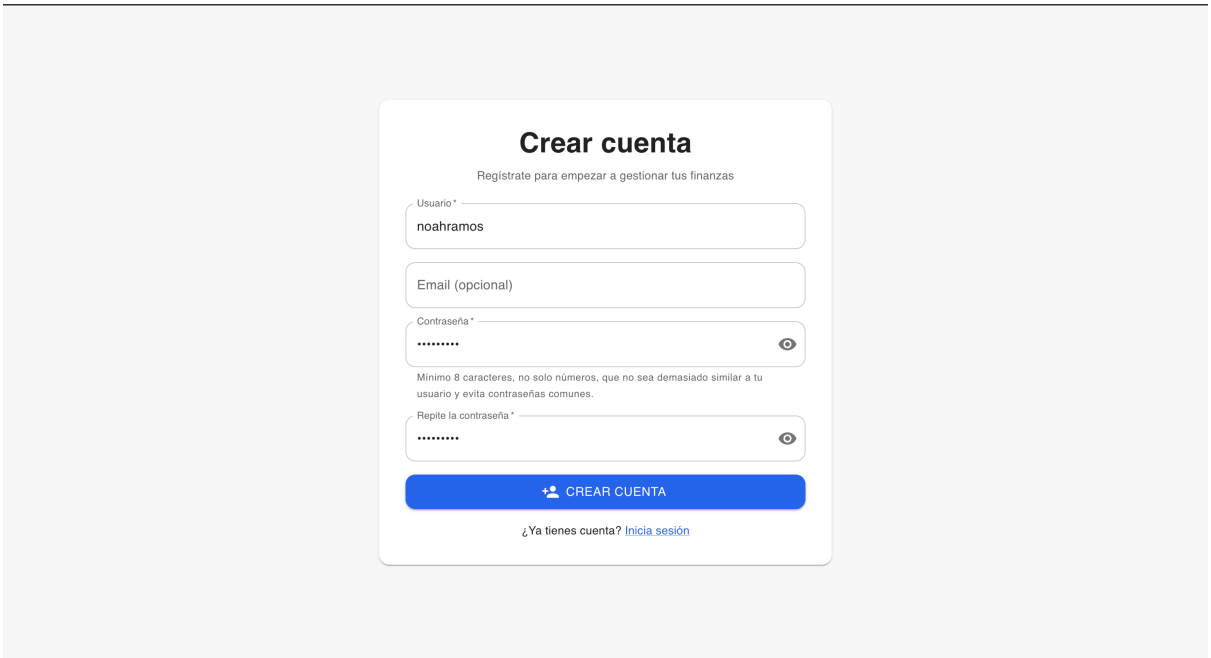


Figura 7.2: Registro de usuario

Para proceder con el registro es necesario clicar en *CREAR CUENTA*. Esto le enviará a la zona de registro para poder tener acceso a la aplicación. Para proceder con el registro es necesario que rellene de forma correcta los campos del formulario y una vez finalizado seleccione *CREAR CUENTA.*

Esto hará que inmediatamente pase a tener un usuario registrado en el sistema y se le redirija directamente a la página de *Categorías*, ya dentro de la aplicación.

7.3.3. Categorías

Una vez dentro, la primera vista que tenemos es la de Categorías, la cual en un primer instante se encuentra vacía a falta de que el nuevo usuario registre en ella las diferentes categorías de sus correspondientes finanzas.

