ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE
INGENIERÍA INFORMÁTICA

Aplicación de gestión de gastos

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Cristina Falcón Carqué

TUTORIZADO POR:

Miguel Alemán Flores

Fecha [Junio/2024]

Resumen

La aplicación de Gestión de Gastos permite a los usuarios registrar los gastos de todas sus cuentas. Además, podrán añadir los gastos en efectivo, facilitándoles llevar la gestión global de todos sus gastos y no solo los de una cuenta bancaria, como ocurre en el caso de las aplicaciones bancarias.

Además, con la sección de gráficas los usuarios podrán analizar de una manera más visual cuál es su patrón de gastos a lo largo del tiempo y su evolución.

Asimismo, los usuarios también podrán agregar sus ingresos, lo que les permitirá tener una visión completa del balance de su economía.

ABSTRACT

The Expense Management application allows users to register expenses for all of their accounts. Furthermore, they can also add cash expenses, making it easier to manage all their expenses globally, and not only those related to bank accounts, as in the case of banking applications.

In addition, with the graphs section, users will be able to analyse their spending pattern over time and evolution in a more visual way.

Users will also be able to add their incomes, which will allow them to have a complete view of the balance of their finances.

Contenido

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

INTRODUCCIÓN

Los gastos forman parte del día a día de las personas, por lo que llevar a cabo una gestión eficiente de estos es fundamental. Muchas veces ocurre que se posee el dinero en efectivo, se realizan gastos y luego no cuadran las cuentas. Con la aplicación Gestión de Gastos se solventan esas situaciones de descontrol financiero, registrando tanto los gastos en efectivo como los realizados con tarjeta, pudiendo así observar el total conjunto de los gastos, ingresos y saldo remanente.

Con un buen control se pueden identificar las tendencias y analizar en dónde se gasta más y dónde se podrían reducir gastos, motivando al usuario a mejorar su economía.

Según un estudio del Instituto Nacional de Estadística en 2023 “el 9,3% de la población manifestó llegar a fin de mes con mucha dificultad”, y un “37,1% no tuvo capacidad para afrontar gastos imprevistos” (INE - Instituto Nacional de Estadística, 2024). Definiendo un presupuesto mes a mes y teniendo en cuenta que los gastos imprevistos suceden, se podrían definir unas reglas de ahorro para obtener un “colchón financiero” que permita afrontar dichos gastos cuando ocurran.

Con Gestión de Gastos los usuarios podrán definir un presupuesto, limitando los gastos según las categorías que definan, permitiéndoles ser conscientes del progreso de estos.

Una de las razones que me motivó para llevar a cabo esta aplicación fue ver a gente de mi entorno anotar sus gastos en Excel, lo que conllevaba tener que crear o duplicar una hoja de cálculo para cada mes, así como tener que establecer fórmulas para calcular el total de los gastos, tarea que además se hace más compleja para aquellos usuarios que no poseen los conocimientos previos y necesitan invertir tiempo en aprender a utilizarlo. Gestión de Gastos posee una interfaz sencilla y fácil de usar para usuarios con un nivel básico en conocimientos tecnológicos, permitiéndoles llevar una gestión de sus finanzas sin la necesidad de tener que preocuparse por si están realizando bien las fórmulas para calcularlas.

DESCRIPCIÓN DEL ROYECTO

El desarrollo de este proyecto se ha divido en dos partes. Por un lado, un *backend* desarrollado en Node JS con Express, y por otro lado, un *frontend* desarrollado con React-native.

En el *backend* se implementa el servidor que dará soporte para el funcionamiento de la aplicación. En él se manejan las solicitudes API RESTful provenientes del *frontend* a través de *endpoints* que permitirán las operaciones CRUD (*Create, Read, Update, Delete*) de los diferentes apartados que contiene la APP (cuentas, gastos, ingresos, etc.).

Asimismo, desde el *backend* también se controla la conexión a la base de datos, en este caso, de MongoDB a través de la biblioteca de Mongoose. Esta biblioteca cuenta con funcionalidades útiles entre las que destacan la creación de esquemas para modelar los datos, la opción de castear los datos según el tipo, añadir validaciones, y otras que serán desarrolladas más adelante.

La implementación de autenticación y autorización al acceso de los datos se realizó con JWT (JSON Web Tokens) para controlar que la aplicación sea segura y los datos solo son accesibles para los usuarios registrados que corresponda.

En la parte del *frontend* se ha desarrollado la interfaz de la aplicación, y es desde donde se realizan las llamadas API mediante la librería Axios para realizar las peticiones.

La combinación de estas partes permite obtener una aplicación completa en la que el usuario podrá realizar las siguientes funcionalidades: registro e inicio de sesión, gestión de cuentas, categorías, gastos e ingresos, exportación de datos, establecimiento de límites de gastos, visualización de gráficas, transferencias entre cuentas y conversión de divisa.

ESTADO ACTUAL

Para comprender un poco mejor el contexto de las aplicaciones de gestión de gastos, se ha realizado una investigación sobre el panorama actual de las aplicaciones más demandadas, sus funcionalidades, las tendencias y el estado actual. Se han buscado aplicaciones en Google Play que cumplan con la gestión de gastos. Dado que existe una gran cantidad de aplicaciones, se expondrán 3 que están entre las más populares.

"Monefy" desarrollada por Reflective Technologies con más de 5 millones de descargas y una puntuación de 4,5 estrellas de 188 mil reseñas. Entre sus funcionalidades principales se encuentran: manejo de divisas e historiales.

"Gestor de gastos – finanzas" de Horoscope365, más de 1 millón de descargas y una puntuación de 4,9 estrellas de 52,8 mil reseñas. Esta aplicación permite establecer presupuestos, obtener análisis y posee una calculadora integrada.

"Registro Contable" de Realbyte Inc. con más de 10 millones de descargas y una puntuación de 4,8 estrellas de 393 mil reseñas. Esta aplicación es más compleja que las anteriores ya que, además cuenta con una aplicación de escritorio. Esta aplicación cuenta con calendario con el registro de los gastos, diversidad de gráficos y permite incluir imágenes en el registro de gastos.

Estas aplicaciones representan una parte del estado actual de aplicaciones de gestión de gastos, cada una con sus diferencias y similitudes. Al analizar las características de estas aplicaciones, se pueden identificar nuevas ideas y proponer soluciones.

En algunas de las reseñas, los usuarios realizaban sugerencias sobre mejoras por lo que se tuvieron en cuenta para posibles funcionalidades, entre ellas, varias coincidían en incluir gráficos o mejoras de estos para la visualización de datos.

Una de las funcionalidades que no se observa en ninguna o no se mencionan, es la posibilidad de registrarse como usuario, en el caso de este proyecto, permitiría al usuario tener los datos al alcance de la mano, simplemente iniciando sesión desde el dispositivo donde desee acceder.

Además, se hará especial hincapié, en implementar una gran cantidad de gráficas para visualizar los datos agrupados de diferentes formas, comparaciones, etc.

OBJETIVOS

BASE DE APRENDIZAJE

Las asignaturas que han servido como base de aprendizaje para realización de este trabajo han sido:

- Metodología del Desarrollo Ágil: se ha utilizado como base para llevar una metodología ágil a la hora de desarrollar el proyecto facilitando la organización de este.
- Diseño de Interfaces de Usuario: gracias a los conocimientos adquiridos en esta asignatura, se ha llevado a cabo la identificación de las personas, los escenarios, así como el diseño de un sketch para la interfaz.
- Ingeniería de Requisitos: ha servido de base para la realización del diagrama de casos de uso, identificación y desarrollo de requisitos, y reconocer los stakeholders.
- Gestión del Software: el desarrollo de las historias de usuario, así como sus criterios de aceptación, estimaciones de tiempo y priorización, la identificación de riesgos y el modelado de la aplicación son los contenidos de esta asignatura que han sido aplicados en este proyecto.

COMPETENCIAS CUBIERTAS

En este proyecto se han cubierto algunas de las competencias de ingeniería del software que se desarrollan en el documento “Objetivos y competencias” del GII (Plan 40).

*“**IS01** – Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.”*

Durante la elaboración se aplicaron los conocimientos adquiridos a lo largo de la carrera, entre ellos, el empleo de metodologías ágiles, permitiendo una gestión eficiente a lo largo del desarrollo del software.

*"**IS02** – Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones."*

Aunque no haya definido un cliente, el tutor actuó como tal, aportando su opinión sobre los requisitos existentes y sugiriendo nuevas ideas que fueron valoradas y de las cuales se especificaron los requisitos correspondientes.

*"**IS04** - Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales."*

A lo largo del desarrollo se analizaron e identificaron problemas que fueron resueltos de manera eficiente. Asimismo, se emplearon pruebas para comprobar que dichos problemas habían sido abordados correctamente.

*"**IS06** – Capacidad para diseñar soluciones apropiadas en uno o más dominios de aplicación utilizando métodos de la ingeniería del software que integren aspectos éticos, sociales, legales y económicos"*

Considerando aspectos legales y éticos entre otros, se han empleado recursos gratuitos para evitar problemas de derechos de autor y se generaron imágenes de personas ficticias para la representación de posibles perfiles de la aplicación

OBJETIVOS DEL PROYECTO

1. **Aprendizaje de nuevas tecnologías**: aprender nuevos lenguajes y tecnologías como son Node JS, React Native, MongoDB y poder aplicar los conocimientos adquiridos.
2. **Autonomía y desarrollo individual**: desarrollar una aplicación desde el principio y en solitario, aplicando los conocimientos que se han obtenido a lo largo de la carrera universitaria.
3. **Implementar una aplicación móvil real** que cumpliera con lo siguiente:
    - Registro y categorización de gastos.
    - Compatibilidad con diversas monedas.
    - Gestión de categorías.

- Administración del dinero y presupuestos.
- Realizar operaciones con la calculadora integrada.
- Seguimiento de la evolución de los gastos mediante la generación de informes y gráficas.
- Aplicación intuitiva y fácil de usar.

METODOLOGÍA

Una vez definida la idea del proyecto, se planificó como se llevaría a cabo:

1. Estudio previo: en primer lugar, se investigó sobre las aplicaciones existentes, así como las necesidades de los usuarios, perfiles de personas que utilizarían la aplicación y qué tecnologías serían las más adecuadas para llevarlo a cabo.
2. Análisis: previo al desarrollo se definió una pila de producto inicial, se identificaron los requisitos y se realizó un modelo conceptual para identificar las entidades y como se relacionarían entre ellas.
3. Diseño: se diseñó un sketch para tener una idea inicial de qué apartados y funcionalidades contendría la aplicación. Además, también se planteó como sería el diseño de la arquitectura, los datos y los módulos.
4. Desarrollo e implementación: se utilizó una metodología ágil para ello, en este caso, SCRUM, "*Es un marco interactivo e incremental que promueve la comunicación efectiva entre los miembros del equipo, fomenta la adaptabilidad y proporciona la capacidad de responder rápidamente a los cambios.*" (Time Solutions Ltd). Se eligió esta metodología ya que es la que principalmente se ha estudiado en la carrera además de ser de uno de los más populares, utilizado por un 56% de las empresas, y un 86% si contamos con las que aplican SCRUM de forma híbrida, según (Petrova, 2019). Dadas las características especiales de un TFG, se han adaptado los elementos de Scrum a las circunstancias específicas.
5. Pruebas / Verificación / Validación: se realizaron llamadas APIs con Postman para comprobar que el código implementado funcionaba como se esperaba, y se comprobó que se cumplían los requisitos establecidos.
6. Documento / Presentación: desarrollo de la memoria con la información del proyecto, y la presentación con un resumen de los datos más relevantes.

TECNOLOGÍAS

HERRAMIENTAS DE ESCRITORIO

En esta sección se definirán las herramientas que han sido necesarias instalar en el PC, para llevar a cabo el proyecto.

Excel

Microsoft Excel es el programa de software de hojas de cálculo líder en el sector y una herramienta avanzada de análisis y visualización de datos (Microsoft, s.f.).

Se empleó una metodología de desarrollo ágil para el proyecto, por lo que se utilizó una plantilla de Excel para seguir el tiempo estimado y el tiempo empleado en las historias de usuario.

Visual Studio Code

Visual Studio Code es un editor de código redefinido y optimizado para crear y depurar aplicaciones web y en la nube modernas. (Visual Studio Code, 2021).

Con esta herramienta se ha desarrollado el código del proyecto, tanto de la parte de back como la del front.

Postman

Postman es una plataforma de API para crear y utilizar API. Postman simplifica cada paso del ciclo de vida de las API y agiliza la colaboración para que pueda crear mejores API, más rápido (What is Postman?

Postman API Platform, 2023).

En este caso ha sido empleada para testear previamente las APIs implementadas para su posterior uso desde el frontend. Asímismo, se ha utilizado para exportar un documento en el que se detalla toda la información referente a cada API.

Teams

Microsoft Teams es una aplicación de colaboración creada para el trabajo híbrido para que usted y su equipo estén informados, organizados y conectados, todo en un mismo lugar (Soporte técnico de Microsoft, s.f.).

Se ha empleado para realizar las reuniones de seguimiento con el tutor.

Herramientas web

Por otro lado, tenemos las herramientas que no han sido necesarias instalarlas ya que se puede hacer uso de ellas desde la misma web.

StoryboardThat

StoryboardThat es una herramienta con la que se han generado algunos de los escenarios de la aplicación.

Ofrece un plan gratuito que es el que se ha usado y permite la creación de dos guiones gráficos por semana con la opción de 3 celdas.

MockFlow

MockFlow es una herramienta de diseño con la que se ha creado el sketch[1] digital para tener una idea previa de la estructura de la interfaz de la aplicación.

MERN Stack

MERN es un stack tecnológico que incluye MongoDB, Express JS, React JS y Node JS, por lo que abarca todo el desarrollo de la aplicación desde la parte del servidor hasta la del cliente.

Mongo DB

Se ha utilizado para almacenar los datos. Se trata de una base de datos no relacional, que utiliza documentos en lugar de tablas.

Express JS

Es un framework de backend que se utiliza junto con Node JS. Facilita el manejo de peticiones HTTP, así como las validaciones de los datos recibidos, etc.

React Native

Es un framework que permite el desarrollo de aplicaciones móviles tanto para Android como para IOS.

En este caso la aplicación que se ha desarrollado solo ha sido para Android, aunque

[1] Boceto hecho a mano

adaptando el código existente, se podría implementar también para IOS.

Node JS

Es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación JavaScript. (Wikipedia, 2024)

JavaScript

JavaScript es un lenguaje de secuencias de comandos que te permite crear contenido de actualización dinámica, controlar multimedia y animar imágenes (¿Qué es JavaScript? | MDN, s.f.).

Familiarización con las herramientas

Node JS, MongoDB y Postman

Puesto que no se contaba con ninguna experiencia con Node JS, se realizó un curso impartido por Fernando Herrera (Herrera, 2022) en el cual se explicaban la creación de un backend server, servicios REST, Json web tokens, entre otros, los cuales fueron clave para el desarrollo de este proyecto.

Asimismo, se explicaba cómo realizar la conexión con la base de datos MongoDB y cómo probar el correcto funcionamiento de las APIs implementadas con Postman.

React native y Simulador Android

De este framework se tenían conocimientos previos ya que en las prácticas curriculares se realizó una aplicación con ello, por lo que no fue necesario realizar

una formación anterior. No obstante, se utilizó la documentación oficial para realizar consultas. (React Native, s.f.).

Herramientas web

En este caso, la mayoría ya habían sido utilizadas en otras asignaturas de la carrera por lo que no requirieron familiarización con ellas. A excepción de MockFlow, pero es muy similar a otras aplicaciones de escritorio que se han empleado para crear prototipos a lo largo de la carrera, por lo que no resultó difícil su uso.

ANÁLISIS

STAKEHOLDERS

Los principales stakeholders de esta aplicación serán usuarios hispanohablantes en un principio, de edades jóvenes o adultas que tengan acceso a dinero y necesiten gestionar sus finanzas. Serán los que interactúen con la aplicación teniendo una implicación directa en esta y pudiendo aportar opiniones o sugerencias, reporte de errores o consultas.

Asimismo, se podría identificar otro rol para el tipo de usuario que sería el usuario *tester* o beta que tendría acceso previo a los lanzamientos y se encargaría de testear activamente todas las funcionalidades de la aplicación, para posteriormente dar una retroalimentación de su experiencia.

Otro tipo de rol de stakeholder sería la autoridad, identificando dos roles para este caso como son la universidad y el tutor. Por un lado, el tutor se implicaría a través de reuniones de seguimiento periódicas, aportando su conocimiento técnico sobre el proceso en la implementación de la aplicación y evaluación de esta. Y, por otro lado, la universidad estaría implicada aportando plazos de entrega, documentación que se debe aportar, y la evaluación final del proyecto.

Aunque ahora mismo no es el caso, en un futuro los bancos podrían ser *stakeholders* si se realizase una sincronización automática de los datos de las cuentas bancarias.

Por último, el desarrollador con el rol de Ingeniero Informático que tendrá implicación directa, desarrollando e implementando todo el proceso del proyecto y la aplicación, con la responsabilidad de que la aplicación funcione bien.

TIPOS

Nombre Usuarios	
Descripción Interactúan con la aplicación	*Representantes* Usuarios finales, usuarios beta o testers

TABLA 1 - STAKEHOLDERS. TIPO: USUARIOS

Nombre Autoridad	
Descripción Validar la aplicación	*Representantes* Tutor, Universidad

TABLA 2 - STAKEHOLDERS. TIPO: AUTORIDAD

Nombre Desarrollador	
Descripción Implementar la aplicación	*Representantes* Ingeniero informático

TABLA 3 - STAKEHOLDERS. TIPO: DESARROLLADOR

ROLES

Nombre Usuario final	
Descripción	Es el usuario que más interactúa con la aplicación, utilizando todas o casi todas sus funcionalidades.
Responsabilidades	Ofrecer retroalimentación cuando surge un problema, hacer sugerencias, aportar su experiencia con la aplicación.
Implicación	Implicación directa, con el uso diario de la aplicación

TABLA 4 - STAKEHOLDERS. ROL: USUARIO FINAL

Nombre Usuario beta o tester	
Descripción	Es el usuario que accede con antelación a las nuevas funcionalidades y se encarga de utilizar la aplicación y comprobar si surgen errores.
Responsabilidades	Probar de manera activa todas las funcionalidades de la aplicación y dar una retroalimentación de su experiencia.
Implicación	La implicación es directa, ya que son los primeros en probar nuevas funcionalidades

TABLA 5 - STAKEHOLDERS. ROL: USUARIO BETA O TESTER

Nombre Tutor	
Descripción	Persona que orienta sobre el proceso
Responsabilidades	Supervisión, asesoramiento y evaluación
Implicación	Se implica a través de reuniones de seguimiento periódicas en las que aporta su opinión sobre la aplicación y las características técnicas del proyecto

TABLA 6 - STAKEHOLDERS. ROL: TUTOR

Nombre Universidad	
Descripción	Institución académica de enseñanza
Responsabilidades	Evaluación
Implicación	Implicación a través de la proporción de información como los plazos de entrega, documentación a aportar, evaluación final

TABLA 7 - STAKEHOLDERS. ROL: UNIVERSIDAD

Nombre Ingeniero informático	
Descripción	Se encarga de describir los requisitos, así como la implementación
Responsabilidades	La mayor parte ya que se encarga de que la aplicación funcione
Implicación	Implicado en la creación de la aplicación

TABLA 8 - STAKEHOLDERS. ROL: INGENIERO INFORMÁTICO

PERSONAS Y ESCENARIOS

En este apartado se definen los diferentes perfiles de usuarios que se han identificado como potenciales usuarios de la aplicación. Estas personas son ficticias y sus caras han sido generadas mediante la herramienta web ThisPersonDoesNotExists.

También se han representado algunos de los posibles escenarios para comprender como actuarían con la aplicación dadas unas situaciones específicas. De esta forma se puede comprobar si cubre las necesidades de los usuarios.

			
Nombre	Julia	Ana	Fernando
Edad	35	22	42
Profesión /Estudio	Dependienta	Estudiante de Grado en Ingeniería Industrial	Administrativo
Nacionalidad	Española	Española	Española
Uso de la App	Gestionar su economía	Establecer presupuestos	Comparar los gastos mensuales
Conocimientos tecnológicos	Utiliza el móvil para chatear con sus amigos, entrar en las redes sociales y ver vídeos en YouTube	Familiarizada con programas de diseño y modelado 3D. Manejo ágil del móvil	A diario utiliza el ordenador para programas de ofimática. El móvil lo utiliza para entrar en redes sociales y ver el tiempo

Escenario 1 – Julia añade un gasto

Escenario 2 – Ana limita sus gastos

Escenario 3 – Fernando compara las gráficas

MODELO CONCEPTUAL

Dado que se ha trabajado con una base de datos no relacional, se ha implementado un modelo conceptual con la herramienta DbSchema, que permite sincronizar la base de datos y realizar un modelo conceptual de las relaciones entre las tablas. Es por ello por lo que no se siguen las recomendaciones de Ingeniería del Software para generar un diagrama de clases.

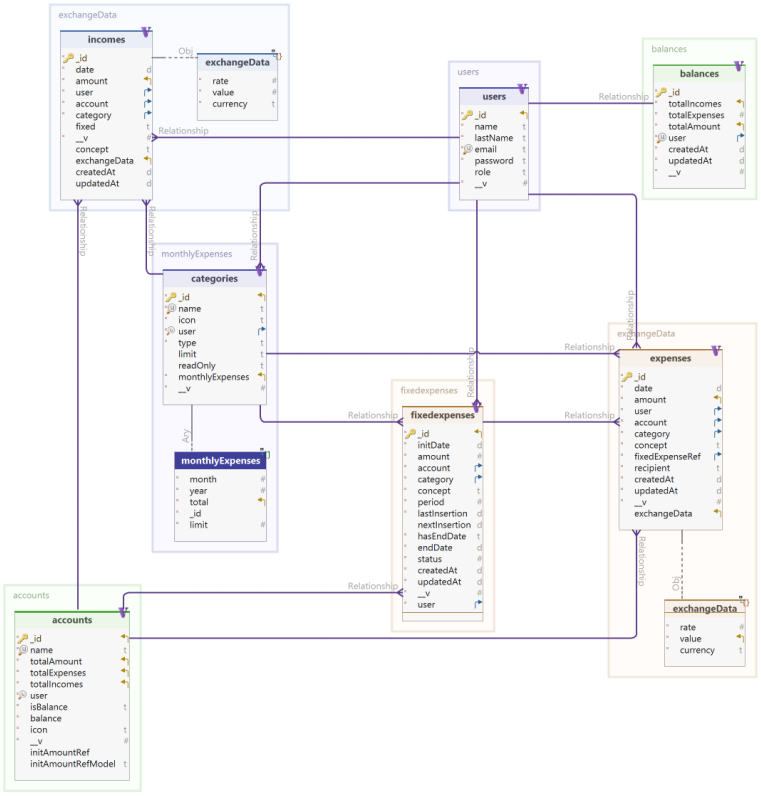


IMAGEN 1 - MODELO CONCEPTUAL

DIAGRAMA DE CASOS DE USO

IMAGEN 2 - DIAGRAMA DE CASOS DE USO

HISTORIAS DE USUARIO

En este apartado se describen las historias de usuario (HU) que se han identificado para llevar a cabo las funcionalidades necesarias para el completo desarrollo del software.

