ECONOMY APP

Projecte Final del Cicle DAW

Alumne: Vicent Roselló Sanchis
DNI: 20039666-L
Tutor: Guillermo Vidal Frasquet

Dades del Projecte

Dades de l'alumne

Nom i cognoms	Vicent Roselló Sanchis
NIF/NIE	20039666L
Curs i CF	2n DAW

Dades del projecte

Títol del projecte	Eonomy App
Nom del tutor individual	Guillermo Vidal Frasquet
Nom del tutor del grup	Guillermo Vidal Frasquet
Resum	Aplicació per gestionar l'economia familiar, centralitzant ingressos, despeses i estalvis amb balanços en temps real. Funciona en un servidor local amb Docker i garanteix privacitat i control per rols.
Abstract	A family finance management app that centralizes incomes, expenses and savings with real-time balance tracking. Runs locally with Docker, ensuring privacy and role-based access control.
Mòduls implicats	• Sistemes informàtics, Bases de dades • Desenvolupament en entorns del servidor • Disseny d'interfícies • Desplegament d'aplicacions Web • Accés a dades • Programació multimèdia i dispositius mòbils
Data de presentació	16 de desembre de 2025

Índex

1. Introducció/Marc del projecte

1.1. Descripció del projecte

Aquest projecte consisteix en el desenvolupament d'una aplicació destinada a gestionar de manera centralitzada i organitzada l'economia domèstica o familiar. L'objectiu principal és proporcionar una eina senzilla que permeta als usuaris administrar de manera eficient els recursos econòmics del nucli familiar, categoritzant i registrant tant les despeses com els estalvis, i permetent actualitzar-los en temps real.

A més, gràcies a la implementació del registre d'usuaris, la seguretat i els rols, cada integrant de la família pot accedir a la informació de manera personalitzada i segura, garantint la privadesa tant de les seues dades com dels seus moviments econòmics.

L'aplicació es compon de quatre elements principals:

- **Una base de dades MySQL**, dissenyada per emmagatzemar tota la informació rellevant sobre l'economia familiar i els usuaris.
- **Una API desenvolupada amb Spring Boot**, encarregada de gestionar la lògica de negoci i la comunicació entre els diferents components del sistema.
- **Un client mòbil i web desenvolupat amb Flutter**, tecnologia que permet crear aplicacions per a diversos sistemes operatius a partir d'un únic codi, oferint una interfície intuïtiva i accessible per als usuaris finals.
- **El sistema de desplegament amb Docker**, que permet allotjar cadascun dels serveis en contenidors independents i consumir l'aplicació dins de la xarxa local. Aquesta decisió respon a la importància de mantindre la privacitat de les dades personals i financeres.

Per tal de mostrar el funcionament de l'aplicació i facilitar-ne la demostració, també s'ha realitzat un desplegament en un servidor casolà. Mitjançant l'ús de la VPN Tailscale, s'ha habilitat un enllaç segur que permet accedir a l'aplicació des de l'exterior per a finalitats de prova.

Amb aquesta solució es pretén millorar l'eficiència econòmica familiar, facilitant l'organització i la categorització de tots els aspectes financers de la llar, reduint despeses innecessàries i optimitzant els estalvis. Tot això contribueix a disminuir el temps d'administració i a oferir una experiència més còmoda i eficient als usuaris.

1.2. Objectius

1. **Desenvolupar** una aplicació mòbil i web per a la gestió integral de l'economia familiar.
2. **Implementar** una base de dades MySQL per emmagatzemar i gestionar tota la informació relacionada amb els usuaris, despeses i estalvis, així com els objectius plantejats per a cadascun.
3. **Crear** una API amb Spring Boot per gestionar la lògica de negoci i facilitar la comunicació entre la base de dades i els clients mòbil i web.

4. **Assegurar** la privacitat i seguretat de les dades mitjançant la implementació de registres d'usuaris, sistemes de rols i permisos d'accés.

5. **Millorar** l'experiència dels integrants del nucli familiar oferint una interfície intuïtiva i accessible mitjançant Flutter.

6. **Desplegar** l'aplicació utilitzant Docker per a garantir la portabilitat i la facilitat de manteniment en un entorn intern de la llar.

7. **Permetre** als usuaris seguir en temps real el desenvolupament de la gestió, oferint consultes informatives sobre l'estat dels saldos, despeses, ingressos i estalvis.

8. **Facilitar** la presentació del projecte desplegant una versió en un servidor casolà per a demostrar la funcionalitat de l'aplicació en un entorn controlat.