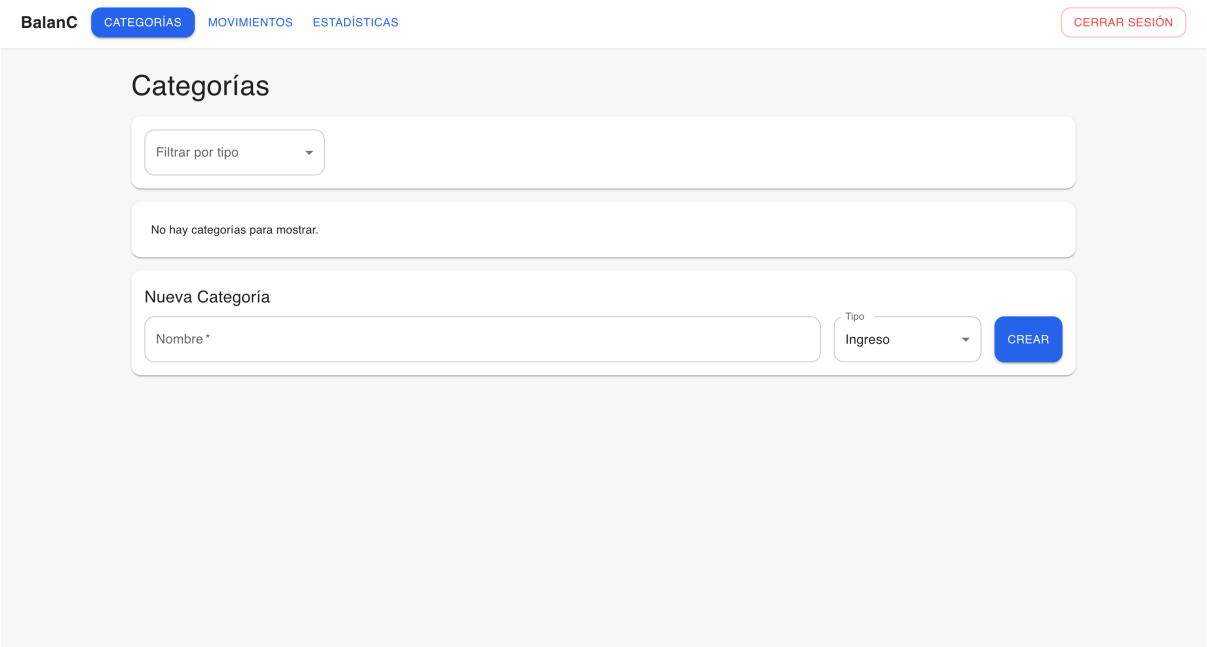


Figura 7.3: Vista Categorías

El usuario dentro de Categorías dispone de varias opciones, primero como es lógico vamos a la zona de creación de categorías. En ella observamos que disponemos de un recuadro para introducir el nombre de la categoría que deseamos crear y a su lado un desplegable para seleccionar el tipo, si se trata de un ingreso o un gasto. Una vez esto es está completado, seleccionando el botón *CREAR,* se creará la categoría y se añadirá a la lista que aparece justamente encima.

Una vez hemos creado nuestras categorías, vemos como aparecen en el listado, indicando el tipo al que pertenecen. A la derecha de cada Categoría vemos un icono de papelera, el cual podemos usar si deseamos eliminar una categoría que no vamos a usar. Cabe recalcar que dicha categoría solo podrá ser eliminada si no contiene ningún movimiento asociado.

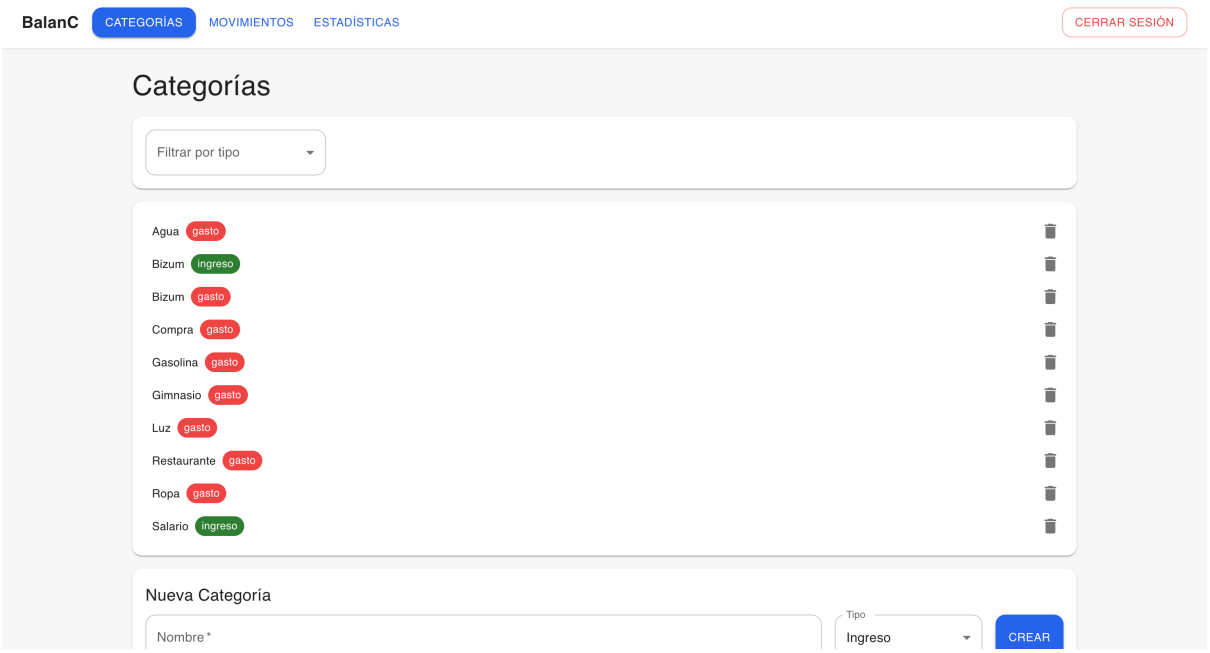


Figura 7.4: Listado de categorías con su icono de borrado.

Justo encima de la lista de categorías, vemos un filtro el cual si seleccionamos podemos filtrar el listado de categorías por todas, ingresos o gastos. Por ejemplo, si pinchamos y seleccionamos filtrar por ingresos se nos mostraría la con solo los ingresos.