HU-01. Registrar usuario
Como usuario quiero registrarme para gestionar los gastos
Criterios de aceptación: - El formulario debe incluir los campos nombre, apellido, correo y contraseña. - Comprobar que no se permite registrar un usuario sin los campos obligatorios (todos los campos mencionados anteriormente). - Cuando se registra un usuario correctamente, se mostrará un mensaje de éxito y el usuario será redirigido a la pantalla para iniciar sesión. - Si no se rellenan los campos obligatorios o los campos contienen errores, se deberá mostrar un mensaje indicándolo. - El correo debe ser válido y único. - Si ya existe un usuario con el correo introducido se mostrará un mensaje.

- La contraseña debe contener al menos 6 caracteres.
- Al crear correctamente un usuario, se añadirán automáticamente unas categorías por defecto

TABLA 9 - HU-01. REGISTRAR USUARIO

HU-02. Crear cuenta
Como usuario quiero crear una cuenta para llevar un seguimiento del saldo de esta
Criterios de aceptación: - El formulario debe incluir los campos nombre, saldo inicial e incluir en el saldo. - Al crear una cuenta se deben añadir automáticamente los campos gastos totales, ingresos totales y saldo. - El campo “Incluir en el saldo” debe contener un botón de ayuda, que abra un modal con la información correspondiente de cómo funciona. - Si se establece un saldo inicial positivo, se creará de forma automática un ingreso con la categoría Saldo Inicial. - Si se establece un saldo inicial negativo, se creará de forma automática un gasto con la categoría Saldo Inicial. - Si el campo saldo inicial permanece en 0, no se crea ninguna transacción. - Si la cuenta se crea correctamente, se establecerán los valores de ingresos totales, gastos totales y saldo, según corresponda. - Cuando se crea una cuenta correctamente, el usuario será redirigido a la pantalla de cuentas. - Si no se rellenan los campos obligatorios o los campos contienen errores, se deberá mostrar un mensaje indicándolo. - El nombre debe ser un campo único. En caso de que se intente crear una cuenta con un nombre existente se deberá informar al usuario. - Si ocurre algún error al crear la cuenta se mostrará un mensaje de error.

TABLA 10 - HU-02. CREAR CUENTA

HU-03. Añadir gasto o ingreso
Como usuario quiero añadir gasto o ingreso (transacción) para llevar un seguimiento de mis finanzas
Criterios de aceptación: - El formulario además debe incluir los campos concepto, categoría, cantidad, tipo de divisa, precio del cambio, fecha y cuenta. - Comprobar que no se permite realizar una transacción sin los campos obligatorios (concepto, categoría, cantidad, tipo de divisa, tasa de cambio, fecha y cuenta). - Se debe permitir cancelar el proceso. - Cuando se guarda una transacción correctamente, se deberá actualizar el total de gastos o ingresos según corresponda y el saldo disponible de la cuenta con la que se ha realizado la transacción. - Una vez guardado la transacción, el usuario será redirigido a la pantalla general de dicha transacción (gastos o ingresos según corresponda). - Si no se rellenan los campos obligatorios o los campos contienen errores, se

deberá mostrar un mensaje indicándolo.
- Por defecto la divisa seleccionada debe ser EUR.
- Las categorías deben ser de tipo gasto.
- El formato de fecha debe ser dd/mm/yyyy.
- Si al añadir una transacción ocurre algún error se deberá mostrar un mensaje de error.
- Comprobar que, si ocurre un error añadiendo un gasto o ingreso, se deberán se revertir los cambios en la base de datos, permaneciendo la cantidad de gastos o ingresos y saldo de la cuenta como estaban.

TABLA 11 - HU-03. AÑADIR GASTO O INGRESO

HU-04. Crear gasto fijo
Como usuario quiero crear gastos fijos para facilitar la inserción de gastos periódicos.

Criterios de aceptación:
- El formulario debe incluir los mismos campos que crear gasto, además de un campo que permita seleccionar si tiene fecha final, fecha de fin y periodo (siendo este último obligatorio).
- Se deben incluir los campos: estado, próxima y última inserción
- Los periodos deben ser: diario, semanal, mensual y anual.
- Si el campo “Tiene fecha final” no está seleccionado, no se podrá especificar una fecha final.
- La fecha inicial tiene que ser igual o posterior al día de la creación del gasto fijo.
- Si se ha establecido una fecha final, esta deberá ser posterior a la fecha de inicio más: 1 día en caso de periodo diario, una 1 semana en caso de periodo semanal, un mes si el periodo es mensual y un año en caso de anual.
- Si deberán realizar automáticamente inserciones de gastos según la fecha de próxima inserción calculada.

TABLA 12 - HU-04. CREAR GASTO FIJO

HU-05. Crear categoría
Como usuario quiero crear una categoría para clasificar mis gastos

Criterios de aceptación:
- El formulario debe incluir los campos nombre, tipo e icono.
- Los tipos posibles son categorías de gastos o de ingresos.
- Si la categoría es de tipo gastos debe incluir un campo adicional, límite.
- El límite no puede ser un valor inferior a 0. Si el límite es 0, es lo mismo que si no tuviera límite.
- El nombre de la categoría es un campo único.
- Comprobar que no se permite crear una categoría sin los campos obligatorios (todos los campos mencionados anteriormente).
- Cuando se crea una categoría correctamente, el usuario será redirigido a la pantalla de categorías.
- Si no se rellenan los campos obligatorios o los campos contienen errores, se deberá mostrar un mensaje indicándolo.

- Si ya existe una categoría con el mismo nombre se mostrará un mensaje.
- Si ocurre algún error al crear la categoría se mostrará un mensaje de error.
- El campo límite debe contener un botón de ayuda, que abra un modal con la información correspondiente de cómo funciona.
- Después de añadir correctamente la categoría, se debe actualizar automáticamente la lista de categorías.

TABLA 13 - HU-05. CREAR CATEGORÍA

HU-06. Ver y eliminar gasto o ingreso
Como usuario quiero ver y eliminar gastos o ingresos para mantener actualizado el registro

Criterios de aceptación:

- Se deben mostrar todos los datos relacionados al gasto o ingreso.
- El usuario debe poder regresar a la pantalla anterior.
- Se deberá mostrar un mensaje de confirmación para asegurar que se desea eliminar el gasto o ingreso.
- Si se confirma la eliminación, el usuario será redirigido a la pantalla de gastos o ingresos.
- Si el usuario cancela la eliminación, será redirigido a los detalles de dicho gasto.
- Si se borra el gasto o ingreso correctamente, se deberá actualizar automáticamente la cantidad total de gastos o ingresos y el saldo disponible.

TABLA 14 - HU-06. VER Y ELIMINAR GASTO O INGRESO

HU-07. Editar y eliminar categoría
Como usuario quiero editar y eliminar categorías para mantenerlas actualizadas

Criterios de aceptación:

- La pantalla de edición de categoría debe mostrar los datos correspondientes a la categoría seleccionada.
- Se deben cumplir las mismas validaciones que al crear la categoría.
- Si la categoría se ha editado correctamente, el usuario debe ser redirigido a la lista de todas las categorías.
- Se debe poder cancelar el proceso.
- Se debe mostrar una confirmación al usuario antes de borrar la categoría.
- Si existen gastos o ingresos asociados con dicha categoría, se mostrará un mensaje indicando que si se borra la categoría se borrarán los datos asociados. El usuario podrá elegir entre confirmar la eliminación o cancelar el proceso.

TABLA 15 - HU-07. EDITAR Y ELIMINAR CATEGORÍA

HU-08. Ver y editar cuentas
Como usuario quiero ver y editar cuentas para mantenerlas actualizadas

Criterios de aceptación:

- La pantalla de edición de cuenta debe mostrar los datos correspondientes a

la cuenta seleccionada, incluidos el saldo total, el total de ingresos y el total de gastos.
- El saldo inicial no se debe poder editar.
- Si se actualiza el campo de “Incluir con el saldo”, el balance total de la aplicación se deberá actualizar según corresponda.
- En caso de que el usuario quiera borrar una cuenta y existan gastos o ingresos asociados, se mostrará un mensaje indicándolo y permitiendo cancelar el proceso, en caso, de confirmar que se quiere eliminar, se borrarán todos los datos asociados.

TABLA 16 - HU-08. VER Y EDITAR CUENTAS

HU-09. Ver y eliminar cuentas
Como usuario quiero ver el listado de cuentas y eliminarlas para gestionarlas.

Criterios de aceptación:
- En caso de que el usuario quiera borrar una cuenta y existan gastos o ingresos asociados, se mostrará un mensaje indicándolo y permitiendo cancelar el proceso, en caso, de confirmar que se quiere eliminar, se borrarán todos los datos asociados.
- Comprobar que se elimina la cuenta seleccionada y no otra.
- El usuario debe poder ver una lista de todas las cuentas que tiene registradas.

TABLA 17 - HU-09. VER Y ELIMINAR CUENTAS

HU-10. Editar mis datos
Como usuario quiero editar mis datos para mantenerlos actualizados.

Criterios de aceptación:
- El usuario debe poder editar su nombre, apellido y correo electrónico.
- Se podrá cancelar el proceso y se quedarán los datos anteriores.
- Si se utiliza un correo que ya está en uso se deberá informar.
- Si se guardan correctamente los datos actualizados, el usuario será redirigido a la pantalla anterior.

TABLA 18 - HU-10. EDITAR MIS DATOS

HU-11. Eliminar usuario
Como usuario quiero eliminar mi usuario para dejar de estar en el sistema.

Criterios de aceptación:
- Se debe mostrar un mensaje de confirmación para asegurar que quiere borrar la cuenta.
- Se mostrará un mensaje indicando que todos sus datos serán borrados del sistema y no se podrán recuperar.

TABLA 19 - HU-11. ELIMINAR USUARIO

HU-12. Editar gasto fijo
Como usuario quiero editar un gasto fijo para mantener actualizados los gastos
Criterios de aceptación: - La pantalla de edición debe mostrar los datos correspondientes al gasto fijo seleccionado. - Si el gasto fijo está activo se debe poder finalizar y si está finalizado, se debe poder activar. - Las validaciones de los campos son los mismos que para crear un gasto fijo. - Se debe poder cancelar el proceso. - Si se edita correctamente un gasto fijo, el usuario será redirigido a la pantalla de gastos fijos.

TABLA 20 - HU-12. EDITAR GASTO FIJO

HU-13. Iniciar y cerrar sesión
Como usuario quiero iniciar y cerrar sesión para acceder a mis datos y evitar que otra persona acceda cuando no la estoy usando.
Criterios de aceptación: - Debe contener los campos de correo y contraseña. - Se mostrará un mensaje de error indicando qué debe corregirse en caso de que no se rellenen los campos obligatorios o contengan errores. - El usuario tiene que estar previamente registrado en el sistema, en caso contrario, se mostrará un mensaje de error indicando que ese usuario no existe. - Tras realizar el inicio de sesión correctamente, el usuario será redirigido a la pantalla principal de la aplicación. - Si la contraseña o el correo son inválidos, se mostrará un mensaje de error indicándolo. - Tras cerrar sesión, el usuario no podrá acceder a ninguna de las funcionalidades de la aplicación hasta que vuelva a iniciar sesión.

TABLA 21 - HU-13. INICIAR Y CERRAR SESIÓN

HU-14. Ver gastos fijos y filtrarlos
Como usuario quiero ver mis gastos fijos y filtrarlos para acceder a sus datos.
Criterios de aceptación: - Se podrá seleccionar un gasto fijo para editarlo. - Los gastos fijos aparecerán divididos según si están activos o finalizados. - Se podrán buscar los gastos fijos por su concepto.

TABLA 22 - HU-14. VER GASTOS FIJOS Y FILTRARLOS

HU-15. Ver gráficos
Como usuario quiero ver gráficos para ver el progreso de mi economía
Criterios de aceptación: - Se podrá visualizar gráficos tanto de ingresos como de gastos, así como una combinación de los dos. - Se podrán visualizar gráficos según el método de pago, de ingresos o gastos anuales, mensuales, comparación de gráficas por mes del año seleccionado.

TABLA 23 - HU-15. VER GRÁFICOS

HU-16. Ver historial de gastos o ingresos
Como usuario quiero ver historial de gastos o ingresos para tener un control de ellos
Criterios de aceptación: - Se deberá poder elegir el mes del año del que se desea obtener el historial. - Se debe poder hacer clic en un ingreso o gasto para ver sus detalles - Los gastos o ingresos podrán ser filtrados según su categoría. - Los gastos o ingresos podrán ser ordenados según su fecha de creación, la cuenta con la que se ha realizado, por concepto y cantidad.

TABLA 24 - HU-16. VER HISTORIAL DE GASTOS O INGRESOS

HU-17. Exportar datos
Como usuario quiero exportar datos para poder utilizarlos fuera de la aplicación
Criterios de aceptación: - Se podrá exportar un informe en formato PDF con los datos del mes seleccionado. - Se podrá exportar un documento Excel con los datos del mes seleccionado. - El nombre del documento debe incluir el mes y el año.

TABLA 25 - HU-17. EXPORTAR DATOS

HU-18. Transferir dinero entre cuentas
Como usuario quiero transferir dinero entre cuentas para agilizar el registro de transacciones
Criterios de aceptación: - El formulario deberá tener los campos: concepto, cantidad, cuenta de origen y cuenta destino. - La cuenta de origen y destino no podrán ser las mismas. - Todos los campos mencionados son obligatorios. - Comprobar que, una vez realizada una transferencia, existe un ingreso en la cuenta destino y un gasto en la cuenta origen.

TABLA 26 - HU-18. TRANSFERIR DINERO ENTRE CUENTAS

REQUISITOS NO FUNCIONALES

Los requisitos no funcionales se identificarán con el código RNF-xx siendo las x el número que los diferencie.

Identificador	RNF-01
Descripción	El sistema debe ser compatible con dispositivos Android.
Tipo	Funcionalidad

TABLA 27 - RNF-01

Identificador	RNF-02
Descripción	El sistema debe ser intuitivo y fácil de usar.
Tipo	Usabilidad

TABLA 28 - RNF-02

Identificador	RNF-03
Descripción	El sistema debe poder ejecutarse en dispositivos móviles IOS realizando cambios mínimos en el código.
Tipo	Funcionalidad

TABLA 29 - RNF-03

Identificador	RNF-04
Descripción	El sistema debe tener un manual de usuario
Tipo	Usabilidad

TABLA 30 - RNF-04

DESARROLLO

ESTIMACIÓN Y PRIORIZACIÓN DE LAS HISTORIAS DE USUARIO

Para estimar la duración de las historias de usuario se determinó que un punto de historia de usuario equivaldría a 3 horas de un día de trabajo. Asimismo, se utilizó la regla MoSCoW para establecer la prioridad de estas. En la tabla que se muestra a continuación se muestran las historias de usuario con su estimación y prioridad correspondiente ordenadas de mayor a menor prioridad.

En dicha estimación se ha tenido en cuenta el tiempo que implicaría el desarrollo de la API y de la parte del *frontend*.

Historia de usuario	Estimación	Prioridad
HU-01. Registrar usuario	3	M
HU-02. Crear cuenta	2	M
HU-03. Añadir gasto o ingreso	3	M
HU-05. Crear categoría	2	M
HU-06. Ver y eliminar gasto o ingreso	4	M
HU-07. Editar y eliminar categoría	1	M
HU-11. Eliminar usuario	2	M
HU-13. Iniciar y cerrar sesión	3	M
HU-16. Ver historial de gastos e ingresos	5	M
HU-08. Ver y editar cuentas	1	S
HU-09. Ver listado de cuentas y eliminar cuenta	1	S

HU-10. Editar datos de usuario	1	S
HU-15. Ver gráficos	6	S
HU-04. Crear gasto fijo	4	C
HU-12. Editar gasto fijo	1	C
HU-14. Ver gastos fijos y filtrarlos	2	C
HU-17. Exportar datos	2	C
HU-18. Transferir dinero entre cuentas	2	C

TABLA 31 - ESTIMACIÓN Y PRIORIZACIÓN DE HISTORIAS DE USUARIO

Tras ello, podemos calcular que el total de puntos de historias de usuario es de 45, por tanto, teniendo en cuenta que la duración de un *sprint* es de dos semanas, la capacidad de trabajo es igual a:

Capacidad de trabajo = 3 días/semana x 3h/día x 2 semanas **= 18 horas**

TRABAJO REALIZADO

Además de las historias de usuario se tuvieron en cuenta otras tareas iniciales, que corresponderían a una fase inicial del trabajo.

- **Tarea 1**: Creación del proyecto del *backend*.
  - **1.1.** Creación y conexión con la base de datos.
  - **1.2.** Estructuración del proyecto.
  - **1.3.** Creación y arranque del servidor.
  - **1.4.** Configuración de rutas (Router)
- **Tarea 2:** Creación del proyecto de *frontend*.
  - **2.1.** Conexión con el backend (Axios)
  - **2.2.** Estructuración del proyecto
  - **2.3.** Creación del Store (Redux)
  - **2.4.** Creación del Router (React Router)

Como requisitos previos para la creación de ambos proyectos se necesitó instalar lo siguiente:

- Node js
- Npm (gestor de paquetes)
- Git (control de versiones)

Una vez instalados, se procedió a la inicialización del proyecto del *backend,* para ello se ejecutó el comando **npm init** y posteriormente se instalaron las siguientes librerías:

- **Express**.
- **Express-validator**: *middleware*[2] que facilita la validación de los datos de entrada de la petición.
- **Mongoose**: permite la conexión entre MongoDB y Node JS.
- **Dotenv**: se utiliza para acceder a las variables declaradas en el archivo .env
- **Jsonwebtoken**: creación de JWT para la autenticación de los usuarios.
- **Bcrypt**: encriptación de las contraseñas.
- **Nodemon**: detecta cambios en el código y reinicia el servidor automáticamente.

El proyecto se ha estructurado por componentes siguiendo la documentación de buenas prácticas en Node JS. (Goldbergyoni, s.f.). La estructura sería tal que así:

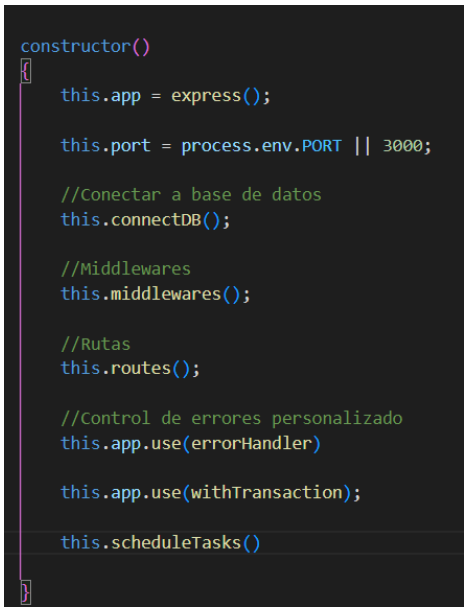
- **src/:** en esta carpeta se encuentra la lógica principal de la aplicación, componentes, rutas, servicios. Dentro de ella, las carpetas con los nombres de los componentes y a su vez estas contienen las siguientes carpetas:
    - **Controllers**: tal y como indica su nombre en inglés, aquí están ubicados los controladores de cada componente.
    - **Model**: en esta carpeta está el modelo que contiene el esquema.
    - **Routes**: contiene las funciones de las rutas a las que se le pueden realizar peticiones.

[2] Códigos que se ejecutan antes de que una petición HTTP llegue al manejador de rutas o antes de que un cliente reciba una respuesta.**Fuente especificada no válida.**

    - **Middlewares**: en ella se encuentran las funciones que tienen acceso al objeto de solicitud (req) y al de respuesta (res), y a la siguiente función de middleware en el ciclo de solicitud/respuesta de la aplicación.
    - **Services**: aquí se incluye todos los métodos que acceden a la base de datos y son llamados desde el controlador, dividiendo así las responsabilidades para mantener un código más limpio.
- Además, dentro de la carpeta src se encuentra una carpeta llamada ***Core*** que contiene código general entre ellos: las configuraciones, excepciones, middlewares generales, manejo de errores...

En el archivo .env se almacenan las variables de entorno necesarias, tales como el puerto en el que se ejecutará el servidor, la dirección para conectarse a la base de datos, y claves.

Se implementó la clase Server en el archivo server.js, desde la que se generará la instancia del servidor en el archivo index.js. En ella se realiza una llamada a la conexión de la base de datos, la ejecución de los middlewares, la generación de las rutas de la API, la gestión del control de errores personalizados, el uso de transacciones para la base de datos y la ejecución de tareas programadas.



```
constructor()
{
    this.app = express();

    this.port = process.env.PORT || 3000;

    //Conectar a base de datos
    this.connectDB();

    //Middlewares
    this.middlewares();

    //Rutas
    this.routes();

    //Control de errores personalizado
    this.app.use(errorHandler)

    this.app.use(withTransaction);

    this.scheduleTasks()

}
```

IMAGEN 3 - CONSTRUCTOR DEL SERVIDOR

Para la inicialización del servidor se implementó el método ***listen()***, que utiliza la variable de clase *app* que es una instancia de la aplicación Express inicializada en el constructor.

```
listen()
{
    this.app.listen(this.port, () =>
    {
        console.log(`Servidor corriendo en el puerto ${this.port}`)
    });
}
```

IMAGEN 4 - MÉTODO LISTEN DEL SERVIDOR

El siguiente paso fue crear el archivo index.js el cual se ejecuta cuando inicializamos la aplicación. Este archivo contiene la creación de una instancia del servidor y su inicialización a través de la llamada al método ***listen()***, tal y como se muestra en la siguiente imagen:

```
const Server = require('./server');
const server = new Server();

server.listen();
```

IMAGEN 5 - INICIALIZACIÓN DEL SERVIDOR

La conexión a la base de datos se realizó usando mongoose. Se han establecido a true las opciones: *useNewUrlParser* y *useUnifiedTopology* siguiendo las recomendaciones de la documentación oficial para evitar que funciones obsoletas afecten sobre el correcto funcionamiento del servidor.