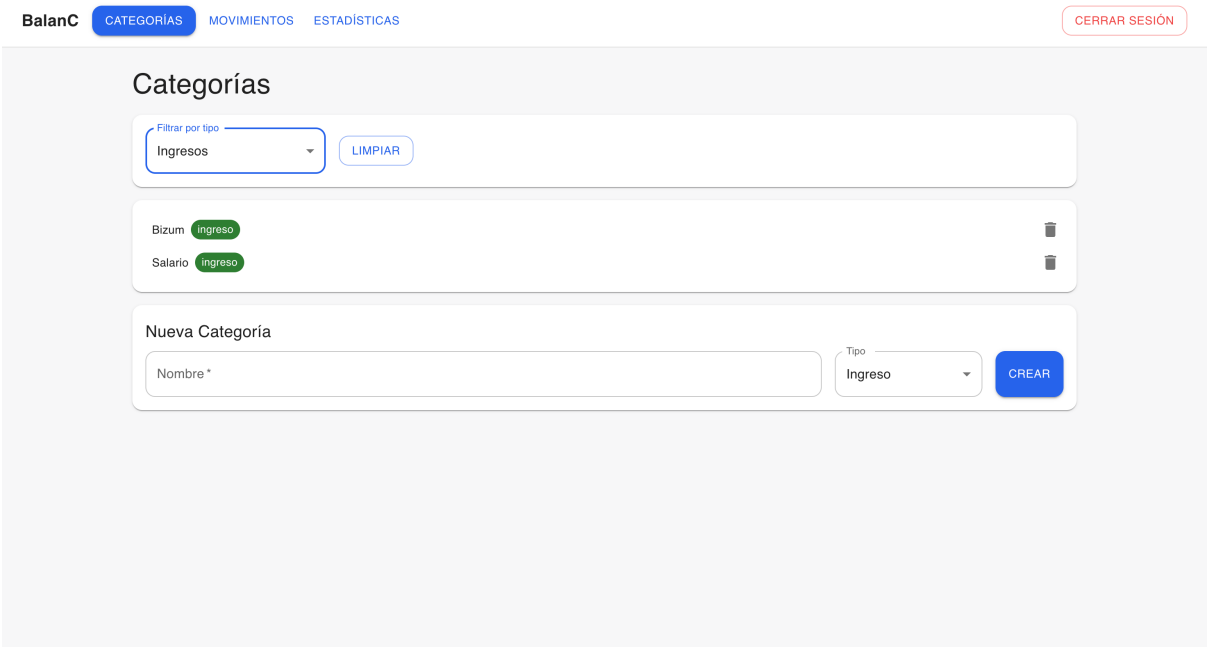


Figura 7.5: Listado de categorías filtrado por ingresos.

7.3.4. Barra de navegación

En la parte superior de la pantalla vemos una barra de navegación, la cual nos sirve para cambiar de vista. Si queremos acceder a Movimientos, Estadísticas o incluso Cerrar Sesión con el botón rojo de la derecha del todo. Además, vemos como Categorías está resaltado en azul, indicando así que nos encontramos actualmente en el apartado o vista de Categorías.

Figura 7.6: Barra de navegación.

7.3.5. Movimientos

Si seleccionamos el apartado de *MOVIMIENTOS* en la barra de navegación cambiaremos de vista y como su propio nombre indica nos encontraremos en el apartado de Movimientos.

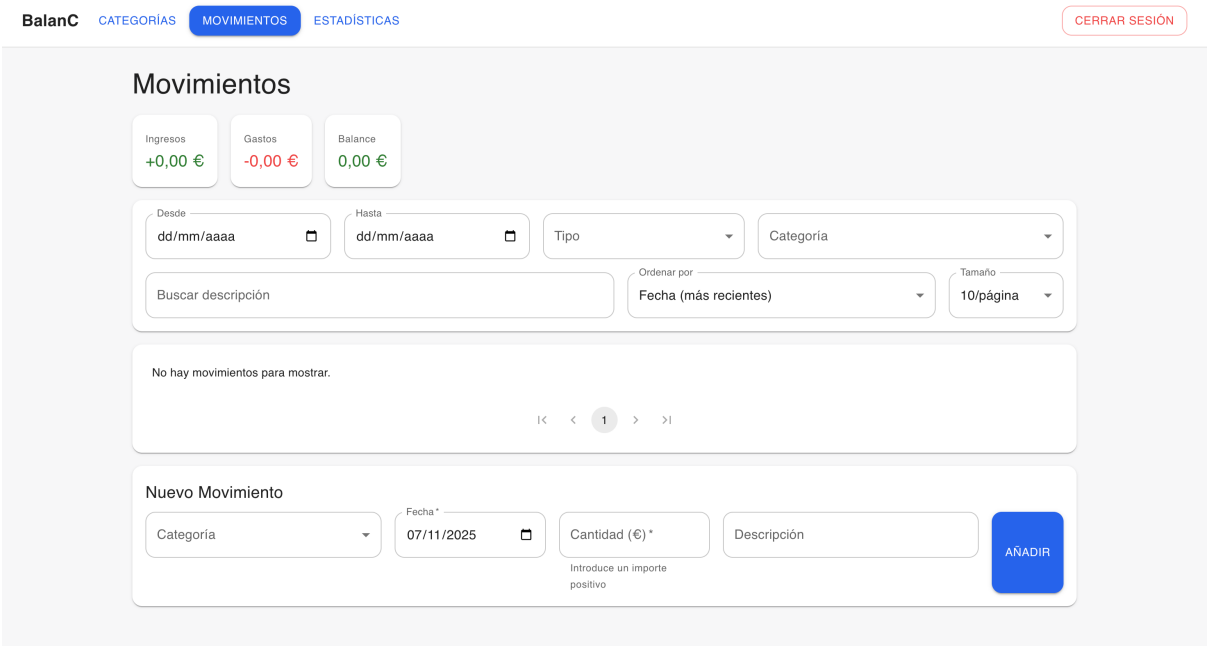


Figura 7.7: Vista Movimientos.