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

const dbConnection = async () =>
{
    try
    {
        await mongoose
            .connect(process.env.MONGODB_ATLAS, {
                useNewUrlParser: true,
                useUnifiedTopology: true
            })
            .then(() => console.log('Conexión de la base de datos correcta'));

    } catch (e)
    {
        console.log(e);
        throw new Error('Error en la incialización de la base de datos');
    }

}

module.exports = {
    dbConnection
}
```

IMAGEN 6 - MÉTODO PARA LA CONEXIÓN A LA BASE DE DATOS

Por último, se creó un controlador de errores personalizado para tener un código más limpio a la hora de lanzar errores. Como se muestra en la imagen se comprueba que tipo de error se ha producido y se devuelve el código y el mensaje correspondiente.

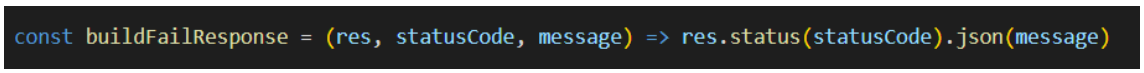
```
const errorHandler = (err, req, res, next) =>
{

    if (err instanceof InvalidEmailOrPasswordException) return build401Response(err, res);
    if (err instanceof InvalidTokenException) return build401Response(err, res)
    if (err instanceof ForbiddenException) return build403Response(err, res);
    if (err instanceof NotFoundException) return build404Response(err, res);
    if (err instanceof MongoError)
    {
        const specificError = mongoErrorHandler(err)
        if (specificError instanceof AlreadyExistsEntityException) return build409Response(err, res);
        return errorHandler(specificError, req, res, next)
    }

    return buildInternalServerErrorResponse(err, res)
}
```

IMAGEN 7 - CONTROLADOR DE ERRORES PERSONALIZADOS

Para evitar duplicar código cada método que construye un tipo de error llama a la función *buildFailResponse* a la que se le pasa por parámetro la respuesta, el código y el mensaje.



```
const buildFailResponse = (res, statusCode, message) => res.status(statusCode).json(message)
```

IMAGEN 8 - MÉTODO PARA CONSTRUIR LA RESPUESTA DE ERROR

Aquí se muestra un ejemplo para el código de error más común 404, en el que se llama el método *buildFailResponse:*

```
const build404Response = (err, res) =>
{
    buildFailResponse(res, 404, {
        status: 404,
        message: err.message || 'Not found'
    })
}
```

IMAGEN 9 - MÉTODO PARA EL ERROR 404

Para la creación del proyecto de *frontend* se utilizó la documentación de React-Native. Además, se instaló un emulador de Android con Android Studio para poder ejecutar la aplicación.

El archivo index.js registra el componente principal que se renderizará, en este caso App, con el nombre que se almacena en el archivo app.json, "FrontGestionGastos"

```
import {AppRegistry} from 'react-native';
import App from './App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

IMAGEN 10 - ARCHIVO INICIAL DE LA APP

La estructuración de carpetas ha sido la siguiente:

- **src/**
    - **assets:** aquí se encuentran los recursos.
    - **components:** componentes reutilizables, por ejemplo, modales, botones personalizados, inputs.
    - **modules:** en esta carpeta, separado por componentes donde se encuentran las acciones que envía los datos al store y los *reducers* que determinan que acciones se realizan y modifica el estado.
    - **navigation:** configuración de la navegación a través de las pantallas de la aplicación, así como la barra de menú.
    - **screens:** todas las pantallas de la aplicación.
    - **services:** llamadas API al *backend*.

- **store:** configuración del Store de Redux.
- **Utils:** utilidades tales como validadores de formulario.

El Store contiene el estado de la aplicación, la única forma de cambiar dichos estados es despachando una acción. La creación de este se realizó siguiendo los pasos de la documentación oficial de Redux (ConfigureStore | Redux Toolkit, s.f.)

```
import { configureStore } from '@reduxjs/toolkit'
import thunk from 'redux-thunk'
import { persistReducer, persistStore } from 'redux-persist';

import AsyncStorage from '@react-native-async-storage/async-storage';
import rootReducer from './rootReducer';


// Configura la persistencia
const persistConfig = {
    key: 'root',
    storage: AsyncStorage,
};

// Combina los reducers en uno solo
const persistedReducer = persistReducer(persistConfig, rootReducer);

// Configura el store
const store = configureStore({
    reducer: persistedReducer,
    middleware: [thunk]
});

// Configura la persistencia
const persistor = persistStore(store);


export { store, persistor };
```

IMAGEN 11 - CONFIGURACIÓN DE STORE DE REDUX

En rootReducer, se almacena la combinación de todos los *reducers* [3] de la aplicación y la implementación de una función que restablece el estado general.

Para finalizar la configuración de Redux, se engloba la aplicación con el Provider al que se le manda el store, permitiendo así que el estado se accesible desde cualquier parte de la aplicación. Asimismo, también se utiliza la PersistGate que garantiza tener siempre la última actualización de los datos del estado.

[3] Funciones que obtienen el estado actual y devuelven uno nuevo.

IMAGEN 12 - CONFIGURACIÓN DE LA APP CON REDUX

Para la navegación entre pantallas, se he establecido en el archivo Routing.js una clase con el mismo nombre de este, en el cual se almacenan los nombres de las pantallas. En el Router.js, se ha implementado el stack de las pantallas para poder navegar entre ellas y la configuración de la barra de menú. Además, aquí se tiene en cuenta si el usuario está autenticado o no, restringiendo el acceso del usuario a las pantallas que no debería.

Por último, se creó RootRouting con las funciones de: navegar, ir hacia atrás, reemplazar y eliminar ruta de la pila, de esta forma el código se hace más claro y limpio cuando sea necesario utilizarlas.

```
import { createNavigationContainerRef, StackActions, CommonActions } from '@react-navigation/native';

export const navigationRef = createNavigationContainerRef()

export function navigate(name, params)
{
    if (navigationRef.isReady())
    {
        navigationRef.navigate(name, params);
    }
}

export function goBack()
{
    if (navigationRef.isReady())
    {
        navigationRef.goBack();
    }
}

export function replace(name, params)
{
    if (navigationRef.isReady())
    {
        navigationRef.dispatch(
            StackActions.replace(name, params),
        );
    }
}

export function removeRouteFromStack(name)
{
    if (navigationRef.isReady())
    {
        navigationRef.dispatch(state =>
        {

            const routes = state.routes.filter(r => r.name !== name);

            return CommonActions.reset({
                ...state,
                routes,
                index: routes.length - 1,
            });
        });
    }
}
```

IMAGEN 13 - FUNCIONES DE NAVEGACIÓN

La conexión con el backend a través de axios se realizó de la siguiente forma, en la que Config.BASE_URL contiene en este caso la dirección en local del servidor:

```
let httpClient = axios.create();
let baseUrl = Config.BASE_URL;
let response = null;
httpClient.defaults.baseURL = baseUrl
```

IMAGEN 14 - AXIOS CONEXIÓN CON EL BACKEND

La conexión se realiza desde un método llamado *launchAsyncTask*, al que se la pasa por parámetro, la etiqueta de la acción que se va a realizar, el método que se

va a utilizar (GET, POST, PUT y DELETE), la URL, la configuración, los parámetros y las funciones de error y éxito. Con esta función se gestionan las llamas API que se van a realizar de forma que no hay que repetir el código múltiples veces. A continuación, se muestra un ejemplo de una petición tipo GET para obtener las categorías de un usuario autenticado.

Con este ejemplo también se explicará cómo funciona el proceso de la petición desde que es disparada con redux hasta que se manda el *backend,* ya que funciona de la misma manera para todas las peticiones.

Para cada componente existe un archivo llamado Nombre del módulo + Actions. En él se encuentran todos los métodos que realizarán llamadas a la API. En este caso cuando se llama al método apiGetCategories, en primer lugar, se establece que los datos se están cargando, a continuación, se despacha el método que se encuentra en el archivo API, en este caso, *getCategories.*, que contiene las funciones de error y éxito respectivamente, estas serán ejecutadas según el resultado de la solicitud.

```
export const apiGetCategories = () => async (dispatch, getState) =>
{
    dispatch(setCategoryDataState({ prop: 'isLoadingCategories', value: true }));
    await dispatch(
        getCategories((tag, response) =>
        {
            console.log('getCategories - ERROR: ', response);
            dispatch({ type: Types.GET_CATEGORIES_FAILED, payload: response });
        }, (tag, response) =>
        {
            console.log('getCategories - SUCCESS: ', response);
            dispatch({
                type: Types.GET_CATEGORIES_SUCCESS,
                payload: response.data.categories,
            });
        }))

    dispatch(setCategoryDataState({ prop: 'isLoadingCategories', value: false }))

};
```

IMAGEN 15 - ACCIÓN DE REDUX PARA OBTENER LAS CATEGORÍAS

El método *getCategories* configura la URL a la que se tiene que realizar la petición, la configuración con el token de autenticación del usuario, se indica que la petición es de tipo GET y se dispara la función mencionada anteriormente (launchAsynkTask).

```
export const getCategories = (callbackError, callbackSuccess) => async (dispatch, getState) =>
{
    let params = {};
    let url = `${BASE_URL}/api/categories`

    const { authToken } = getState().AuthReducer
    let config = {
        headers: { Authorization: 'Bearer ' + authToken },
    };

    return dispatch(launchAsyncTask(Tags.GET_CATEGORIES, GET, url, config, params, callbackError, callbackSuccess));
};
```

IMAGEN 16 - MÉTODO PARA OBTENER LAS CATEGORÍAS

En launchAsyncTask se comprueba que tipo de petición se ha realizado, y se ejecuta con los datos proporcionados por parámetros, como este ejemplo trata de una petición de tipo GET, mostraremos ese caso. No obstante, para el resto de las peticiones es igual, variando el método que se llama desde httpClient.

```
if (verb === GET)
{
    await httpClient
        .get(url, config)
        .then((result) =>
        {
            response = result;
        })
        .catch((error) =>
        {
            Alert.alert(
                `Error`,
                `Ha ocurrido un error. Reinicie la app e inténtelo más tarde`,
                [{
                    text: 'Aceptar'
                }]
            );
            response = error.response;
        });
}
```

IMAGEN 17 - MANEJO DE PETICIONES GET DESDE LAUNCHASYNTASK

Por último, este método llama a la función *onResponse,* que dependiendo del estado de la respuesta de la petición ejecuta la función de éxito o de error, mencionadas anteriormente.

```
export const onResponse = (tag, response, callbackError, callbackSuccess) => async (dispatch) =>
{
    console.log('TAG: ', tag, ' | Response: ', response);

    if (response === undefined || response === null) return;

    switch (response.status)
    {
        case 200:
            callbackSuccess(tag, response.data);
            break;

        case 400:
            callbackError(tag, response.data);
            break;
```

IMAGEN 18 - MÉTODO ONRESPONSE

Cuando estas funciones son ejecutadas, se retorna al método *apiGetCategories* del *Actions* (Imagen 13), y se ejecuta el método de éxito o fallo retornado anteriormente. Una vez ahí, se despacha el método del *reducer* según el *type* especificado y el *payload* enviado.

El *reducer* contiene una variable denominada INITIAL_STATE, que almacena los estados en este caso de las categorías, así como el reductor al que se le pasa por parámetro el estado actual y una acción. Se comprueba que acción ha sido despachada, y se devuelve el estado actualizado.

```
const INITIAL_STATE = {
    name: '',
    icon: '',
    type: '',
    categories: [],
    initCategories: [],
    category: [],
    errors: [],
    isLoadingCategories: false,
    isLoadingCategory: false,
};
```

IMAGEN 19 - ESTADO INICIAL DE LAS CATEGORÍAS

En este caso y suponiendo que todo ha salido bien, el tipo despachado sería GET_CATEGORIES_SUCCESS y, por tanto, se establecería en categorías, el *payload* obtenido previamente, es decir, las categorías obtenidas desde el back.

```
case Types.GET_CATEGORIES_SUCCESS:
    return { ...state, categories: action.payload };
```

IMAGEN 20 - EJEMPLO DE CASO DE ÉXITO OBTENIENDO CATEGORÍAS

REGISTRO

En le parte del *backend* para implementar el registro, se ha creado el modelo Usuario con la función Schema de mongoose, que contiene los campos nombre, apellidos, correo y contraseña. Todos estos campos son obligatorios y de tipo *string*. Además, el email es un campo único identificador del usuario.

Para la API, se ha creado la ruta de tipo POST, con la validación de los campos. El método *validateFields*, es un método middleware que se utilizará en todas las rutas, permite validar que los campos recibidos cumplen las validaciones que se han establecido. En él se utiliza validationResult de express-validator, en caso de que no ocurra ningún error, continúa la petición.

```
router.post(
    '/',
    [
        check('name').not().isEmpty().isString().escape(),
        check('lastName').not().isEmpty().isString().escape(),
        check('email', 'email is required').isEmail().escape(),
        check('password', 'password is required').not().isEmpty().escape(),
        validateFields,
    ],
    createUserController,
)
```

IMAGEN 21 - RUTA POST PARA EL REGISTRO DE USUARIOS

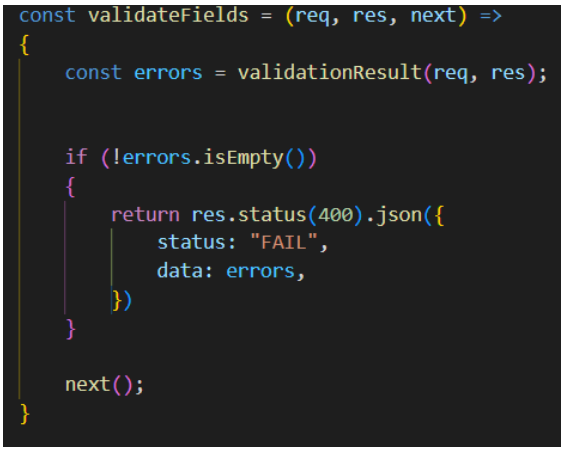


IMAGEN 22 - VALIDADOR DE LOS CAMPOS DE LAS PETICIONES

Si se han cumplido todas las validaciones, entonces se llama al controlador, que tiene los parámetros: *request, response, next, sesión*. En el controlador se llama al servicio para crear un usuario, *createUser*, y se le pasan los datos de la petición.

El servicio para crear un usuario también llama a un método que se ha implementado que crea un modelo *Balance,* en el cual se almacenará el total de gastos, ingresos y saldo remanente de todas las cuentas que así lo indiquen.

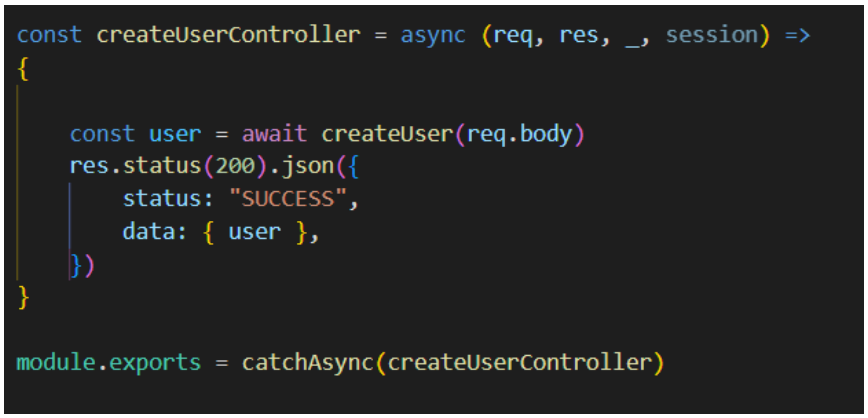


IMAGEN 23 - CONTROLADOR PARA LA CREACIÓN DE USUARIOS

Si todo ha salido correctamente, se devuelve una respuesta con un código 200 y los datos del usuario. Como se observa en la exportación del controlador se engloba en un método catchAsync, el cual se ha implementado para evitar código duplicado de los *try* y *catch*, y gestionar el control de excepciones, en caso de que ocurra un error, este se disparará.

```
const catchAsync = (fn) => async (req, res, next) =>
{
    const session = await mongoose.startSession();

    session.startTransaction();

    try
    {
        await fn(req, res, next, session);
        await session.commitTransaction();
        session.endSession();
    } catch (error)
    {
        await session.abortTransaction();
        session.endSession();
        next(error);
    }
};
```

IMAGEN 24 - MÉTODO CATCHASYNC PARA EL CONTROL DE ERRORES

El servicio para crear usuarios utiliza los métodos *genSaltSync* y *hashSync* para crear un hash[4] de la contraseña, que será la que se almacene en la base de datos.

En el servidor se estableció la ruta /api/users que será a donde se realicen las peticiones. En la parte del *frontend*, se creó la configuración de redux para este módulo, la petición API, y el formulario de registro.

Para el formulario de registro también se crearon validaciones que comprueban que no pueden mandarse campos vacíos y que la contraseña debe tener al menos 6 caracteres. Esto se consiguió mediante una clase llamada *FormValidatorsManager* y un método *formRegister*. En esta clase se establecerán todos los validadores de los formularios. Si ocurren errores, se devolverán los mensajes correspondientes.

[4] Función criptográfica para cifrar una contraseña

```
static formRegister = (props) =>
{
    const { name, lastName, email, password } = props;

    const error = [];

    if (!isValidEmail(email) && isValidString(email))
        error.push({ key: 'email', value: "Email no válido" });

    if (!isValidPassword(password))
        error.push({ key: 'password', value: "La contraseña debe tener al menos 6 caracteres" });

    if (!isValidString(name))
        error.push({ key: 'name', value: "Campo obligatorio" });

    if (!isValidString(email))
        error.push({ key: 'email', value: "Campo obligatorio" });

    if (!isValidString(lastName))
        error.push({ key: 'lastName', value: "Campo obligatorio" });

    if (error.length !== 0) return error;

    return null;

}
```

IMAGEN 25 - VALIDADOR DE FORMULARIO DE REGISTRO DE USUARIO

Antes de realizar la petición, se llama al validador con los datos introducidos por el usuario, si ocurre algún fallo, se muestran los errores en la interfaz y no se realiza la llamada.

Por otro lado, una vez realizada la petición, si ya existe un usuario con el correo introducido, se devuelve ese error y se muestra. Si ocurre otro error no contemplado en la aplicación aparece un modal con un mensaje general.

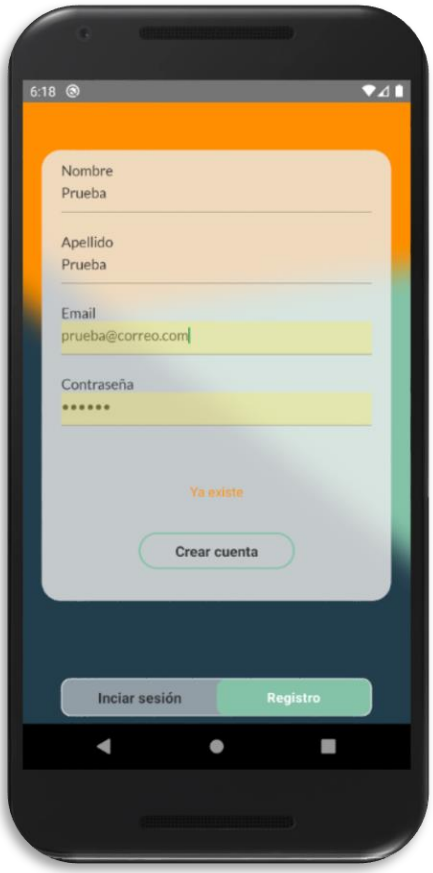


IMAGEN 26 - REGISTRO CON USUARIO EXISTENTE IMAGEN

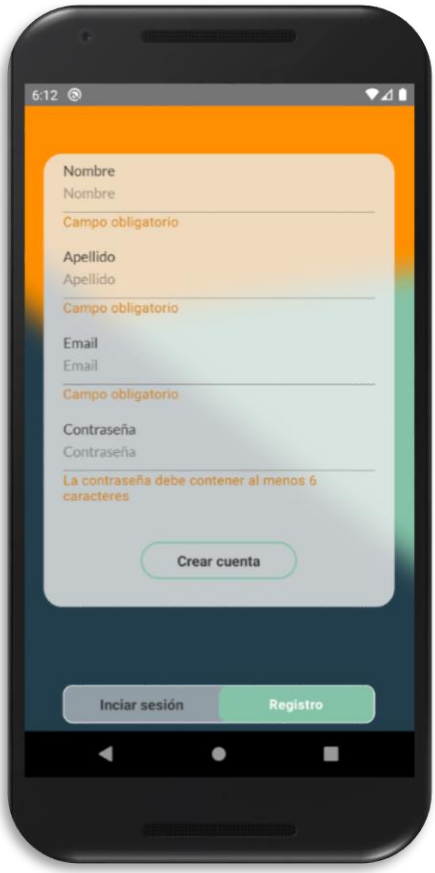


IMAGEN 27 - REGISTRO CON VALIDACIÓN DE CAMPOS

INICIAR Y CERRAR SESIÓN

Cuando se realiza una petición de inicio de sesión, se valida el formato del correo y que ningún campo esté vacío, se busca en la base de datos el usuario con el correo proporcionado, posteriormente con la ayuda de la librería bcrypt, se compara la contraseña proporcionada y la del usuario obtenido de la base de datos. En caso de que no se haya encontrado ningún usuario con el correo proporcionado o que la contraseña no coincida se lanzarán los errores, *NotFoundException* o *InvalidEmailOrPasswordException,* respectivamente. Por el contrario, si todo ha salido bien, se genera un token del tipo JWT. Este se utilizará para identificar al usuario en todas las peticiones futuras.

Además, cada vez que se realice una petición que requiera que el usuario esté autenticado, primero se validará dicho token con el middleware *validateJWT* y posteriormente que el usuario solo pueda acceder a sus datos. Esto último se

consigue con el middleware hasPermission de cada modelo, que verifica que el usuario tenga los permisos correspondientes sobre la acción o los datos a los que se van a acceder.

En la interfaz también se validan los campos antes de mandarlos, y se muestran los mensajes de error cuando corresponde. Asimismo, una vez realizada la petición, si el usuario y la contraseña no coinciden o el usuario no existe, se informará tal y como se muestra en las imágenes.

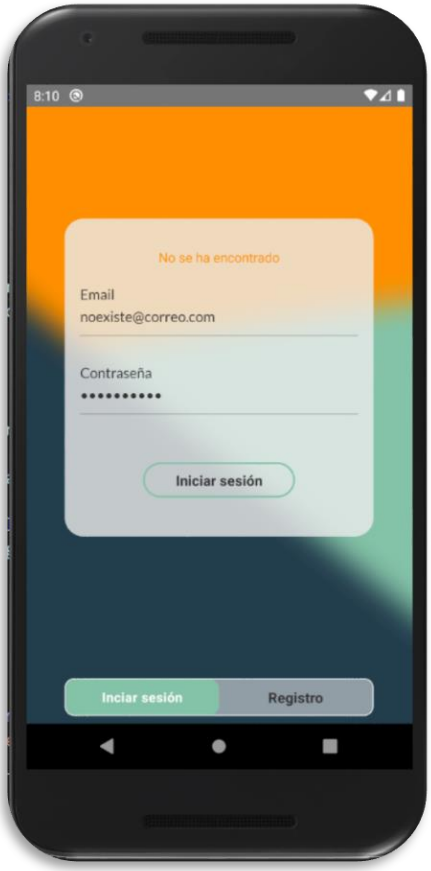


IMAGEN 28 - INICIO DE SESIÓN. USUARIO NO EXISTENTE

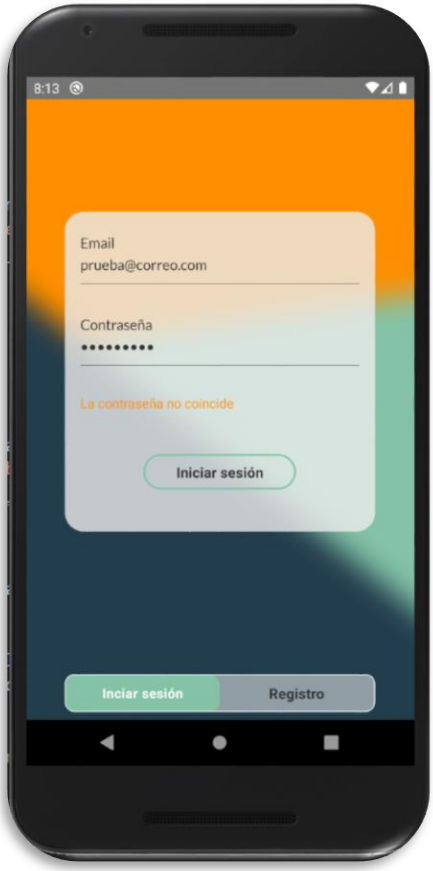


IMAGEN 29 - INICIO DE SESIÓN. CONTRASEÑA INVÁLIDA

Para cerrar sesión, se obtiene el token de autenticación y se decodifica para obtener su fecha de expiración, ambos se guardan en la base de datos en el modelo *Blacklist*. Esta servirá en un futuro para comprobar en las peticiones que no se están enviando tokens que ya han sido utilizados. Si todo ha salido correcto, en la parte del

frontend se eliminan los datos del usuario autenticado que estaban almacenados en redux y se redirecciona a la pantalla de inicio de sesión.

CREAR CUENTA

Cuando se crea una cuenta, si los campos que se muestran en la imagen siguiente son correctos, se comprueba el valor del saldo inicial, si es positivo (mayor que 0) se crea por defecto un ingreso, si es negativo un gasto, ambos con el concepto Saldo Inicial + Nombre de la cuenta, si se deja en 0 no se crea ningún registro. Si se crea un registro (gasto o ingreso), se almacena el id en la instancia de la cuenta y se actualizan los valores totales de esta (total de ingresos, total de gastos y saldo). Si además el usuario ha seleccionado la opción de “Incluir con el saldo”, se añade una referencia al saldo (Modelo Balance) del usuario y se actualiza (ingresos y gastos totales y saldo remanente).

Si la petición se ha resuelto satisfactoriamente, el usuario es redirigido a la pantalla con la lista de cuentas.

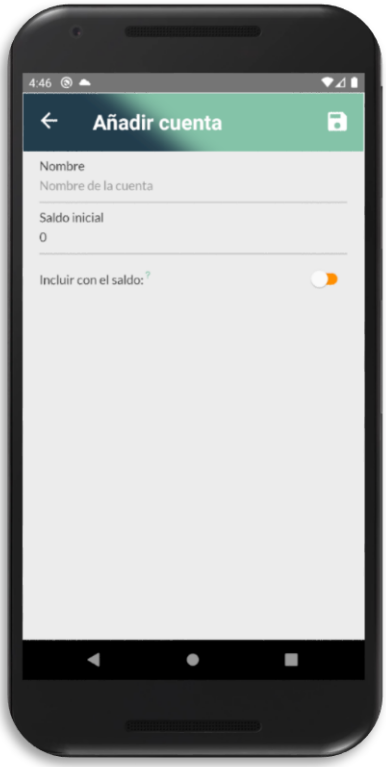


IMAGEN 30 - PANTALLA PARA CREAR CUENTAS

CREAR CATEGORÍA

Las categorías pueden ser de dos tipos: gastos o ingresos. Las categorías de gastos tienen un campo adicional, el límite, que no puede ser negativo. El límite controla si la cantidad en euros de los gastos introducidos superan al mismo, es decir, permite establecer un presupuesto para las categorías.

Para controlar el límite, se ha creado un campo en las categorías llamado *monthlyExpenses*, es un array que contiene los valores: mes, año, total y límite. Al principio se planteó el límite como un campo general de categoría, pero después se concluyó que no era lo correcto, ya que así no se tenía un control del histórico de su valor. Con *monthlyExpenses* se mantiene el registro del límite establecido para todos los meses y del total acumulado.

Por tanto, en el backend cada vez que se crea una categoría de tipo gasto se añade automáticamente el array montlyExpenses con los valores iniciales de: mes actual, año actual, total igual a 0, y límite que haya establecido el cliente. Si es de tipo ingreso, este campo se omite.

Además, se han añadido unas categorías por defecto cuando se crea un usuario para ofrecer un aterrizaje en la aplicación más llevadero, que podrán ser editadas o eliminadas según convenga.

En la parte de la interfaz se hicieron las validaciones pertinentes, y se añadió una cantidad considerable de iconos para que las categorías se puedan identificar más fácil visualmente. También se ha tenido en cuenta añadir un modal con la explicación del campo límite mejorando con ello la experiencia de usuario. A continuación, se puede observar el formulario para la creación de categorías y el modal para le selección de iconos de esta.

IMAGEN 31 - FORMULARIO PARA CREAR CATEGORÍAS

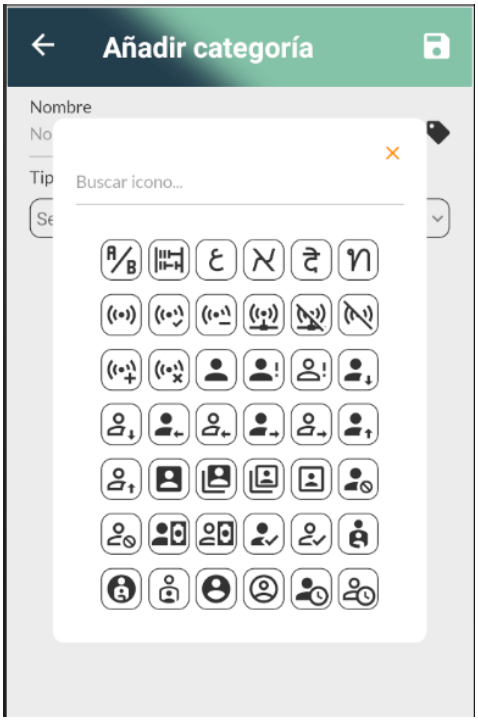


IMAGEN 32 - MODAL SELECCIÓN DE ICONO

AÑADIR GASTO O INGRESO

Esta historia de usuario se decidió que fuese en conjunto, tanto ingresos como gastos, ya que ambos tienen los mismos campos. No obstante, se concluyó que cada uno tuviera su propio modelo para que fueran flexibles y tuvieran escalabilidad en un futuro y facilitar las búsquedas en base de datos.

En el formulario que se muestra a continuación, se observan los campos necesarios para crear un gasto o ingreso, todos obligatorios. El usuario deberá haber creado previamente al menos una categoría y una cuenta si desea añadir un gasto o ingreso. Para seleccionar la categoría se deberá hacer clic en el icono que aparece al lado del campo “Concepto”, este icono abrirá un modal con todas las categorías disponibles (Aparecerán las de tipo gastos o ingresos según corresponda).

IMAGEN 33 - FORMULARIO PARA CREAR GASTOS E INGRESOS

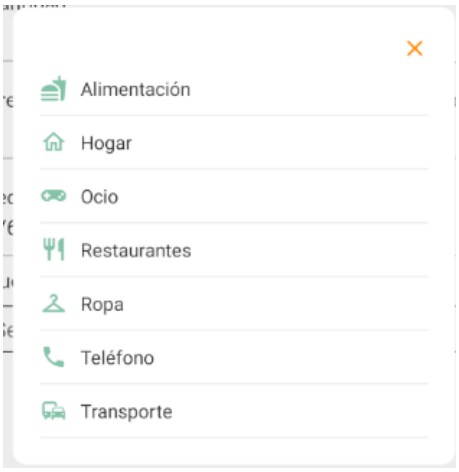


IMAGEN 34 - MODAL DE CATEGORÍAS PARA GASTOS

También existe la posibilidad de insertar una cantidad en otra divisa que no sea el euro. Esta será convertida con el precio que aparece en el campo "Precio 1 USD a EUR) a euros. En este caso, se muestra USD ya que es la divisa seleccionada, pero variará según cuál se haya elegido. El precio del cambio de divisa se obtiene a través de una llamada GET a la API Frakfurter (Frankfurter Exchange rates and currency data API, s.f.) que devuelve los valores actuales. Si estos no coinciden con los que el usuario utilizó cuando realizó el gasto o el ingreso, tendrá la posibilidad de editar dicho precio. La cantidad convertida se calculará según el precio establecido.

En lo que a base de datos se refiere en este tema, se almacenará la cantidad convertida en el campo *amount*, y el resto de los datos se incluirán en el array exchageData (divisa, precio del cambio y cantidad introducida).

IMAGEN 35 - EJEMPLO CONVERSIÓN DE DIVISA

Cuando se añade un gasto, en primer lugar, se obtiene la categoría seleccionada y se comprueba si contiene algún registro de gastos para la fecha indicada. Si existe, se actualiza el total de gastos, y se analiza si la fecha es posterior a

la actual, en tal caso, el límite de ese mes se actualiza con el valor del límite del mes actual en caso de que exista o con la del mes anterior al del gasto que contenga un límite. Esto se ha realizado así, para que cuando llegue ese mes, si el límite se ha ido modificando los meses anteriores, se tenga el valor del límite más reciente.

Si no se ha encontrado ningún registro de gastos para la categoría, se crea uno nuevo. Si la fecha del gasto es posterior a la actual, es decir, el mes es posterior al actual y el año es el mismo o posterior, entonces se busca el valor del límite más reciente para actualizarlo. Por el contrario, si se trata de un gasto antiguo, se establece a 0. En el caso de los ingresos esto no se tiene en cuenta ya que las categorías de tipo ingreso no tienen límite.

Una vez añadido el gasto o el ingreso, se actualizan los totales de la cuenta seleccionada, y en caso de que la cuenta tenga la opción de “Incluir en el saldo” también se actualizará el saldo total de la aplicación.

VER Y ELIMINAR GASTO O INGRESO

Dado que los campos de gastos e ingresos eran los mismos, se ha optado por visualizar losg e ingresos de la misma manera. A continuación, se muestra un ejemplo de la visualización de un gasto y del modal de confirmación para borrarlo.

IMAGEN 36 - VISUALIZACIÓN DE UN GASTO

IMAGEN 37 - MODAL DE CONFIRMACIÓN PARA ELIMINAR UN GASTO

Cuando un usuario elimina un gasto o ingreso, se realizan unas acciones adicionales en el backend para mantener la consistencia de los datos. Entre ellas, actualizar el total acumulado de la categoría que tenía asociada si se trata de un gasto, actualizar el total de ingresos o gastos de la cuenta según corresponda, y el saldo total de esta, y en caso de que la cuenta tenga habilitada la opción “Incluir con el saldo”, actualizar también los datos globales de la aplicación.

Hay que destacar que todas las acciones que conllevan actualización de múltiples documentos se realizan a través de las denominadas transacciones, garantizando la consistencia e integridad de la aplicación, ya que se realiza de forma atómica, es decir, que todas las operaciones mencionadas anteriormente se ejecutan como si fuera una sola, por lo que si alguna falla, ninguna de ellas es guardada. Por ejemplo, si cuando se elimina un gasto, ocurre un fallo actualizando la

cuenta, la eliminación se suspende, evitando así que se elimine el gasto y la cuenta quede desactualizada.

EDITAR Y ELIMINAR CATEGORÍA

Cuando se selecciona una categoría para editarla, se envía una solicitud con el id de esta, obteniendo toda su información y mostrándola. Al editar una categoría no se permite cambiar el tipo, evitando así problemas de integridad e inconsistencia. Para que quede más claro para el usuario, al selector de "tipo" se le ha bajado la opacidad un 50% dando la impresión de estar inactivo, además de haber deshabilitado su función.

IMAGEN 38 - FORMULARIO DE EDICIÓN DE CATEGORÍA

Al eliminar una categoría se han tenido en cuenta todos los gastos o ingresos que pueda llevar asociados, por lo que se ha creado un modal de confirmación que indica que los datos asociados también serán borrados.

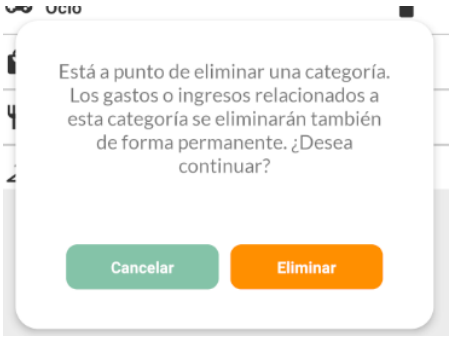


IMAGEN 39 - MODAL DE CONFIRMACIÓN PARA ELIMINAR CATEGORÍA

Teniendo en cuenta lo mencionado anteriormente, habría que considerar que los gastos o ingresos asociados a una categoría, podrían estar asociados a diferentes cuentas, por lo que se eliminan los gastos o ingresos, se obtienen todas las cuentas del usuario autenticado y se actualizan. La actualización de las cuentas también conlleva que se recalculen los gastos e ingresos totales de la aplicación, que está implícito en la actualización de cada cuenta.

```
const deleteCategory = async (categoryId, user, session) =>
{
    const category = await Category.findOneAndDelete({ _id: categoryId }).session(session)
    if (!category) throw new NotFoundException(`Category with the id ${categoryId} not found`)

    if (category.type == "Expenses")
        await Expense.deleteMany({ category: { $in: categoryId } }).session(session)
    else await Income.deleteMany({ category: { $in: categoryId } }).session(session)

    const accounts = await getAccountsByUser(user)

    await updateAccountsAmount(accounts, user, session)

    return category
}

const updateAccountsAmount = async (accounts, user, session) =>
{
    await Promise.all(accounts.map(ac => updateAccountAmounts(ac.id, user, session)));
};
```

IMAGEN 40 - MÉTODO PARA ELIMINAR UNA CATEGORÍA

Eliminar usuario

Eliminar un usuario conlleva eliminar todos sus datos, por lo que previamente se le muestra un modal de confirmación informando que se perderá toda la información y con la opción de que el usuario confirme o cancele la acción. Puesto que el modal es el mismo que componente que se utiliza para la confirmación de otros elementos, se omitirá su imagen.

Asimismo, se realizará una transacción que borré todos los datos que contengan el id del usuario.

Ver historial de gastos e ingresos

Existe un historial de gastos y otro de ingresos, en ambos, por defecto, aparecen los del mes actual. Se va a explicar el historial de gastos como ejemplo, ya

que el de ingresos es igual, pero con las llamadas API correspondientes a los mismos.

El usuario puede cambiar la fecha del historial a través de un modal que se ha implementado para ello. Además, es un componente que será reutilizado en más partes de la aplicación.

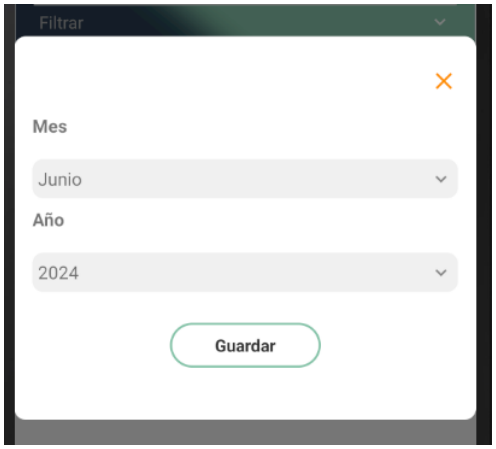


IMAGEN 41 - MODAL SELECTOR DE FECHA

Asimismo, se podrán filtrar por categoría y ordenar por diferentes opciones; cantidad: de mayor a menor y viceversa y fecha: más recientes y antiguos. También se añadió un buscador para encontrar gastos por su concepto o el nombre de la cuenta.

IMAGEN 42 - FILTROS HISTORIAL

Para implementarlo, se han tenido varias cosas en cuenta, entre ellas que, si se ha seleccionado previamente un orden y se restablecen los filtros de categoría, se mantenga ese orden una vez se restablecidas. También que, si hay filtros aplicados y se cambia la fecha, se mantengan dichos filtros y se muestren los datos con esas características.

Para mostrar cada gasto o ingreso se ha creado un componente reutilizable que muestra el icono de la categoría que se seleccionó al crear el gasto o ingreso, el concepto, la cantidad, la cuenta con la que ha sido pagado o en la que se ha ingresado y la fecha. En cada elemento se podrá hacer clic para cuando se implemente la opción de *Ver gasto* o *Ver ingreso*. A continuación, se puede observar esto que se comenta.

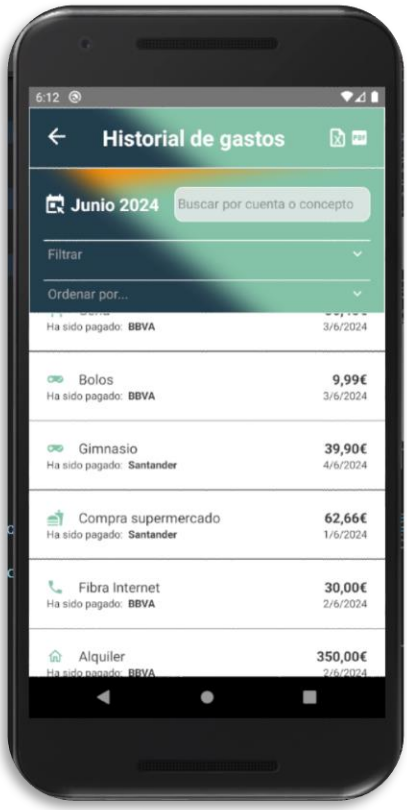


IMAGEN 43 - HISTORIAL DE GASTOS

Se han implementado varias APIs para estas vistas: obtener gastos por fechas, obtener las categorías del usuario de tipo gastos o ingreso, obtener gastos según la categoría, obtener los ingresos y obtener los ingresos según la categoría.

VER LISTADO DE CUENTAS Y ELIMINAR CUENTA

Para listar las cuentas se ha implementado otro componente reutilizable. En este caso muestra el nombre de la cuenta, el saldo total y un icono para eliminar. Cuando se quiere eliminar una cuenta, aparece un modal de confirmación. Si la cuenta se elimina, se eliminan los gastos e ingresos asociados, y se actualiza el balance en caso de que corresponda.

Si se hace clic sobre alguna cuenta, se redirigirá a la pantalla para ver los detalles y editar la cuenta.

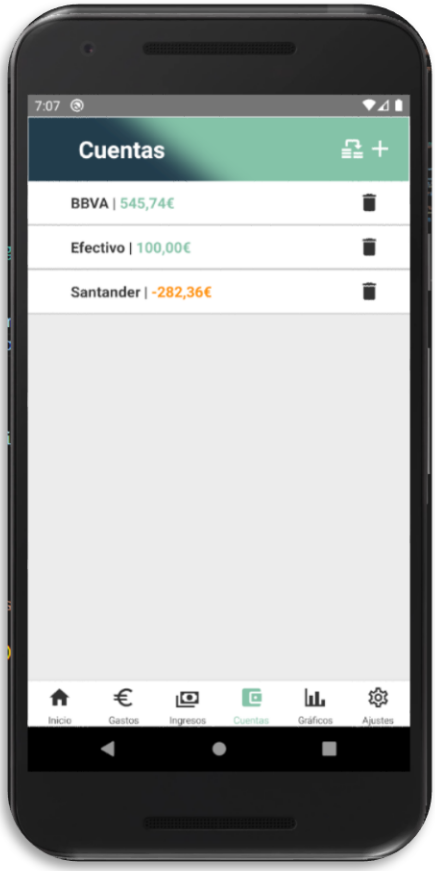


IMAGEN 44 - LISTADO DE CUENTAS

VER Y EDITAR CUENTA

Los datos que se muestran al ver una cuenta son: el saldo total, total de gastos y de ingresos. Así como los datos actuales de la cuenta. El saldo inicial no se puede modificar.