Como podemos ver en la figura anterior, así se ve el apartado de Movimientos cuando todavía no hemos añadido ninguno. Tal y como pasaba en Categorías, vemos como la pantalla está dividida en tres, teniendo en la parte superior los filtros, en el centro la zona de la lista de movimientos, en este caso vacía, y debajo la zona para crear nuevos movimientos. A diferencia de la sección de Categorías, vemos que encima de los filtros aparecen unos valores que nos indican los ingresos, los gastos y el balance. Estos nos van a mostrar de forma clara y concisa el saldo del que disponemos, lo que hemos ingresado y lo que hemos gastado.

En la zona de añadir movimientos, vemos como para crear un movimiento debemos añadir la categoría, la cual podemos seleccionar entre las disponibles, la fecha, que se puede poner de forma manual o seleccionando en el calendario, que por defecto nos muestra la del día actual para que sea más cómodo a la hora de añadir movimientos, la cantidad, la cual se nos indica previamente que debe ser un importe positivo y la descripción, que es opcional pero bastante recomendable para cuando queramos buscar movimientos por su descripción.

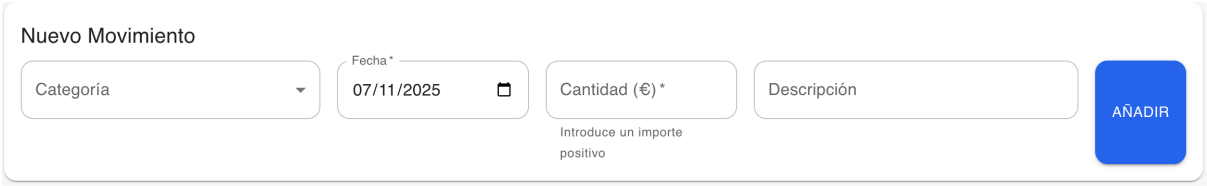


Figura 7.8: Nuevo Movimiento.

Una vez hemos completado los diferentes campos para el registro del movimiento, vemos como justamente encima nos aparece el listado de los movimientos existentes entre los cuales se puede ir navegando para verlos todos, ya que si aparecieran todos de golpe nos mostraría una página demasiado extensa.

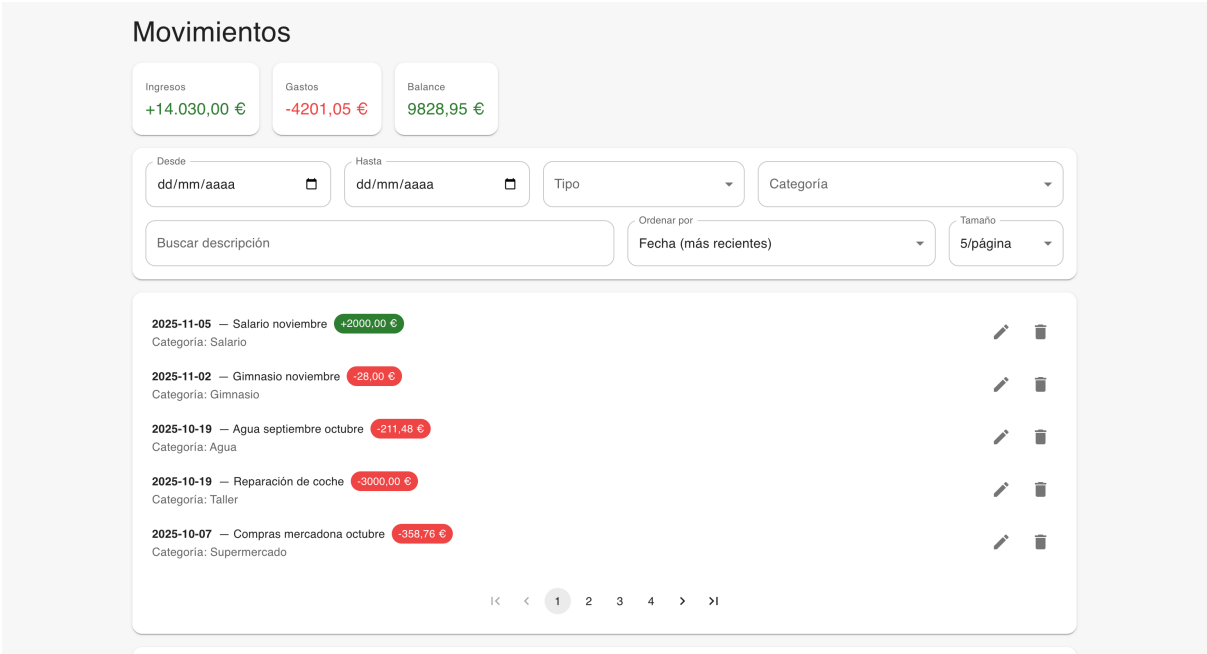


Figura 7.9: Listado de Movimientos.