Como se muestra en el código, si se activa la opción de *Incluir con el saldo* se añade la referencia del saldo total a la cuenta, en caso contrario, se elimina la referencia existente, y se actualiza el saldo total de la aplicación.

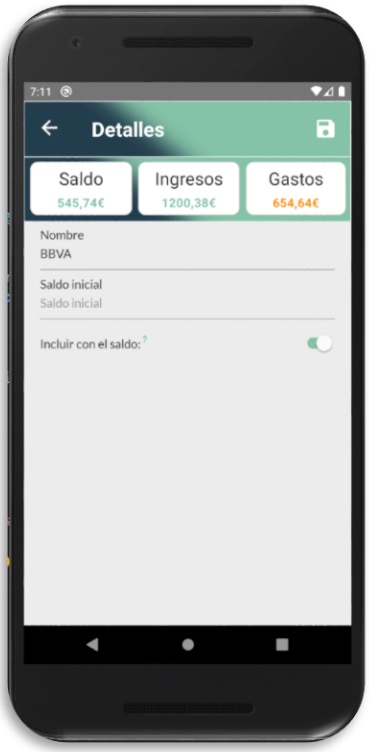


IMAGEN 45 - VER Y EDITAR CUENTA

```
const updateAccount = async (accountId, body, user, session) =>
{
    const { name, icon, isBalance } = body;

    let account;
    if (isBalance)
    {
        const balance = await Balance.find({ user })

        account = await Account.findByIdAndUpdate(
            accountId,
            { name, user, icon, isBalance, balance: balance._id },
            { new: true },
        ).session(session)
    }
    else account = await Account.findByIdAndUpdate(
        accountId,
        { name, user, icon, isBalance, balance: null },
        { new: true },
    ).session(session)

    await updateBalance(user, session)
    return account
}
```

IMAGEN 46 - CÓDIGO EDITAR CUENTA

EDITAR DATOS DE USUARIO

El usuario podrá editar su nombre, apellido, y correo, para ello se muestran en cada campo los datos que ya están almacenados. Tanto en la interfaz, como en la API se validan los campos, todos obligatorios, evitando que no se actualicen datos vacíos.

VER GRÁFICOS

En esta pantalla principal, se observa una comparación de gastos e ingresos a lo largo del año actual en una gráfica lineal múltiple y dos botones para seleccionar Ingresos o Gastos, que redirigen a las gráficas correspondientes.

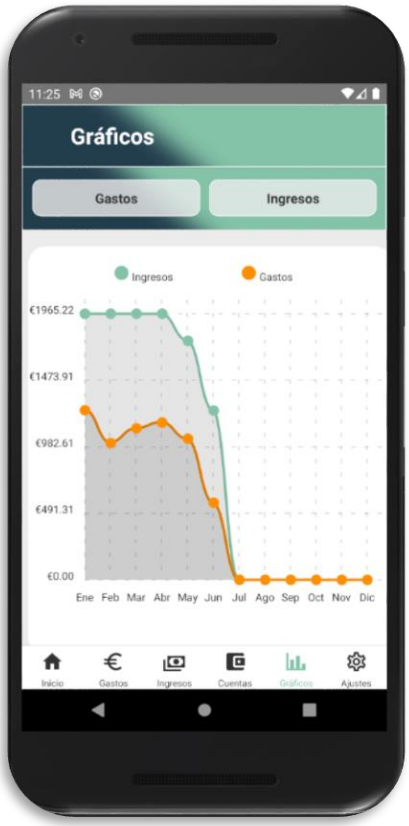


IMAGEN 47 - PANTALLA PRINCIPAL DEL APARTADO GRÁFICOS

Los tipos de gráficas que se muestran tanto para gastos como para ingresos son:

- **Mensuales**: aquí se muestra una gráfica de barras con los gastos o ingresos del mes actual agrupados por categorías. El usuario podrá seleccionar la fecha que desea visualizar.

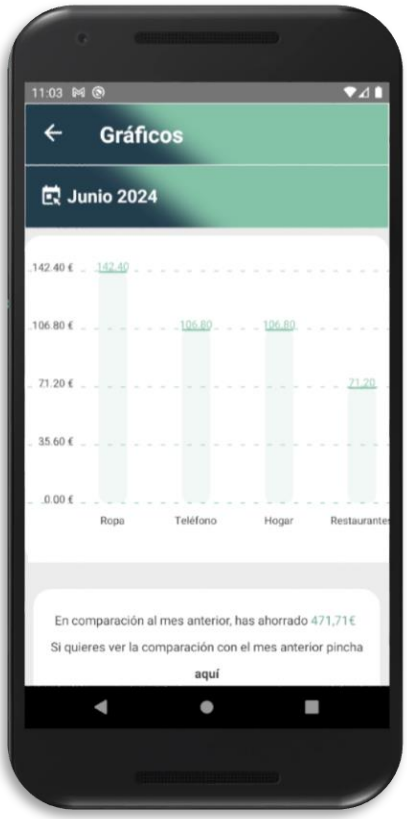


IMAGEN 48 - GASTOS MENSUALES AGRUPADOS POR CATEGORÍAS

- **Anuales**: en este caso, se encuentran dos gráficas: una lineal múltiple y otra de pastel. En la gráfica lineal se muestran los meses del año en el eje x y la cantidad gastada o ingresada (según corresponda) en el eje y. Cada línea de la gráfica representa una categoría, por lo que el usuario podrá comparar los gastos o ingresos por categoría a lo largo del año. Por defecto, se mostrará el año actual. La gráfica de pastel muestra los mismos datos, pero en porcentajes.

IMAGEN 49 – GASTOS ANUALES AGRUPADOS POR CATEGORÍAS

- **Categorías anuales**: este gráfico permite visualizar los gastos o ingresos de una sola categoría a lo largo del año. Esta gráfica se ha pensado por si el número de categorías es muy elevado, aquí se podrá ver de forma más clara, cada categoría en detalle.

IMAGEN 50 – GASTOS ANUALES POR CATEGORÍA

- **Método de pago mensual:** gráfica de pastel que muestra los gastos o ingresos mensuales agrupados por categoría según la cuenta seleccionada. Además, se incluyen los porcentajes, visualmente para que sea más claro de identificar. Por defecto, se muestra el mes actual y ninguna cuenta seleccionada.

IMAGEN 51 – GRÁDICO DE GASTOS MENSUALES SEGÚN LA CUENTA

- **Método de pago anual:** similar a la anterior con la diferencia de que, en vez de ser los gastos o ingresos mensuales, son anuales.

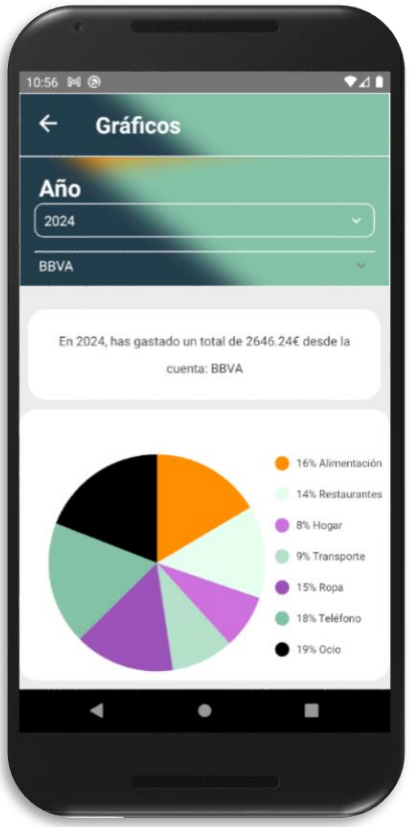


IMAGEN 52 – GRÁFICO GASTOS ANUALES SEGÚN LA CUENTA

- **Comparación por fechas:** aquí se permite comparar dos fechas del año. En el eje x aparecen las categorías, y en el eje y la cantidad. Por defecto aparece la comparación del mes actual con el mes anterior. Además, se han incluido los datos en forma de tabla.

IMAGEN 53 – GRÁFICO COMPARACIÓN POR FECHAS

- **Predicción:** gráfica lineal múltiple. Se observan los gastos del año en curso, los del año anterior, y una predicción del total de gastos para este mes.

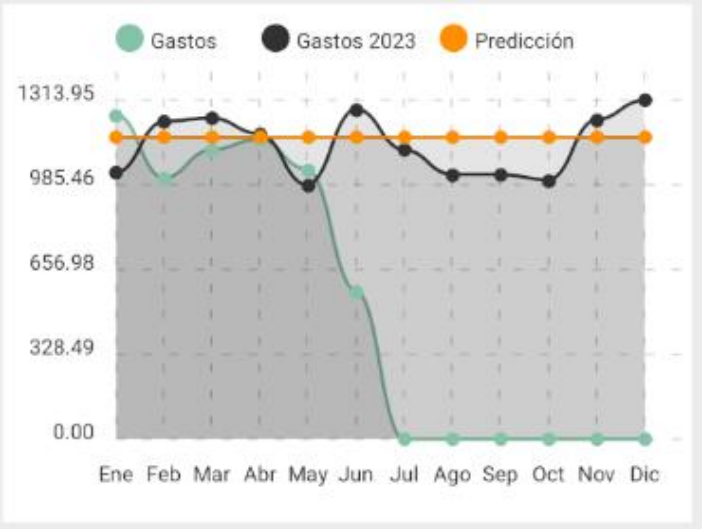


IMAGEN 54 – GRÁFICO CON PREDICCIÓN DE GASTOS PARA EL MES ACTUAL

- **Progreso del presupuesto:** gráficas de pastel. Se puede observar el progreso del mes en curso del presupuesto establecido para cada categoría.

IMAGEN 55 - GRÁFICO CON EL PROGRESO DEL PRESUPUESTO POR CATEGORÍAS

CREAR GASTO FIJO

Los gastos fijos permiten la creación de gastos automáticos según el periodo que se establezca. Está pensado para aquellos gastos recurrentes con el mismo importe, por ejemplo, el pago del alquiler.

El formulario del gasto fijo incluye los mismos campos que para crear un gasto sin tener en cuenta la conversión de divisas y, con los campos adicionales de fecha de inicio, ¿Tiene fecha final?, fecha fin y periodo. El campo fecha fin, solo será visible si previamente se ha marcado el campo "¿Tiene fecha final?".

Las validaciones de este formulario más destacables serían comprobar que:

- La fecha de fin tiene que ser posterior a la fecha de inicio.
- La fecha de inicio no puede ser anterior a la fecha actual.
- Si se selecciona un periodo semanal, la fecha final tiene que ser mínimo 1 semana después de la de inicio.
- Si se selecciona un periodo mensual, la fecha final tiene que ser mínimo un mes posterior a la fecha de inicio.
- Si selecciona un periodo anual, la fecha final tiene que ser mínimo un año posterior a la fecha de inicio.

IMAGEN 56 - FORMULARIO GASTO FIJO

Además, de los campos que se muestran en el formulario, se incluyen próxima y última inserción, y estado. Al crearlo, el estado se establece en activo.

Cuando un gasto fijo es añadido, se comprueba la fecha de inicio, en caso de que coincida con la fecha actual. Se realiza una inserción automática de un gasto y se calcula cuando será la próxima inserción teniendo en cuenta el periodo establecido y se establece la fecha de última inserción la fecha actual.

Si el inicio del gasto fijo está programado para más adelante, se establecerá como fecha de inicio la fecha indicada.

Para la inserción automática de gastos, se ha implementado un *cron job*[5] en el servidor. Se ha programado para que se ejecute todos los días a las 12 de la noche. Esto se ha realizado de la siguiente forma estableciendo '0 0 * * *' en el cron.

1. Minuto. En este caso en el minuto 0.
2. Hora. 0 indica principio de cada hora.
3. Día del mes
4. Mes
5. Día de la semana

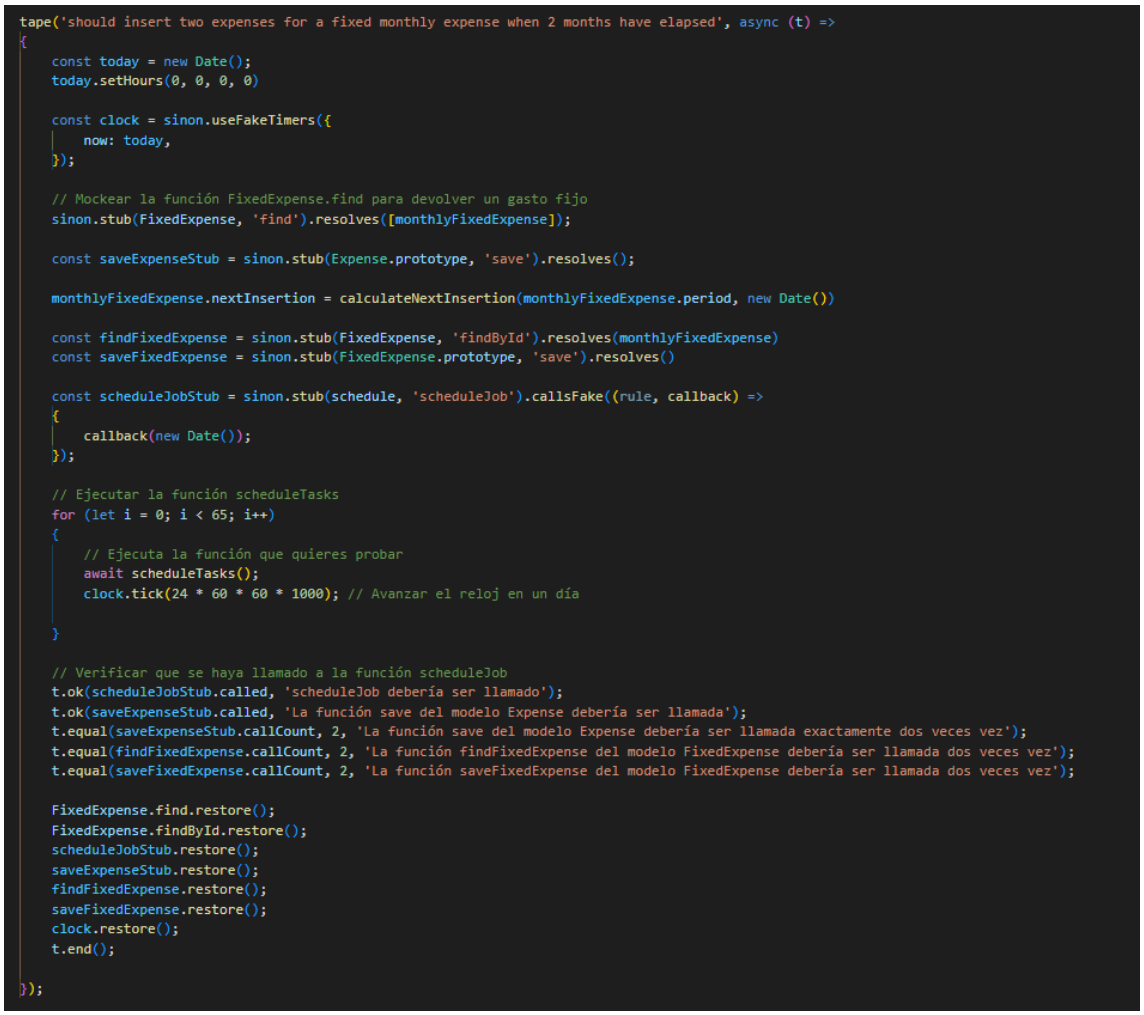
El * indica en cualquier momento.

Como no es posible mantener el servidor ejecutándose diariamente debido a que se está ejecutando en local, se realizaron tests para comprobar su correcto funcionamiento. Para ello se utilizaron las librerías: *tape* para crear las pruebas, y *sinon* para crear *stubs*[6]. Asimismo, se utiliza el método *useFakeTimers* para simular el transcurso del tiempo, incrementando el reloj en un día con clock.tick(24 * 60 * 60 * 1000)

Se han utilizado stubs para simular los métodos de guardar y encontrar en base de datos. En el ejemplo que se muestra a continuación, se implementa un test para comprobar que un gasto fijo mensual y sin fecha final, crea dos gastos a lo largo de 65 días. Para comprobarlo se asegura que se haya llamado al método de crear gasto, y guardar gasto fijo dos veces.

[5] Programa de utilidad para repetir tareas en un momento posterior.**Fuente especificada no válida.**

[6] Objeto falso que se usa en lugar de uno real con el propósito de hacer que el programa se comporte de forma necesaria para poder testearlo. (cesc1989, 2022)



```
tape('should insert two expenses for a fixed monthly expense when 2 months have elapsed', async (t) =>
{
    const today = new Date();
    today.setHours(0, 0, 0, 0)

    const clock = sinon.useFakeTimers({
        now: today,
    });

    // Mockear la función FixedExpense.find para devolver un gasto fijo
    sinon.stub(FixedExpense, 'find').resolves([monthlyFixedExpense]);

    const saveExpenseStub = sinon.stub(Expense.prototype, 'save').resolves();

    monthlyFixedExpense.nextInsertion = calculateNextInsertion(monthlyFixedExpense.period, new Date())

    const findFixedExpense = sinon.stub(FixedExpense, 'findById').resolves(monthlyFixedExpense)
    const saveFixedExpense = sinon.stub(FixedExpense.prototype, 'save').resolves()

    const scheduleJobStub = sinon.stub(schedule, 'scheduleJob').callsFake((rule, callback) =>
    {
        callback(new Date());
    });

    // Ejecutar la función scheduleTasks
    for (let i = 0; i < 65; i++)
    {
        // Ejecuta la función que quieres probar
        await scheduleTasks();
        clock.tick(24 * 60 * 60 * 1000); // Avanzar el reloj en un día

    }

    // Verificar que se haya llamado a la función scheduleJob
    t.ok(scheduleJobStub.called, 'scheduleJob debería ser llamado');
    t.ok(saveExpenseStub.called, 'La función save del modelo Expense debería ser llamada');
    t.equal(saveExpenseStub.callCount, 2, 'La función save del modelo Expense debería ser llamada exactamente dos veces vez');
    t.equal(findFixedExpense.callCount, 2, 'La función findFixedExpense del modelo FixedExpense debería ser llamada dos veces vez');
    t.equal(saveFixedExpense.callCount, 2, 'La función saveFixedExpense del modelo FixedExpense debería ser llamada dos veces vez');

    FixedExpense.find.restore();
    FixedExpense.findById.restore();
    scheduleJobStub.restore();
    saveExpenseStub.restore();
    findFixedExpense.restore();
    saveFixedExpense.restore();
    clock.restore();
    t.end();

});
```

IMAGEN 57 - MÉTODO PARA TESTEAR CRON JOB

EDITAR GASTO FIJO

Editar un gasto fijo incluye también la funcionalidad de activar o finalizar según si el estado es inactivo o activo respectivamente, influyendo en sí se siguen incluyendo o no los gastos recurrentes. Si la fecha de inicio se ha establecido el día actual, se realiza una inserción del gasto, sino se establecerá la fecha de próxima inserción, el día establecido.

EXPORTAR DATOS

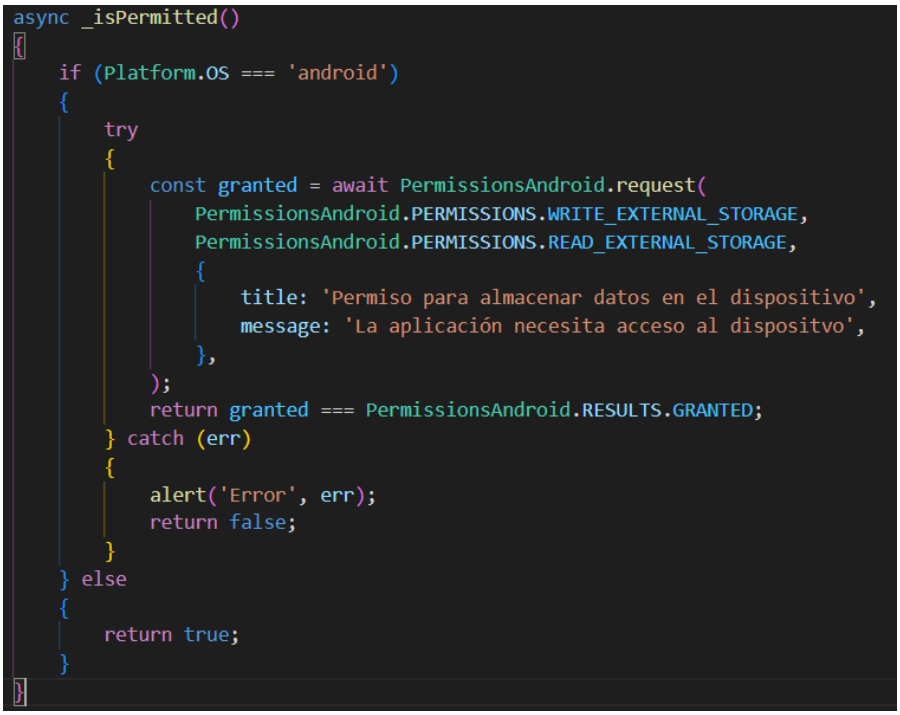
Existen dos formas para exportar datos, en formato Excel o un informe PDF. Este último está solo disponible desde el historial de gastos. Exportar Excel se permite realizarlo desde ambos historiales. Se exportarán los gastos o ingresos según el historial correspondiente.

Se utilizan las librerías *react-native-fs para escribir el archivo en el dispositivo, xlsx* para generar un Excel y *react-native-html-to-pdf* para generar el PDF convirtiendo el código HTML en PDF.

Antes de exportar datos, se necesita solicitar permisos para guardar datos en el dispositivo móvil, para ello se debe configurar el archivo AndroidManifest, añadiendo los permisos de lectura y escritura.

IMAGEN 58 - CONFIGURACIÓN PERMISOS ANDROID

Este código valida si se ha permitido el acceso para almacenar datos, se utilizará para comprobar si se pueden exportar los archivos o no.



```
async _isPermitted()
{
    if (Platform.OS === 'android')
    {
        try
        {
            const granted = await PermissionsAndroid.request(
                PermissionsAndroid.PERMISSIONS.WRITE_EXTERNAL_STORAGE,
                PermissionsAndroid.PERMISSIONS.READ_EXTERNAL_STORAGE,
                {
                    title: 'Permiso para almacenar datos en el dispositivo',
                    message: 'La aplicación necesita acceso al dispositvo',
                },
            );
            return granted === PermissionsAndroid.RESULTS.GRANTED;
        } catch (err)
        {
            alert('Error', err);
            return false;
        }
    } else
    {
        return true;
    }
}
```

IMAGEN 59 – MÉTODO PARA COMPROBAR SI EL USUARIO HA ACEPTADO LOS PERMISOS

Los datos que se exportan en Excel son: concepto, cantidad, cuenta, categoría y fecha. En los gastos también se incluyen las columnas gasto fijo (sí o no dependiendo si es un gasto fijo), y periodo.

El PDF que se genera es el siguiente:

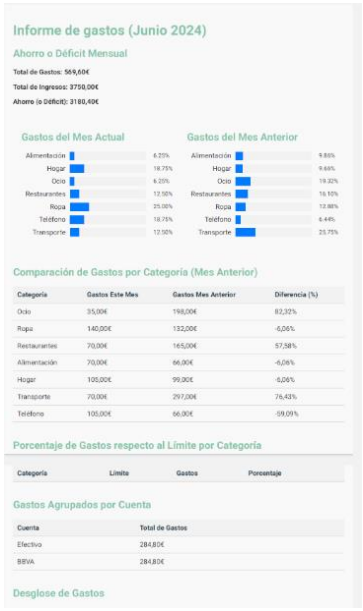


IMAGEN 60 - INFORME PDF (1° PARTE)

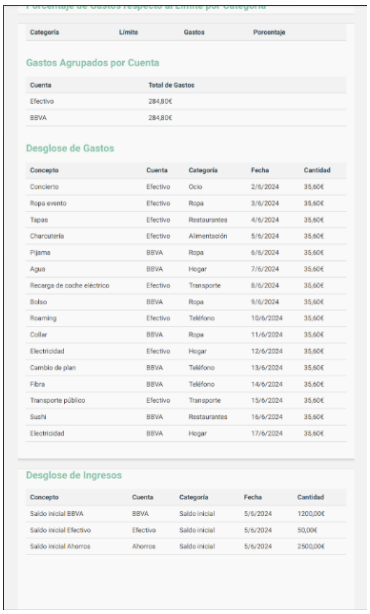


IMAGEN 61 - INFORME PDF (2° PARTE)

VER GASTOS FIJOS Y FILTRARLOS

El listado de gastos fijos muestra por separado los activos y finalizados facilitando su búsqueda. Además, se ha incluido un buscador para encontrarlos por nombre, que mostrará los que se encuentren, tanto activos como finalizados. Si el usuario borra la búsqueda aparecerán de nuevo todos los gastos fijos.

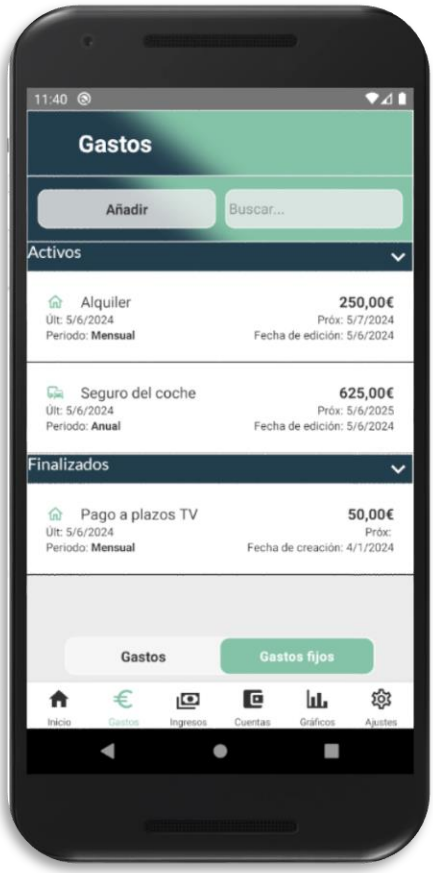


IMAGEN 62 - LISTADO DE GASTOS FIJOS

Esta es la función implementada para realizar la búsqueda. Para no distinguir entre mayúsculas o minúsculas, el filtrado se realiza convirtiendo el concepto del gasto fijo y la palabra introducida a minúsculas y luego se comprueba que el concepto contenga lo escrito en el campo de búsqueda. De esta manera se permite una búsqueda más amplia y sin tantas restricciones.

```
_findFixedExpense(value)
{
    let filteredExpenses;
    let valueLower = value.toLowerCase();

    if (value === '') filteredExpenses = this.props.fixedExpenses

    else filteredExpenses = this.props.fixedExpenses.filter(expense => expense.concept.toLowerCase().includes
    (valueLower));

    this.setState({ name: value, filteredExpenses })
}
```

IMAGEN 63 - MÉTODO PARA ENCONTRAR GASTOS FIJOS POR CONCEPTO

TRANSFERIR DINERO ENTRE CUENTAS

La transferencia entre cuentas agiliza las transacciones, ya que evita al usuario tener que introducir un gasto y un ingreso manualmente. De esta manera el usuario solo necesitará introducir un concepto, la cantidad que se transferirá y las cuentas de origen y destino. Además de las validaciones de tipo de campo y campos obligatorios, también se ha validado que las cuentas de origen y destino no pueden ser las mismas.

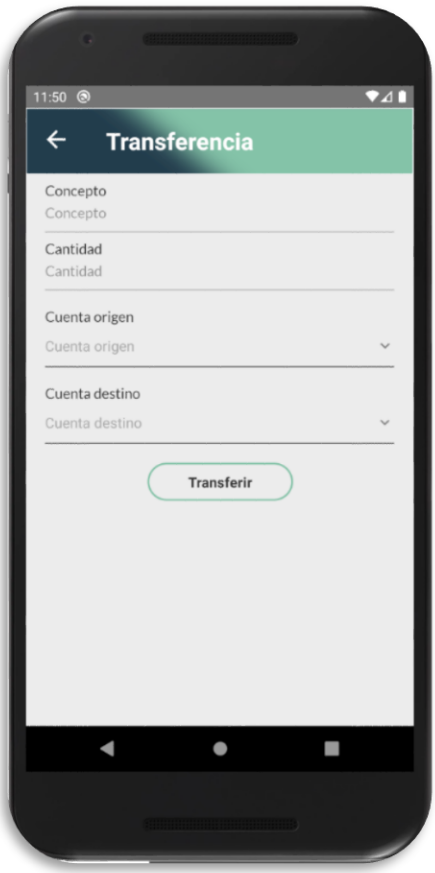


IMAGEN 64 - TRANSFERENCIA ENTRE CUENTAS

La transferencia conlleva realizar un gasto en la cuenta de origen y un ingreso en la de destino. Para este caso se ha añadido la categoría “Transferencia” por defecto cuando se registra un usuario. Dicha categoría solo será utilizada para estos casos. Asimismo, la fecha que incluirán el gasto y el ingreso será la del momento en la que se realiza la transferencia.

Se ha utilizado las transacciones para asegurar que, si ocurre algún fallo, no aparezcan gastos o ingresos que no corresponden.

```
const transfer = async (user, props, session) =>
{
    const { fromAccount, toAccount, amount, concept } = props;

    const userExists = await User.findById(user)
    if (!userExists) throw new NotFoundException(`User with id ${user} not found`)

    const category = await Category.findOne({ user, name: "Transferencia" })
    if (!category) throw new NotFoundException(`Category not found`)

    const sourceAccount = await Account.findById(fromAccount)
    if (!sourceAccount) throw new NotFoundException(`Account with id ${fromAccount} not found`)

    const destinationAccount = await Account.findById(toAccount)
    if (!destinationAccount) throw new NotFoundException(`Account with id ${toAccount} not found`)

    const date = new Date()

    await createExpense(user, { account: fromAccount, date, category: category._id, amount, concept }, session)

    await createIncome(user, { account: toAccount, date, category: category._id, amount, concept }, session)

    return
}
```

IMAGEN 65 - MÉTODO TRANSFERENCIA ENTRE CUENTAS

SKETCH

Se ha realizado un *mockup* previo a la implementación de la aplicación. Se ha dividido por secciones para una mejor visualización debido a su gran tamaño. Las secciones que se encuentran a continuación son:

- **Inicio**: pantalla principal de la aplicación.
- **Registro e inicio de sesión.**
- **Ajustes**: pantallas del apartado de ajustes y de cada una de las acciones que se pueden realizar.
- **Ingresos:** pantalla principal del apartado, y las de añadir, ver detalles e historial
- **Gastos:** pantalla principal del apartado, y las de añadir, ver detalles e historial.
- **Cuentas:** pantalla principal y las de añadir, editar y transferir entre cuentas.
- **Gráficos:** pantalla principal y ejemplo de cómo se podrían representar las gráficas.