Una de las herramientas más potentes que tiene este apartado es la zona de filtrado. Como normalmente tendremos muchos movimientos y con el paso del tiempo más todavía es de

vital importancia que podamos filtrar movimientos según nos convenga. Para ello como vimos anteriormente, podemos filtrar por fechas, tipo, categoría, realizar una búsqueda a través de la descripción e incluso ordenar la aparición de los movimientos.

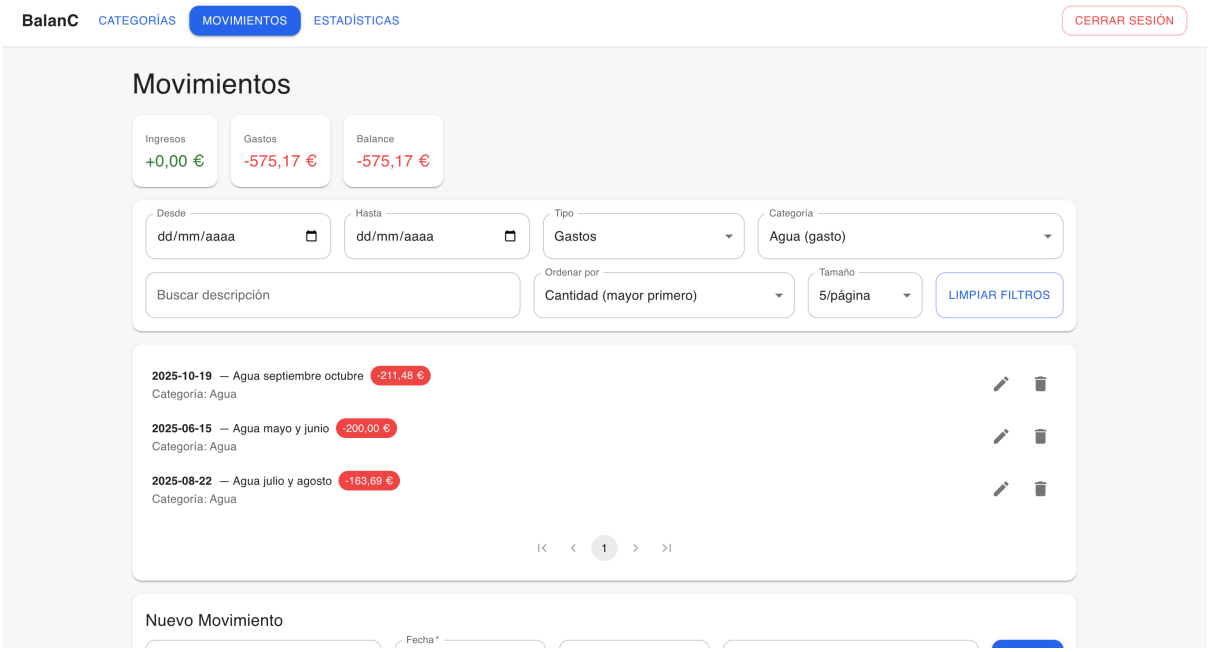
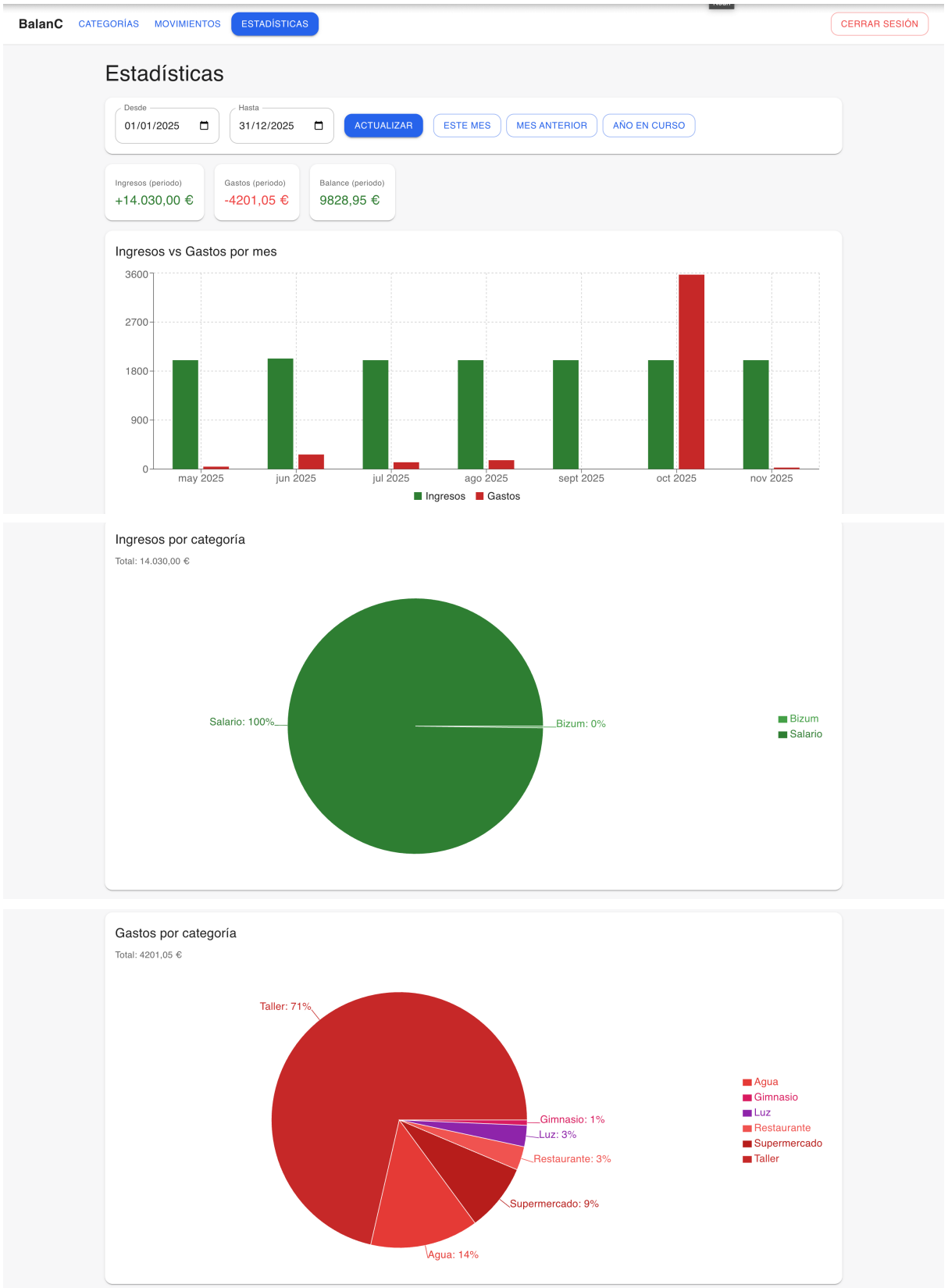