INICIO

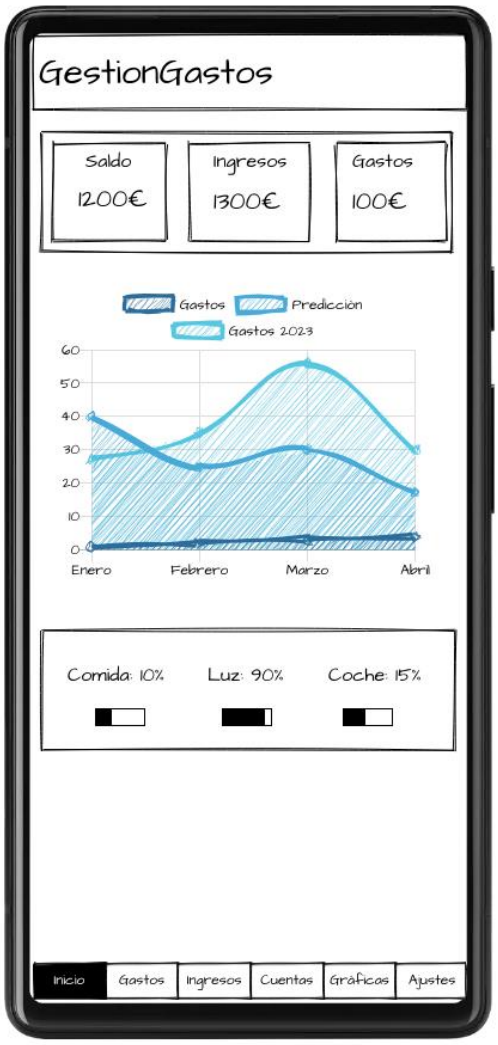


IMAGEN 66 - MOCKUP. PANTALLA PRINCIPAL

REGISTRO E INICIO DE SESIÓN

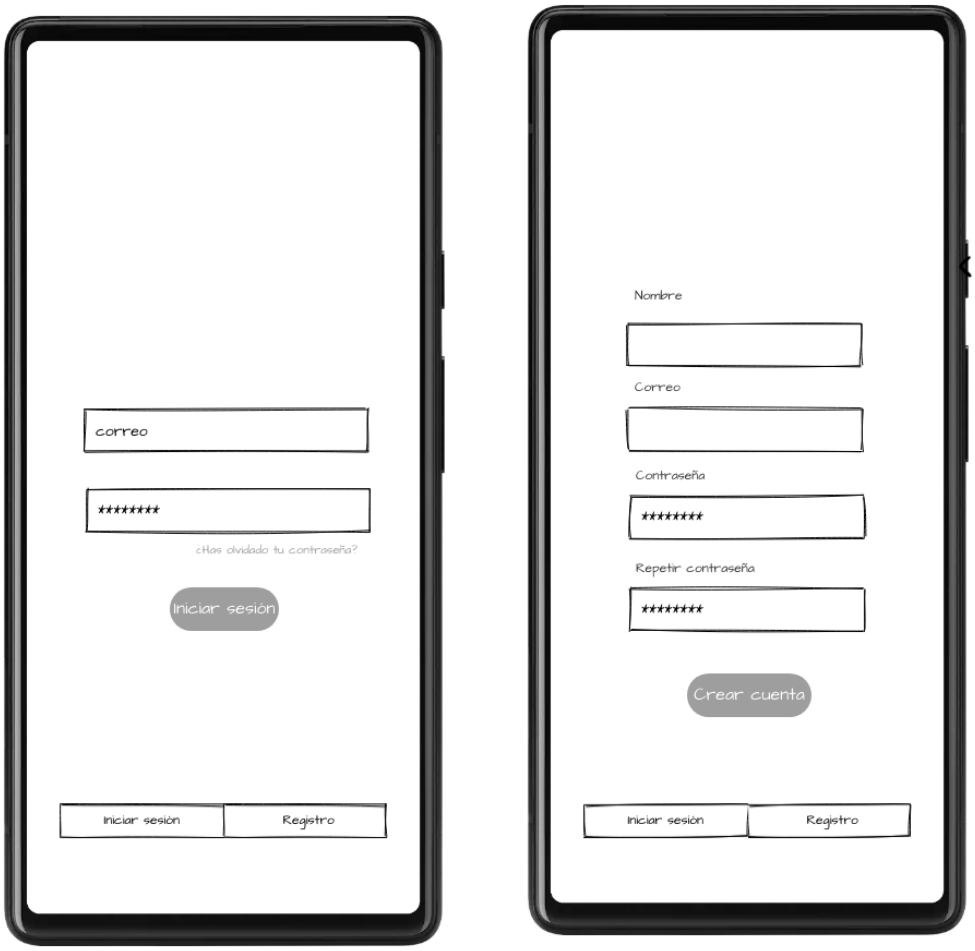


IMAGEN 67 - MOCKUP. PATALLA DE REGISTRO E INICIO DE SESIÓN

AJUSTES

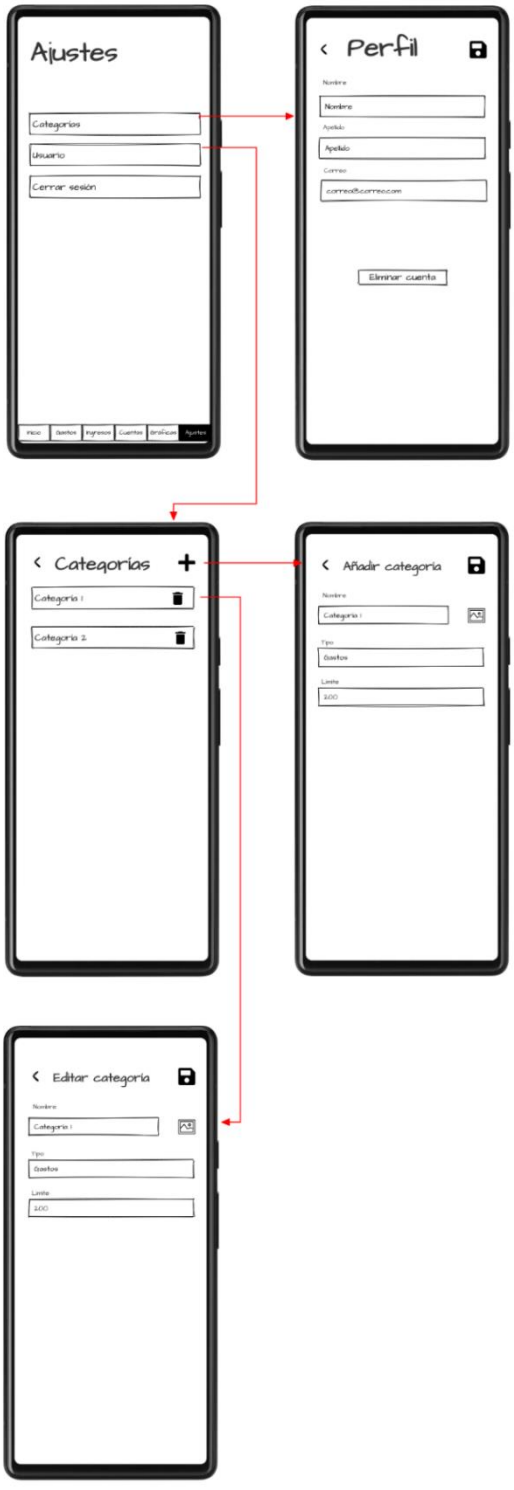


IMAGEN 68 - MOCKUP. PANTALLAS DE AJUSTES

INGRESOS

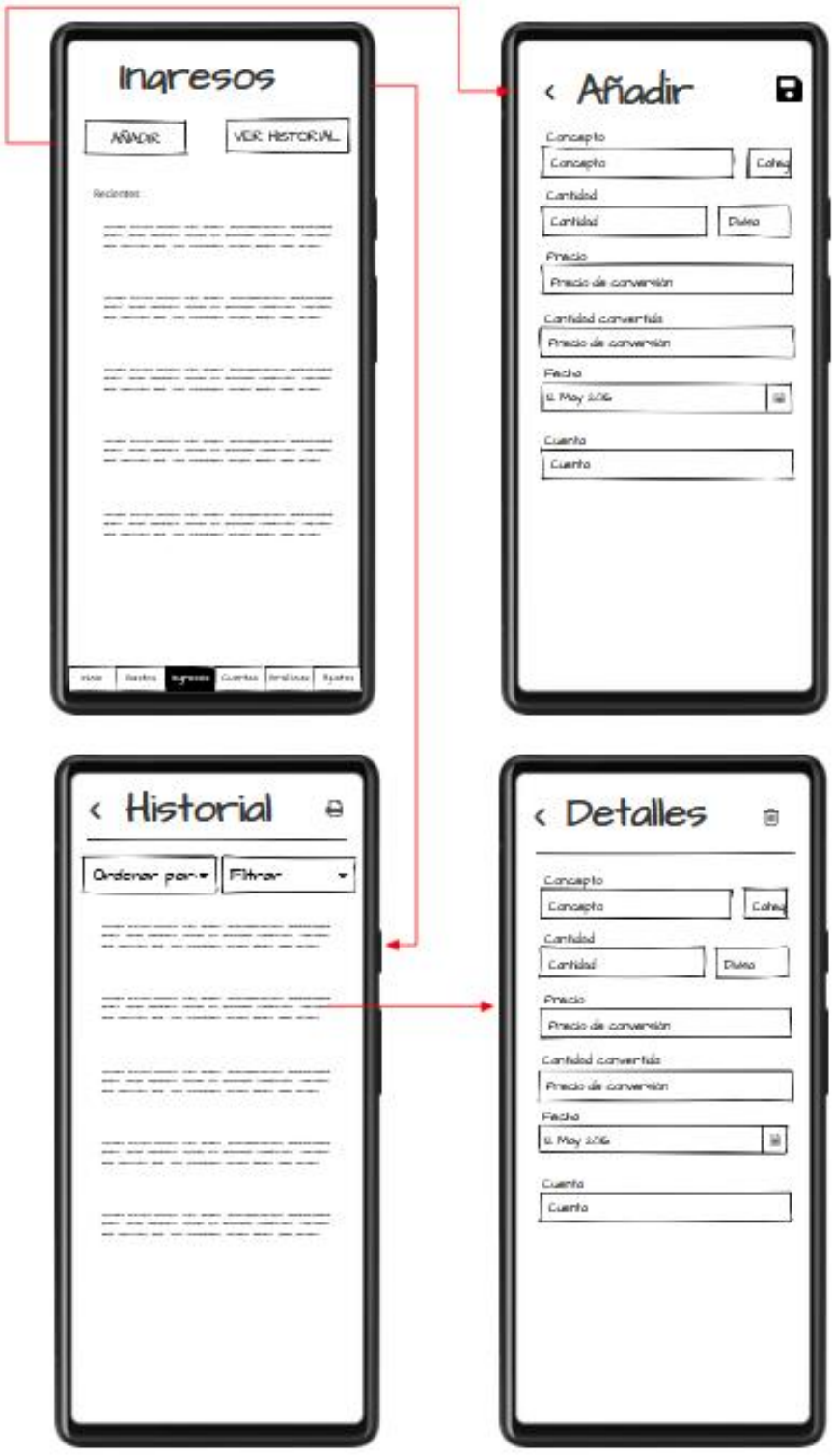


IMAGEN 69 - MOCKUP. PANTALLAS DE INGRESOS

GASTOS

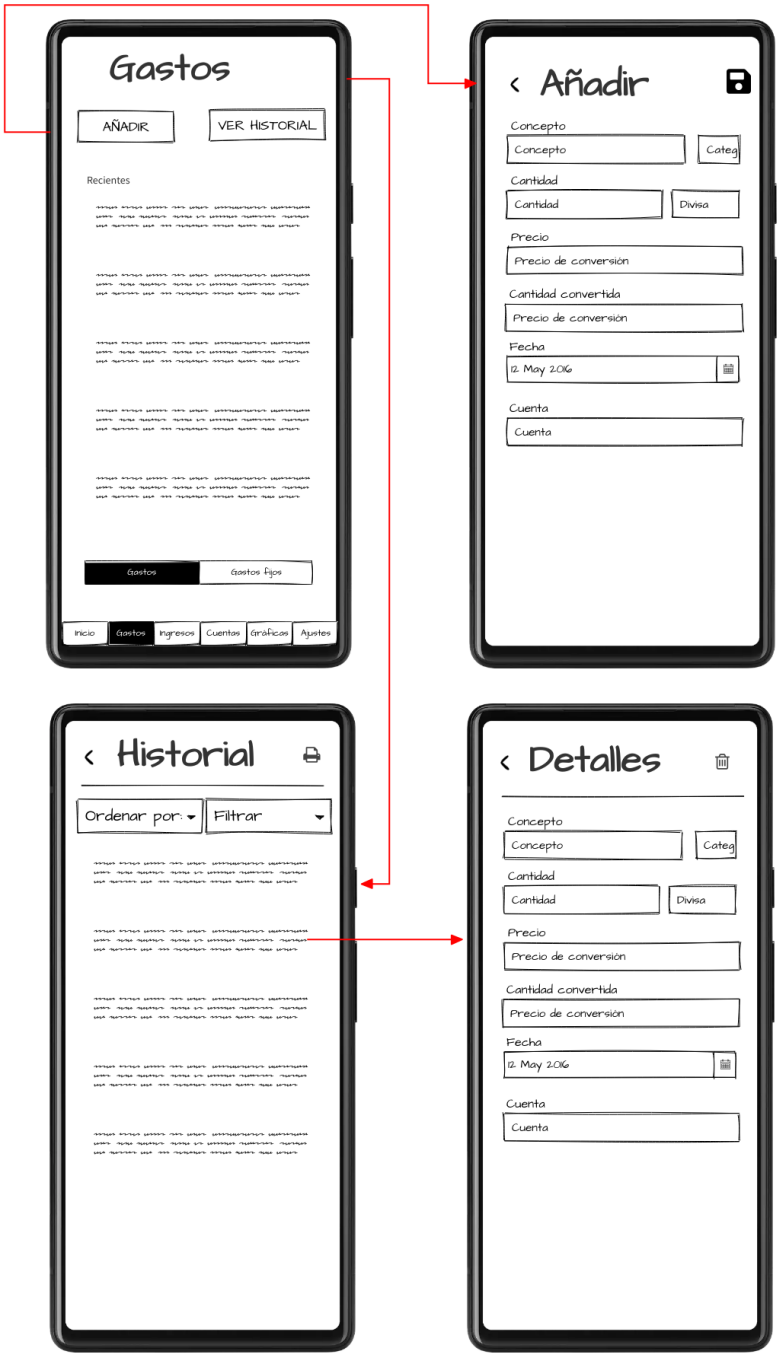


IMAGEN 70 - MOCKUP. PANTALLAS DE GASTOS

CUENTAS

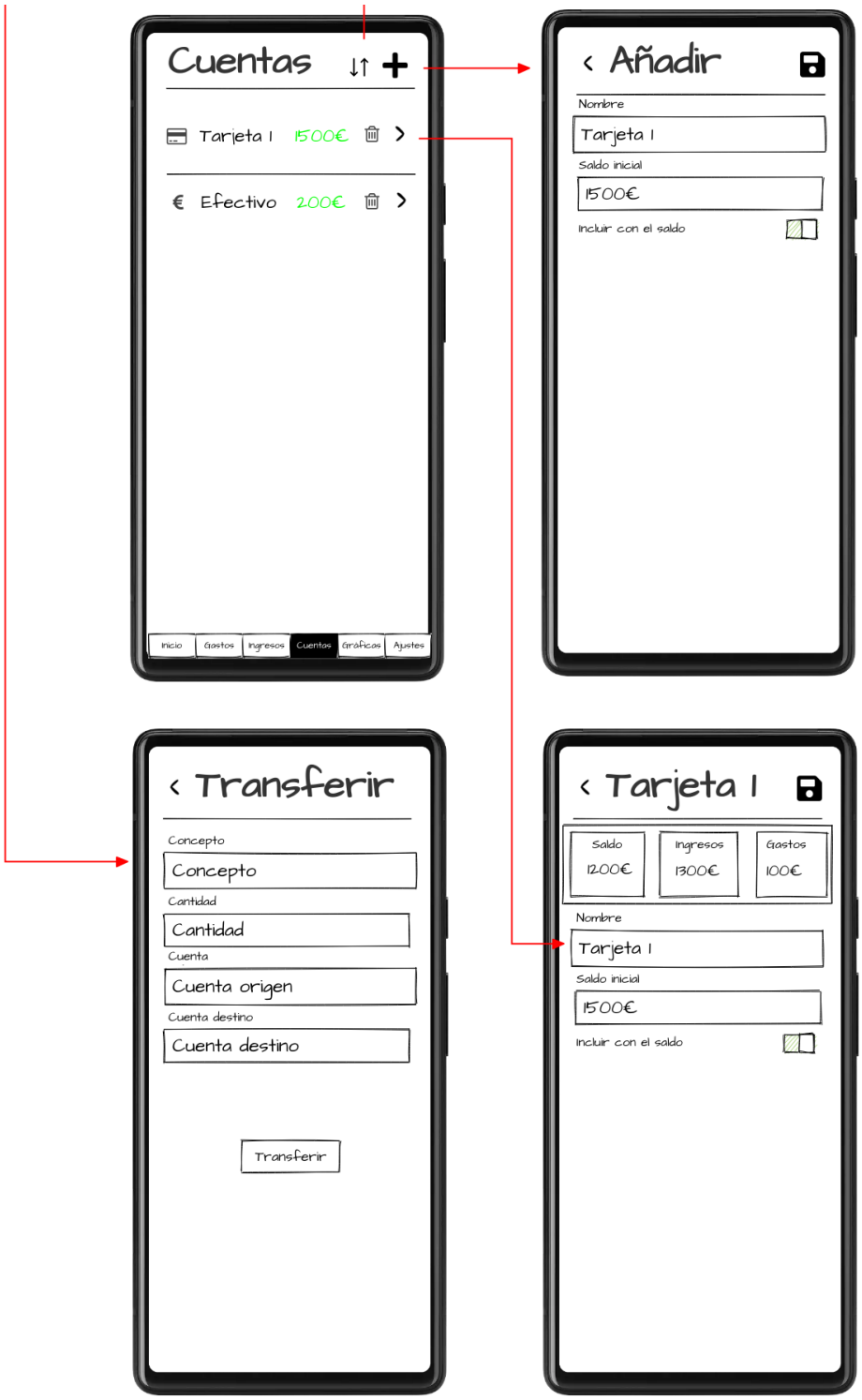


IMAGEN 71 - MOCKUP. PANTALLAS DE CUENTAS

GRÁFICOS

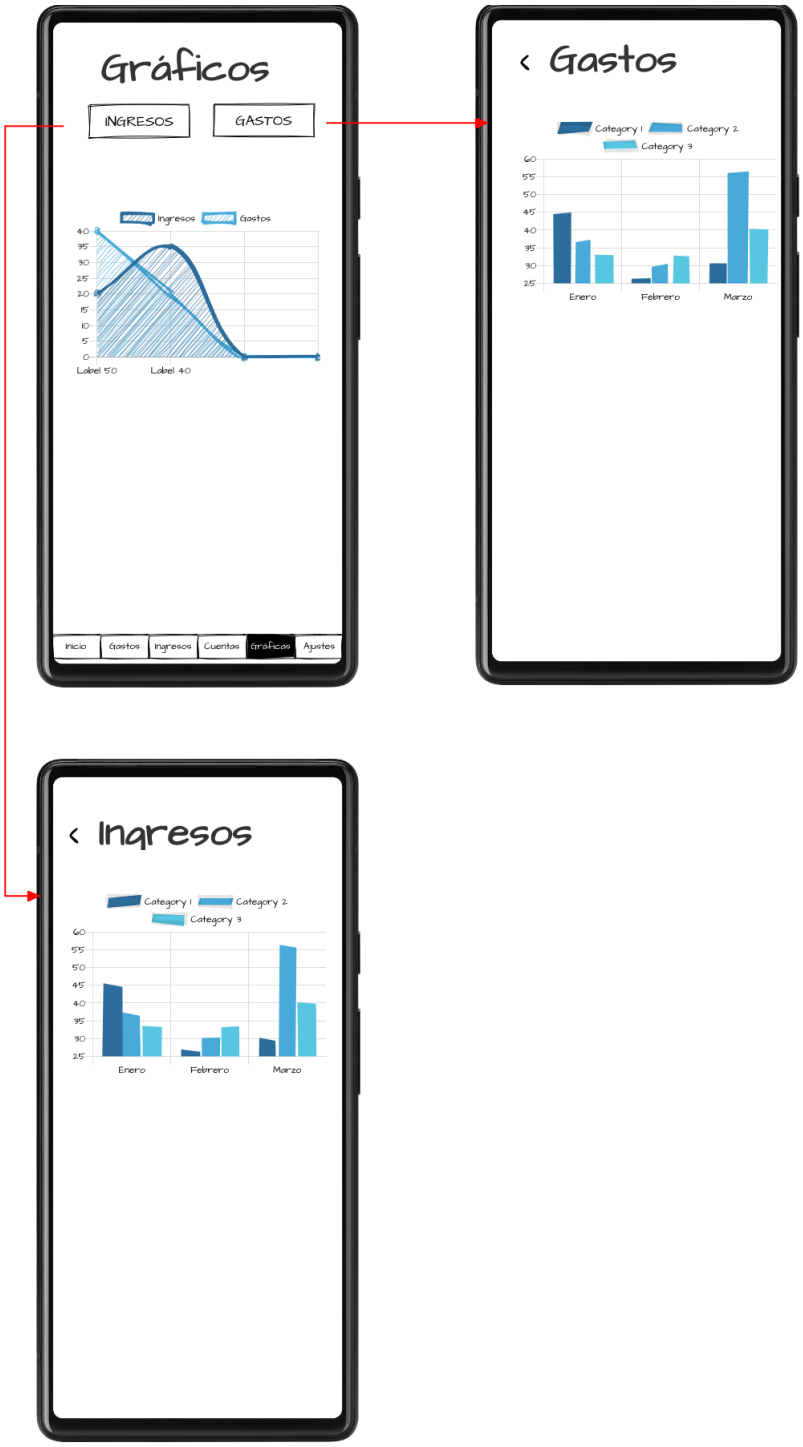


IMAGEN 72 - MOCKUP. PANTALLAS DE GRÁFICOS

PALETA DE COLORES

En este apartado, se mostrará la paleta de colores que se ha elegido para la aplicación. Con estos colores se define la identidad de marca, consiguiendo con ellos que los usuarios al visualizar este conjunto de colores lo relacionen con la aplicación.

En la tabla que se muestra a continuación, se detalla para qué se utiliza cada color, el porqué de su elección y sus respectivos códigos en hexadecimal y RGB.

	Uso	Significado	Hex	RGB
Azul oscuro	Fondo y cabecera de la aplicación	Representa elegancia, seriedad y confianza	#233D4D	35, 61, 77
Verde menta	Botones y elementos resaltados	Transmite armonía, equilibrio y calma	#84C3A8	132, 195, 168
Gris claro	Fondo de la página y secciones	Proporciona un ambiente suave y agradable	#ECECEC	236, 236, 236
Gris oscuro	Texto y elementos de la interfaz	Sugiere formalidad, elegancia y sofisticación	#333333	51, 51, 51
Naranja	Gráficos y datos positivos	Representa energía, positividad y entusiasmo	#FF8F00	255, 143, 0
Blanco	Modales, fondos de gráficas, y resto de fondos secundarios	Realzar contraste con el fondo existente	#FFFFFF	256, 256, 256

TABLA 32 – EXPLICACIÓN PALETA DE COLORES

En definitiva, esta composición de colores elegida tiene como objetivo transmitir a los usuarios dos aspectos importantes: por un lado, la elegancia y seriedad de la gestión de sus gastos y, por otro lado, una sensación positividad y equilibro al poner a su disposición una herramienta para controlar su economía y, por consiguiente, se alcance un mayor equilibrio en las finanzas personales.

233D4D	84C3A8	ECECEC	333333	FF8F00	FFFFFF

TABLA 33 - PALETA DE COLORES

TIPOGRAFÍA

En cuanto a la tipografía elegida, es gratuitas y ha sido obtenida de Google Fonts. Posee la característica de que es sans-serif, es decir, sin remate. Variará su uso según el contexto, es decir, se utilizará en negrita o cursiva según se requiera.

ICONOGRAFÍA

Los iconos han sido seleccionados de la librería MaterialCommunityIcons que permite su integración con react-native. Los iconos permiten una rápida identificación de los elementos en la interfaz. En la mayoría de los casos se han empleado como botones o como ayuda visual para reconocer las categorías.

La aplicación cuenta con una barra inferior de navegación con 6 botones - Inicio, Gastos, Ingresos, Cuentas, Gráficos y Ajustes – que redirigen al usuario a cada apartado de la aplicación. Por defecto, cuentan con un fondo oscuro, excepto cuando se ha seleccionado que cambia al color verde elegido para la aplicación.

IMAGEN 73 - BARRA DE NAVEGACIÓN

Además, dentro del apartado de ajustes, se han utilizado los siguientes iconos para cada uno de los apartados.

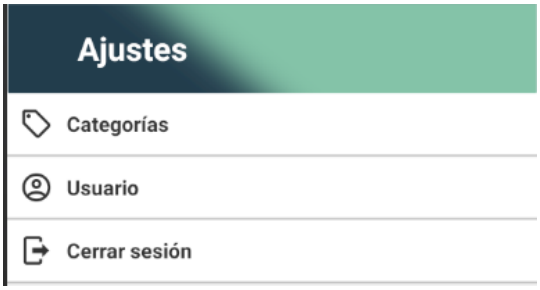


IMAGEN 74 - PANTALLA DE AJUSTES

El resto de los iconos de la aplicación, dependerán de los que el usuario elija para cada categoría creada. Se ha proporcionado todos los disponibles de la librería mencionada anteriormente.

Logo

El logo de la aplicación se ha diseñado con *Canva*. Se decidió utilizar un gráfico de líneas simbolizando el seguimiento de los gastos a lo largo del tiempo. Se optó que fuera en ascendente y verde para dar una sensación de prosperidad en la economía. El minimalismo de este se debe a que será más fácil de recordar y que al ser un logo de una aplicación y en la mayoría de los casos se verá como un elemento pequeño, no se quería saturar la imagen.

En un principio se diseñó con dos formatos, uno con el fondo en verde y el símbolo en blanco y viceversa, por si era necesario utilizarlo en varios contextos. Además, se diseñaron tanto con forma circular como cuadrada debido a la diversidad de presentaciones que prestan los diferentes dispositivos Android y sus versiones.

IMAGEN 75 - LOGOS

CONCLUSIÓN Y TRABAJOS FUTUROS

Este proyecto ha contribuido a un desarrollo personal y profesional. Desde su pensamiento, como idea propia, hasta su implementación utilizando un stack tecnológico que quería aprender y en el cual no tenía experiencia previa. He ido superando obstáculos a medida que iban surgiendo, lo que me ha incrementado mi capacidad para resolver problemas.

En el caso más personal, el estar trabajando a jornada completa a la misma vez que realizaba el proyecto, ha influido en que el tiempo de desarrollo de este haya sido mayor del esperado. Además de tener que emplear mayor tiempo del habitual en estar con el ordenador ha requerido mayor gestión del tiempo y energía, pero haber llegado hasta aquí me demuestra a mí misma la capacidad de superar desafíos.

Aportando una visión más objetiva, este proyecto finaliza con una aplicación con una utilidad real, que permite entre otras funcionalidades, añadir gastos e ingresos con conversión de divisas con valores de los precios en tiempo real, generación de informes PDFs, exportación de los datos a Excel, visualización de los datos en gráficas y utilización de un stack tecnológico innovador y escalable.

No obstante, dada la limitación de tiempo para realizar el proyecto, quedaron ideas y cosas pendientes para realizar en un futuro. Entre ellas están:

- Adaptar el código para que funcione en dispositivos IOS.
- Crear una función de recuperación de contraseña.
- Desplegar el servidor en la nube.
- Añadir cuentas y gastos compartidos para unidades familiares.
- Añadir más diversidad de gráficos para visualizar los datos, y organizarlos de una manera más útil.
- Sincronización de datos automáticamente desde cuentas bancarias online.
- Importar datos a través de Excel.

ANEXO

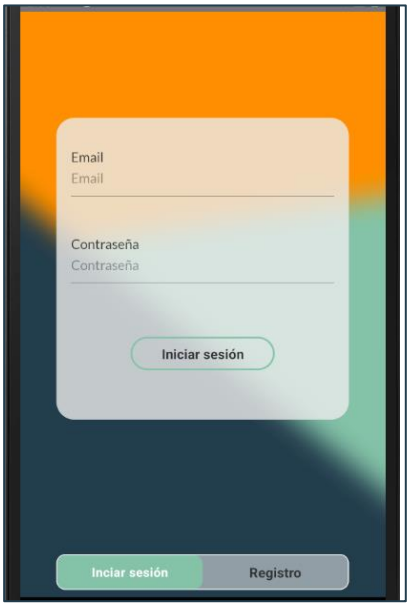
MANUAL DE USUARIO

AUTENTICACIÓN

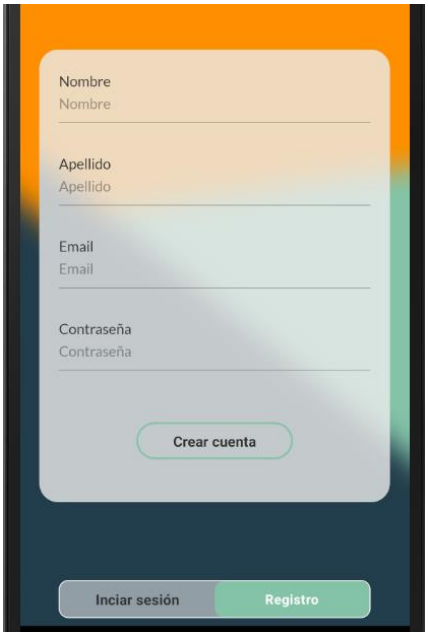
El formulario para iniciar sesión es la primera pantalla que se muestra cuando se inicia la aplicación.

En caso de ya tener una cuenta, deberá introducir el correo y contraseña y pulsar el botón “Iniciar sesión”.

Si el usuario aún no tiene cuenta, deberá hacer clic en el botón de “Registro” en la parte inferior de la pantalla. Esto le redirigirá al formulario para crear una cuenta.



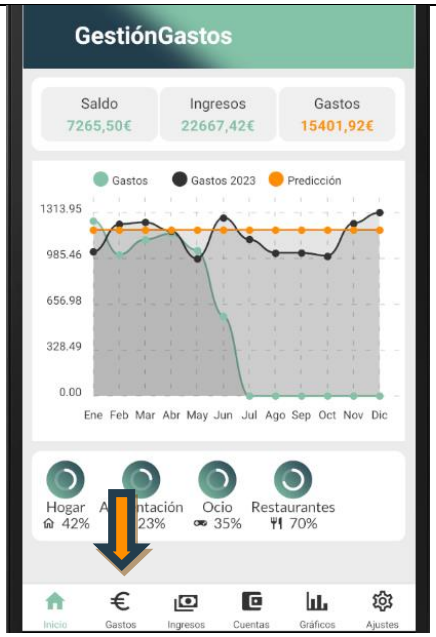
Para crear un usuario, se deberá introducir: nombre, apellido, correo y contraseña, y seleccionar “Crear cuenta”. Una vez creada la cuenta, deberá introducir los datos para iniciar sesión en el formulario que se mostró anteriormente.



GESTIONAR GASTOS

AÑADIR GASTO

El usuario deberá hacer clic en el menú horizontal inferior de la aplicación en el botón donde se lee “Gastos”.



Una vez ahí, hará clic en el botón “Añadir” en la parte superior.

Deberá introducir todos los datos correspondientes. Además, para seleccionar una categoría deberá pulsar sobre el botón que se indica.

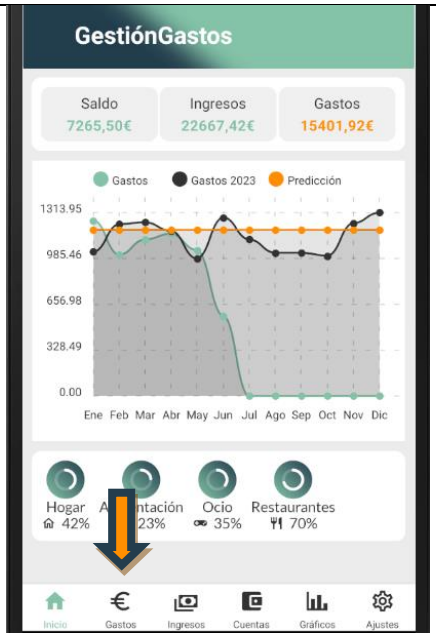
Una vez se hayan introducido los datos, se deberá clicar sobre el icono  y el gasto será guardado.

En caso de querer cancelar o volver

atrás, deberá pulsar sobre el icono de la esquina superior izquierda.

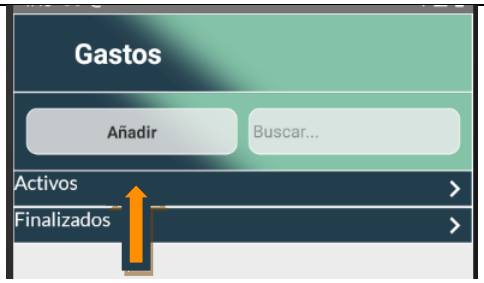
AÑADIR GASTO FIJO

Deberá hacer clic en el menú horizontal inferior de la aplicación en el botón donde se lee “Gastos”.



Una vez ahí, se deberá pulsar sobre el botón “Gastos fijos” como se indica en la imagen.

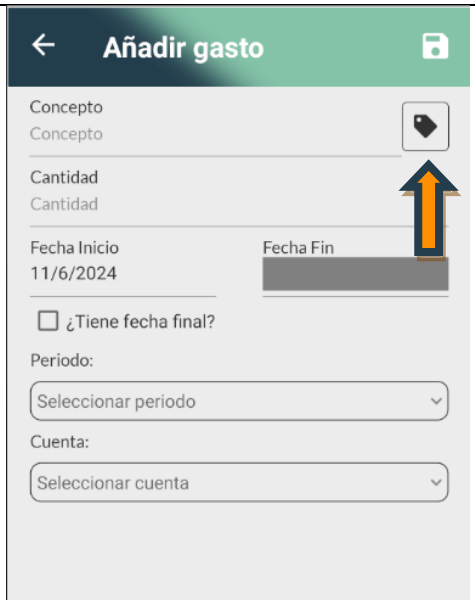
Hacer clic en el botón “Añadir” en la parte superior.



Deberá introducir todos los datos correspondientes. Además, para seleccionar una categoría deberá pulsar sobre el botón que se indica.

Una vez se hayan introducido los datos, deberá clicar sobre el icono  y el gasto será guardado.

En caso de querer cancelar o volver atrás, deberá pulsar sobre el icono  de la esquina superior izquierda.



En la sección de gastos fijos, aparecerán todos los gastos fijos, tanto los activos como los finalizados. Se deberá hacer clic sobre el que se quiera activar o finalizar.

En caso de que se desee editar el gasto fijo se deberán actualizar los datos que corresponda y seleccionar el icono  .

Para activar o finalizar un gasto fijo se deberá pulsar sobre el botón inferior que se indica. Este contendrá la palabra "Finalizar" o "Activar" según corresponda.

Desde la sección de gastos, deberá pulsar el botón “Historial”

Para cambiar las fechas de los gastos que se muestran se deberá pulsar sobre la fecha.

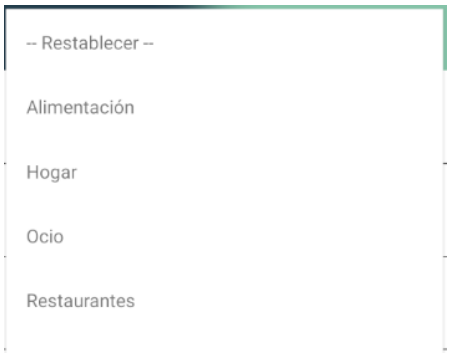
Esto mostrará un recuadro donde podrá elegir el mes y el año.

Una vez seleccionada la fecha se deberá pulsar sobre “Guardar”. Para cancelarlo se deberá hacer clic sobre la “X” de la esquina superior derecha”.

Si necesita buscar un gasto por el concepto o la cuenta con la que se realizó, deberá introducir el nombre en el recuadro que se indica.

Para filtrar por categorías, se deberá pulsar sobre el botón “Filtrar”, y aparecerá un desplegable como el que se muestra.

Deberá elegir la categoría que desee o seleccionar “Restablecer” en caso de que quiera quitar el filtro.



Si necesita ordenar los gastos, deberá pulsar sobre “Ordenar por” y se le mostrarán varias opciones para ordenar. Una vez seleccione uno, los gastos aparecerán ordenados.

EXPORTAR DATOS

Desde el historial de gastos, se deberá seleccionar el icono si lo que se necesita es exportar a Excel.

Para exportar a PDF se deberá hacer clic en el icono .

Ver y/o eliminar gasto

Para ver un gasto deberá hacer clic sobre el que desee. Se le mostrará una pantalla con los detalles del gasto.

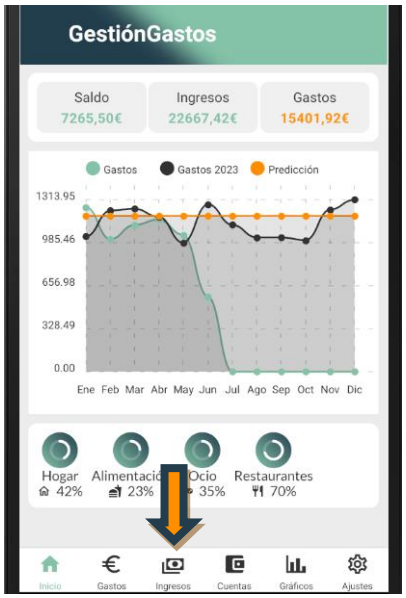
Si lo que quiere es eliminarlo, deberá hacer clic sobre el botón inferior “Eliminar gasto”

En caso de querer cancelar o volver atrás, deberá pulsar sobre el icono de la esquina superior izquierda.

Gestionar ingresos

Añadir ingreso

Deberá hacer clic en el menú horizontal inferior de la aplicación en el botón donde se lee “Ingresos”.



Una vez ahí, hará clic en el botón “Añadir” en la parte superior.

Deberá introducir todos los datos correspondientes. Además, para seleccionar una categoría deberá pulsar sobre el botón que se indica.

Una vez se hayan introducido los datos, el usuario deberá clicar sobre el icono  y el ingreso será guardado.

En caso de querer cancelar o volver atrás, deberá pulsar sobre el icono  de la esquina superior izquierda.

VER HISTORIAL

Desde la sección de gastos, deberá pulsar el botón “Historial”

Para cambiar las fechas de los ingresos que se muestran se deberá pulsar sobre la fecha.

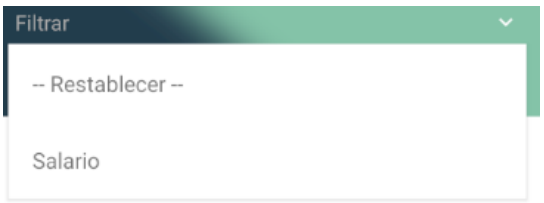
Esto mostrará un recuadro donde podrá elegir el mes y el año.

Una vez seleccionada la fecha se deberá pulsar sobre “Guardar”. Para cancelarlo se deberá hacer clic sobre la “X” de la

esquina superior derecha”.

Para filtrar por categorías, se deberá pulsar sobre el botón “Filtrar”, y aparecerá un desplegable como el que se muestra.

Deberá elegir la categoría que desee o seleccionar “Restablecer” en caso de que quiera quitar el filtro.



Si necesita ordenar los ingresos, deberá pulsar sobre “Ordenar por” y se le mostrarán varias opciones para ordenar. Una vez seleccione uno, los ingresos aparecerán ordenados.

EXPORTAR DATOS

Para exportar los ingresos a Excel, deberá hacer clic sobre el icono .

VER Y/O ELIMINAR INGRESO

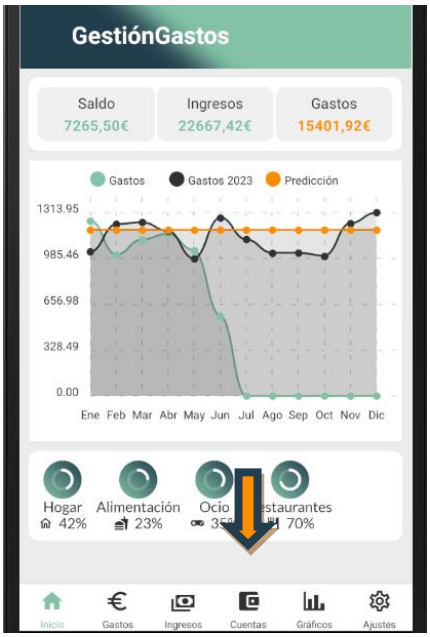
Para ver un ingreso deberá hacer clic sobre el que desee. Se le mostrará una pantalla con los detalles del ingreso.

Si lo que quiere es eliminarlo, deberá hacer clic sobre el botón inferior "Eliminar ingreso"

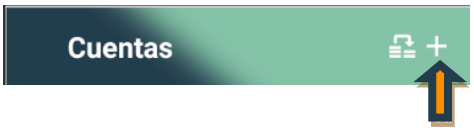
GESTIONAR CUENTAS

AÑADIR CUENTA

Deberá hacer clic en el menú horizontal inferior de la aplicación en el botón donde se lee “Cuenta”.



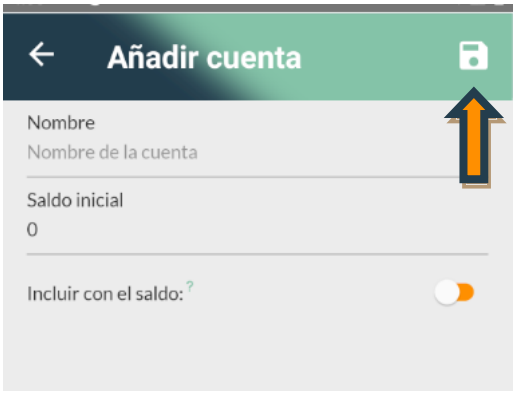
Una vez ahí, hará clic en el icono + en la parte superior.



Deberá introducir todos los datos correspondientes.

Una vez se hayan introducido los datos, deberá clicar sobre el icono y la cuenta será creada.

En caso de querer cancelar o volver atrás, deberá pulsar sobre el icono de la esquina superior izquierda.



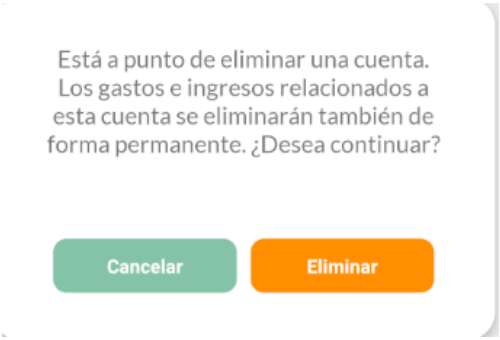
VER LISTADO DE CUENTAS Y ELIMINAR CUENTA

Desde la pantalla principal de la sección de las cuentas, podrá ver todas las cuentas creadas.

Para eliminar una , deberá pulsar sobre el icono de la papelera.

Una vez pulsado el icono, la aparecerá un recuadro. Para eliminar deberá pulsar sobre “Eliminar”.

En caso de no querer eliminar la cuenta, deberá pulsar sobre el botón “Cancelar”.



VER Y EDITAR CUENTA

Para ver o editar una cuenta, hay que clicar sobre la que se desee realizar la acción.

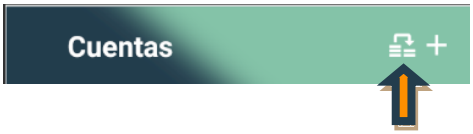
Una vez seleccionada podrá ver los datos de la cuenta, y editarla.

Si se ha editado la cuenta, deberá clicar sobre el icono para guardar los cambios.

En caso de querer cancelar o volver atrás, deberá pulsar sobre el icono de la esquina superior izquierda.

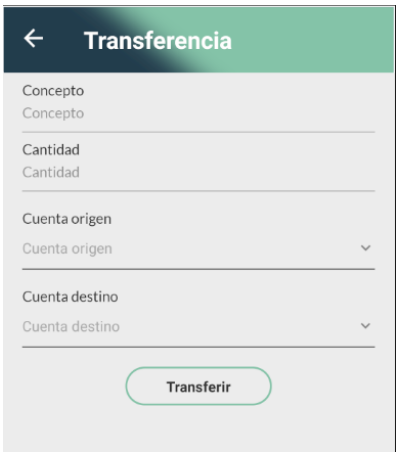
TRANSFERIR DINERO ENTRE CUENTAS

Desde la pantalla principal de la sección de cuentas, se deberá clicar sobre el icono que se indica.



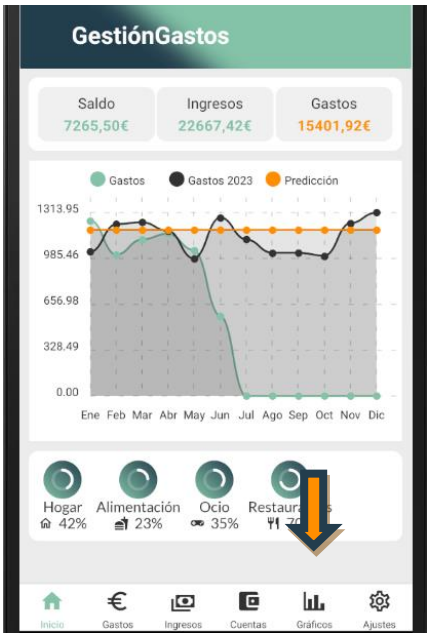
Se deben introducir los datos correspondientes y pulsar sobre el botón “Transferir”.

En caso de querer cancelar o volver atrás, deberá pulsar sobre el icono de la esquina superior izquierda.

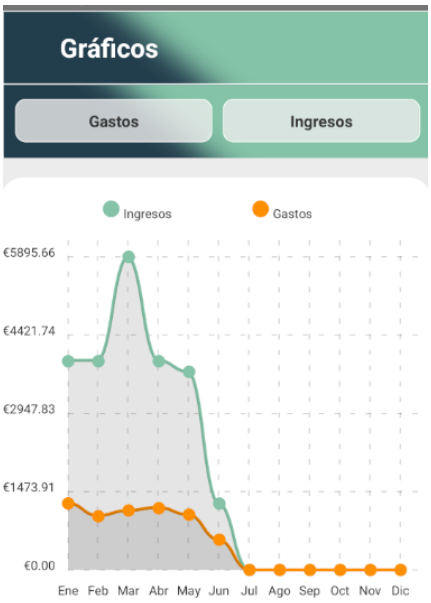


VER GRÁFICOS

Deberá hacer clic en el menú horizontal inferior de la aplicación en el botón donde se lee “Gráficos”.



En la pantalla principal de la sección aparecerá un gráfico con la comparación de los gastos e ingresos del año en curso.



Para ver los gráficos de gastos deberá seleccionar el botón “Gastos” si por el contrario desea ver los de ingresos hará clic sobre el botón de “Ingresos”

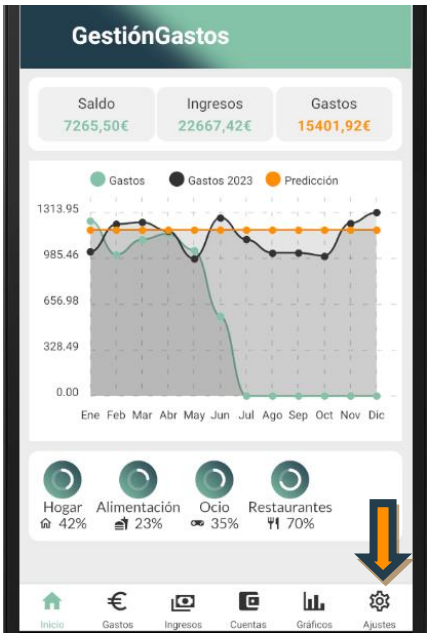
A continuación, aparecerá un listado con los gráficos disponibles. Se deberá seleccionar el que se desee visualizar.

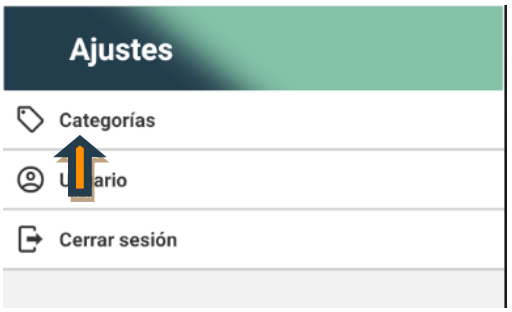
GESTIONAR CATEGORÍAS

AÑADIR CATEGORÍA

Deberá hacer clic en el menú horizontal inferior de la aplicación en el botón donde se lee “Ajustes”.

Una vez ahí debe seleccionar “Categorías”.





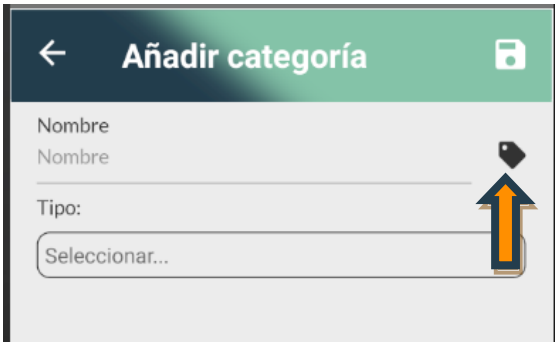
Una vez ahí, hará clic en el icono + en la parte superior.

Deberá introducir todos los datos correspondientes.

Para seleccionar el icono que representará la categoría se deberá hacer clic sobre el elemento que se indica.

Una vez se hayan introducido los datos, el usuario deberá clicar sobre el icono y la categoría será creada.

En caso de querer cancelar o volver atrás, deberá pulsar sobre el icono de la esquina superior izquierda.



VER LISTADO Y ELIMINAR CATEGORÍA

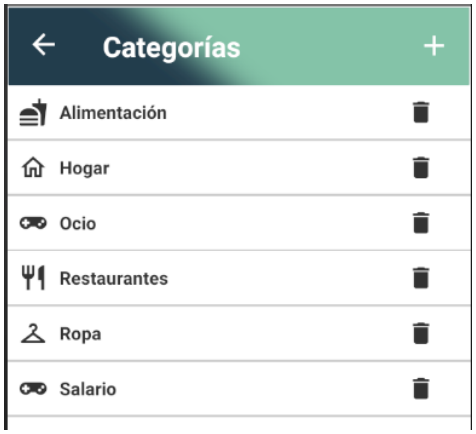
Desde la pantalla principal de la sección de las categorías, podrá ver todas las categorías.

Para eliminar una , deberá pulsar sobre el icono de la papelera.

Una vez pulsado el icono, la aparecerá un recuadro. Para eliminar deberá pulsar sobre “Eliminar”.

En caso de no querer eliminar la categoría, deberá pulsar sobre el botón “Cancelar”.

Para ver o editar una categoría, hay que clicar sobre la que se desee realizar la acción.



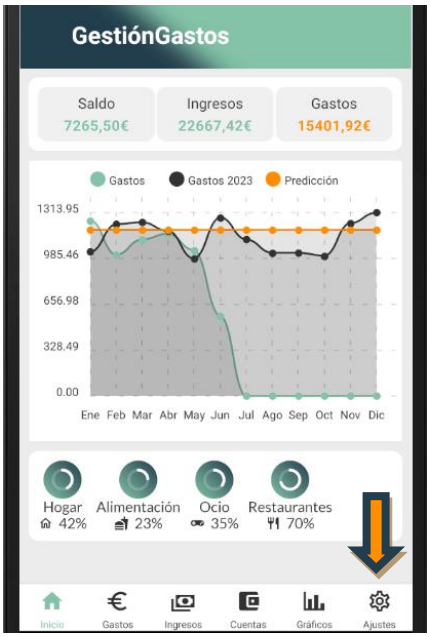
Una vez seleccionada el usuario podrá editarla.

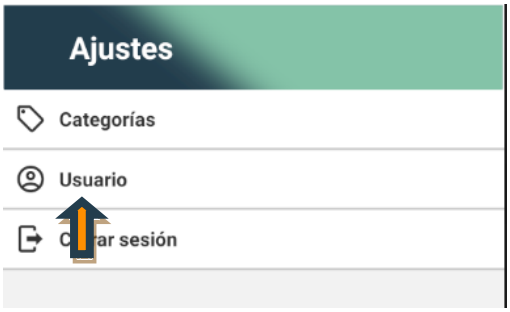
Si se ha editado la categoría, deberá clicar sobre el icono para guardar los cambios.

En caso de querer cancelar o volver atrás, deberá pulsar sobre el icono de la esquina superior izquierda.

Deberá hacer clic en el menú horizontal inferior de la aplicación en el botón donde se lee “Ajustes”.

Una vez ahí debe seleccionar “Usuario”.

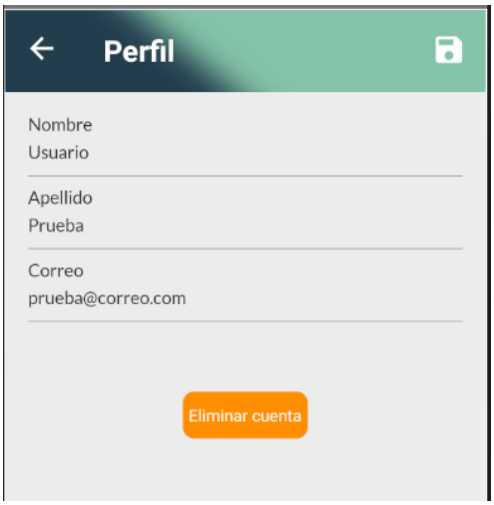




Si se quiere editar los datos, estos deberán ser modificados y a continuación pulsar sobre el icono .

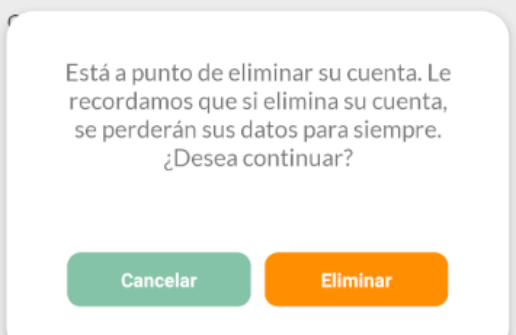
Si lo que se desee es eliminar la cuenta deberá hacer clic sobre el botón “Eliminar cuenta”

Aparecerá un mensaje de confirmación, para eliminar se deberá pulsar sobre el botón “Eliminar”, si por el contrario se desee cancelar, se deberá pulsar sobre



“Cancelar”.

En caso de querer cancelar o volver atrás, deberá pulsar sobre el icono de la esquina superior izquierda.



Repositorios de software

Backend: https://github.com/CristinaFC/GestionGastos

Frontend: https://github.com/CristinaFC/FrontGestionGastos

REFERENCIAS

Alcaraz, M. (21 de 01 de 2021). *billage*. Obtenido de https://www.getbillage.com/es/blog/metodologia-kanban-ventajas-y-caracteristicas#question_1

cesc1989. (Octubre de 2022). *Otro Espacio Blog*. Recuperado el 2024

Developer Mozilla. (s.f.). Recuperado el 8 de Junio de 2023, de https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript#una_definici%C3%B3n_de_alto_nivel

Frankfurter Exchange rates and currency data API. (s.f.). Obtenido de https://www.frankfurter.app/

Goldbergyoni. (s.f.). *GitHub*. Recuperado el 2023, de https://github.com/goldbergyoni/nodebestpractices

Herrera, F. (Noviembre de 2022). Node: De cero a experto.

INE - Instituto Nacional de Estadística. (26 de Febrero de 2024). *Encuesta de Condiciones de Vida (ECV)*. Obtenido de Instituto Nacional de Estadística: https://www.ine.es/dyngs/Prensa/ECV2023.htm

Microsoft. (s.f.). Recuperado el 8 de Junio de 2023, de https://www.microsoft.com/es-es/microsoft-365/excel

Petrova, S. (18 de Enero de 2019). *Adeva*. Recuperado el 28 de Mayo de 2024, de https://adevait.com/blog/remote-work/adopting-agile-the-latest-reports-about-the-popular-mindset

Postman API Platform. (8 de Junio de 2023). Obtenido de https://www.postman.com/product/what-is-postman/

React Native. (s.f.). *React Native*. Recuperado el 2023, de https://reactnative.dev/

Redux. (s.f.). Recuperado el 2023, de https://redux-toolkit.js.org/api/configureStore

Soporte técnico de Microsoft. (s.f.). Recuperado el 8 de Junio de 2023, de https://support.microsoft.com/es-es/office/introducci%C3%B3n-a-microsoft-teams-b98d533f-118e-4bae-bf44-3df2470c2b12

Time Solutions Ltd. (s.f.). *Time Solutions Ltd.* Recuperado el 2024, de https://www.timecamp.com/es/planner/glossary/marco-de-trabajo-scrum/

Visual Studio Code. (3 de Noviembre de 2021). Recuperado el 3 de Junio de 2023, de https://code.visualstudio.com/

Wikipedia. (2 de Mayo de 2024). Recuperado el 27 de Mayo de 2024, de https://es.wikipedia.org/wiki/Node.js

Enlaces externos

1. Fotos de personas ficticias: https://thispersondoesnotexist.com/
2. Generación de escenarios: https://www.storyboardthat.com/es
3. Diseño del mockup: https://mockflow.com/
4. Creación del logo: https://www.canva.com/