Figura 7.10: Filtrado de Movimientos.

Como vemos en la figura anterior, los diferentes filtros podemos combinarlos como nos resulte más conveniente para encontrar los movimientos que estemos buscando. Cabe destacar que los valores de los datos relevantes de ingresos, gastos y balance también se modifican según el filtrado que esté aplicado en ese momento.

7.3.5. Estadísticas

En el apartado de *ESTADÍSTICAS* nos encontramos con la parte más visual de la aplicación, en la que el usuario puede ver de forma clara a través de los gráficos como está gestionando su dinero. Para ello se muestran dos estilos de gráficas diferentes, una primera de barras que relaciona los ingresos y los gastos según el mes del año, y otro gráfico que muestra los ingresos y los gastos por categorías para mostrar al usuario de forma más directa que le proporciona más ingresos y en qué exactamente se está gastando su dinero. Aquí, al igual que en la parte de movimientos tenemos los valores de ingresos, gastos y balance claros y a la vista, los cuales, al igual que las gráficas, se modifican con los filtros de periodos de tiempo deseados. Esto se puede ajustar para un rango de fechas deseado, por el año natural, por el mes actual o por el mes anterior.



Figuras 7.11, 7.12, 7.13: Vista de Estadísticas

Por ejemplo, si queremos filtrar por el mes que más gastos se observa que hay, mes de octubre, obtendríamos los siguientes resultados.

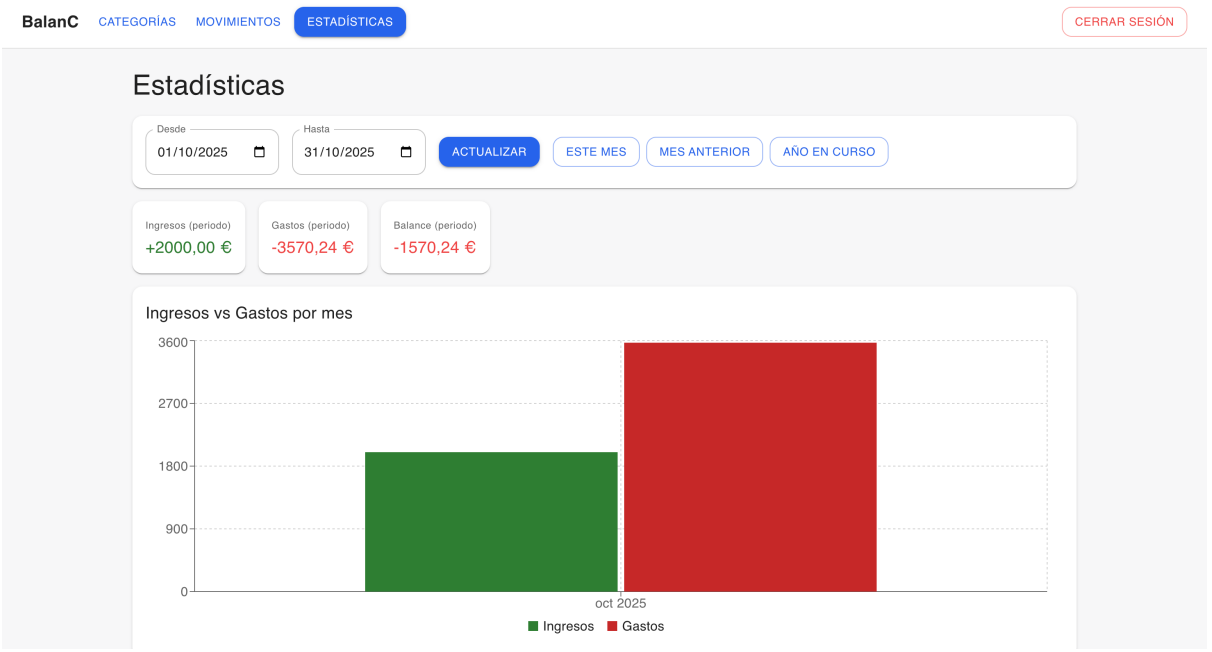


Figura 7.14: Filtrado de Estadísticas.

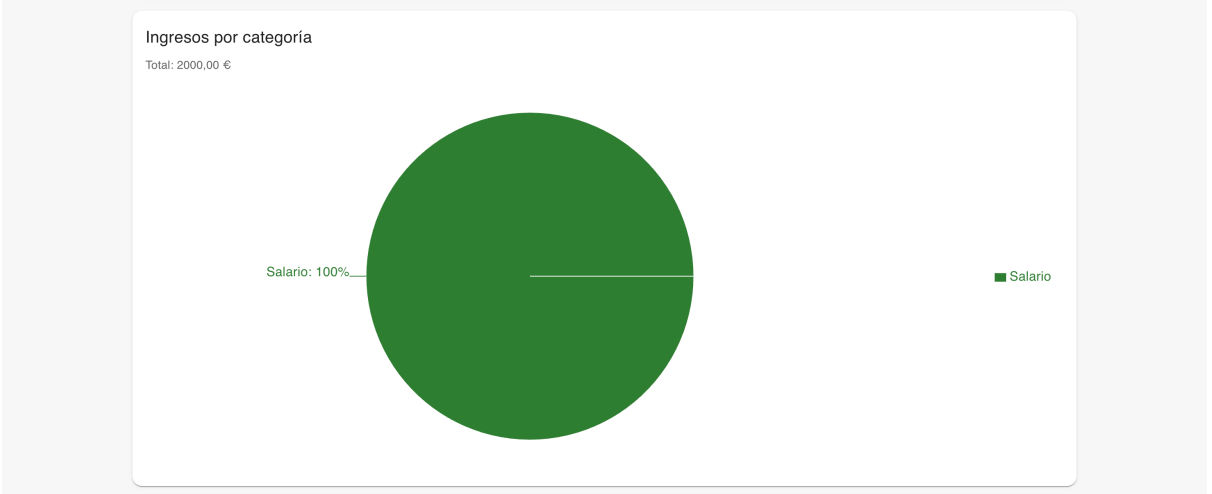


Figura 7.15: Filtrado de Estadísticas.

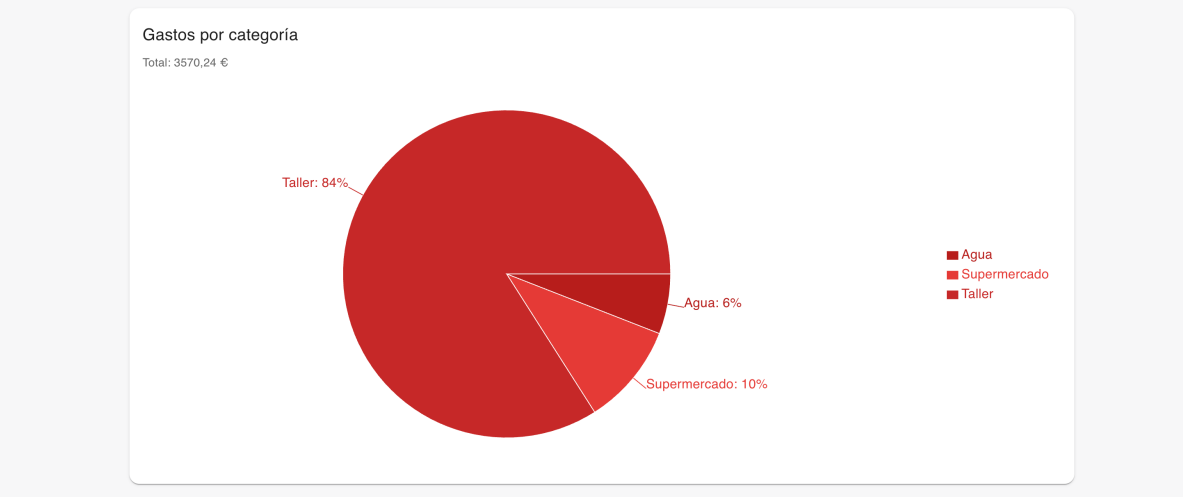


Figura 7.16: Filtrado de Estadísticas.

Una vez hemos terminado de registrar, analizar y visualizar nuestras finanzas, cuando lo deseemos y desde donde nos encontremos podemos cerrar sesión y así concluir con el uso de la aplicación.

Capítulo 8: Conclusiones

Como análisis tras la finalización del proyecto, este capítulo sintetiza los logros alcanzados y las experiencias acumuladas durante el desarrollo de la aplicación web de gestión de finanzas personales. Asimismo, se establece un margen de mejorar para el cual la aplicación pueda ampliar su alcance y refinar su calidad en futuras iteraciones.

Los objetivos planteados en el primer capítulo han sido cumplidos. Se ha entregado una solución funcional que posibilita el registro de ingresos y gastos, la personalización de categorías y la consulta de resúmenes y KPIs por periodo, todo ello accesible desde una interfaz web creada en React que consumen una API REST implementada con Django y Django REST Framework sobre una base de datos SQL construida en PostgreSQL. Más allá del resultado conseguido, el proyecto ha supuesto una mejora notable en la comprensión de los fundamentos de la arquitectura cliente-servidor, el diseño de APIs coherentes y la aplicación de reglas de negocio para asegurar la integridad de los datos a través de restricciones que impiden inconsistencias lógicas (importes positivos o unicidad de categorías por nombre, usuario y tipo).

En términos tecnológicos, la elección de Django para el backend y React para el frontend ha aportado robustez y productividad. Si bien es cierto que estas herramientas conllevan una curva de aprendizaje, el resultado ha sido un sistema modular y extensible.

Respecto a las líneas de trabajo futuras, existen diversas opciones que permitirían ampliar el valor de la herramienta. Se podrían incorporar presupuestos por categoría y periodo, con seguimiento de cumplimientos y alertas al acercarse o exceder límites. También se podría introducir movimientos recurrentes para evitar que los usuarios tengan que realizar acciones repetitivas. Una mejora obvia que se podría añadir sería el despliegue de dicha aplicación para que esta no funcione únicamente en local y la posibilidad de expandir la aplicación a diferentes dispositivos, como los móviles.

En un contexto actual y con cada vez más presente y futuro, donde la inteligencia artificial evoluciona con rapidez, el proyecto deja una base sólida sobre la que poder incorporar capacidades de predicción y personalización sin alterar las bases del sistema.

En conclusión, el trabajo ha cristalizado en una solución funcional, clara y extensible que demuestra la viabilidad del enfoque propuesto y sienta las bases para una evolución natural hacia lo comentado anteriormente. El aprendizaje obtenido, constituye, quizás, el resultado más valioso de este proceso, el cual será de una enorme utilidad en el futuro profesional que me depara.

Bibliografía

[1] Django Software Foundation. (s. f.). Django documentation. Django Project. https://docs.djangoproject.com/

[2] Tom Christie et al. (s. f.). Django REST Framework. https://www.django-rest-framework.org/

[3] Meta. (s. f.). React – Documentation. https://react.dev/

[4] The PostgreSQL Global Development Group. (s. f.). PostgreSQL Documentation. https://www.postgresql.org/docs/

[5] Axios contributors. (s. f.). Axios — Promise based HTTP client. https://axios-http.com/

[6] Auth0 Inc. (s. f.). JSON Web Tokens (JWT) Introduction. https://jwt.io/introduction

[7] Jones, M., Bradley, J., & Sakimura, N. (2015). RFC 7519: JSON Web Token (JWT). IETF. https://www.rfc-editor.org/rfc/rfc7519

[8] MUI. (s. f.). Material UI — React components. https://mui.com/

[9] Recharts. (s. f.). Recharts — A composable charting library for React. https://recharts.org/

[10] Python Software Foundation. (s. f.). Python 3 Documentation. https://docs.python.org/3/

[11] OpenJS Foundation. (s. f.). Node.js. https://nodejs.org/

[12] npm, Inc. (s. f.). npm | Home. https://www.npmjs.com/

[13] Git–SCM. (s. f.). Git Documentation. https://git-scm.com/doc

[14] GitHub. (2025). GitHub: Let's build from here. https://github.com/

[15] Postman. (s. f.). Postman API Platform. https://www.postman.com/

[16] diagrams.net. (s. f.). draw.io — free flowchart maker and diagrams online. https://app.diagrams.net/

[17] Google. (s. f.). Google Drive — Plataforma de almacenamiento personal en la nube. https://workspace.google.com/intl/es/products/drive/

[18] Google. (s. f.). Google Meet. https://meet.google.com/

[19] Adam Johnson y colaboradores. (s. f.). django-cors-headers. https://github.com/adamchainz/django-cors-headers

[20] MDN Web Docs. (s. f.). HTTP access control (CORS). https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[21] MDN Web Docs. (s. f.). Fetch API. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

[22] OWASP. (s. f.). Authentication Cheat Sheet. https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

[23] OWASP. (2021). OWASP Top 10: Security Risks. https://owasp.org/www-project-top-ten/

[24] Jazzband. (s. f.). python-decouple — Strict separation of settings from code. https://github.com/jazzband/python-decouple

[25] psycopg contributors. (s. f.). psycopg — PostgreSQL database adapter for Python. https://www.psycopg.org/

[26] React Training. (s. f.). React Router. https://reactrouter.com/

[27] Atlassian. (s. f.). Diagramas de Gantt | Atlassian. https://www.atlassian.com/es/agile/project-management/gantt-chart

[29] You Need A Budget LLC. (s. f.). YNAB — You Need A Budget. https://www.ynab.com/

[30] SPENDEE a.s. (s. f.). Money Manager & Budget Planner | Spendee. https://www.spendee.com/

[31] Fintonic Servicios Financieros, S.L. (s. f.). Fintonic https://www.fintonic.com/es-ES/inicio/