1.3. Tipus de projecte

Tipus de Projecte

Aquest projecte és de tipus intern, ja que està pensat per a ser utilitzat dins del nucli familiar com una ferramenta que garantisca la privacitat de les dades sensibles relacionades amb la situació financera de la llar. L'aplicació pot allotjar-se en qualsevol ordinador de casa i posar-se a disposició dels usuaris a través de la xarxa local.

Tanmateix, gràcies a l'ús de tecnologies com les VPN o l'obertura controlada de ports del router, és possible accedir-hi des de l'exterior, sempre tenint en compte les mesures de seguretat necessàries. Cal remarcar que l'aplicació incorpora un sistema complet d'autenticació i autorització per a protegir l'accés als recursos.

L'objectiu principal és oferir una eina personalitzada per a la gestió econòmica de la llar, adaptada a les necessitats específiques de cada família.

Requeriments del projecte

- **Requeriments de programari:**
  - **Llenguatges i frameworks:**
    - El desenvolupament de l'API es realitza en **Java** utilitzant **Spring Boot** per a la lògica de negoci.
    - El client mòbil i web es desenvolupa en **Dart** utilitzant **Flutter** per permetre la programació multiplataforma.
    - La base de dades s'implementa en **MySQL** per gestionar la informació.

- **Entorn de desenvolupament:**
  - Per al desenvolupament i creació de la base de dades, s'utilitza **MySQL Workbench**.
  - **Visual Studio Code** es fa servir per a desenvolupar l'API, formant conjuntament amb MySQL Workbench la tecnologia backend. Tanmateix també s'utilitza per al desenvolupament del frontend amb el framework de **Flutter**.
  - Es necessita **Android Studio** per a gestionar els SDK d'Android i emular dispositius mòbils.
  - Per a realitzar proves a l'API, s'utilitza **Swagger**, una potent llibreria de **Spring** que proporciona una interfície gràfica per realitzar peticions a l'API, sent una alternativa a altres eines com Postman.

- **Requeriments d'infraestructura:**
  - **Infraestructura interna:**

    El projecte es desplega en qualsevol ordinador domèstic, que actua com a servidor intern per posar a disposició dels usuaris totes les funcionalitats de l'aplicació a través de la xarxa local. Mitjançant Docker, es creen i gestionen els contenidors que allotgen els diferents serveis del sistema: la base de dades, l'API desenvolupada amb Spring Boot i el client web/mòbil generat amb Flutter.
  - **Xarxa:**

    Per a protegir les dades personals i econòmiques dels usuaris, l'aplicació funciona dins d'una xarxa local privada, evitant l'exposició directa a internet. Aquest enfocament garanteix la seguretat i confidencialitat de la informació, ja que només els dispositius connectats a la xarxa interna poden accedir al servei.
  - **Accés remot per a demostració:**

    Per tal de permetre la presentació i comprovació del funcionament de l'aplicació fora de l'entorn local, s'ha optat per un desplegament en un servidor casolà combinat amb l'ús de la VPN Tailscale. Aquesta solució crea un enllaç segur que permet accedir a l'aplicació des de l'exterior de manera controlada, mantenint sempre la protecció de les dades i fent ús de dades fictícies quan siga necessari.

- **Requeriments de migració:** No aplicable, ja que es tracta d'una implementació nova sense dades prèvies a migrar.

1.4. Orientacions per al desenvolupament i recursos

Durant el desenvolupament d'aquest projecte, s'ha seguit una metodologia en cascada més coneguda coma waterfall, per a cada fase del projecte. Les fases s'han dividit de la següent forma: construcció de la base de dades, desenvolupament de la api, desenvolupament del client i finalment el desplegament.

Com a bones pràctiques, he fet ús de patrons de disseny per a una estructura de codi clara i mantenible, com ara be MVC i la gestió d'usuaris i permisos amb estàndards de seguretat actuals.

Les eines i tecnologies utilitzades inclouen:

- **Git** per al control de versions, facilitant la col·laboració i mantenint un historial de canvis detallat.
- **Swagger** per documentar i provar l'API de manera eficaç.
- **Docker** per crear entorns de desenvolupament i producció consistents i aïllats.

Principal recursos:

- Temari implicat al cicles de DAM i DAW.

Recursos web:

- **Documentació oficial de Spring Boot:** Ha estat la guia principal per a la configuració i desenvolupament de l'API.
- **Flutter.dev:** La documentació oficial de Flutter, usada per desenvolupar la interfície d'usuari tant per a la web com per a dispositius mòbils.
- **Eïnes de IA:** Han estat recursos valuosos per resoldre dubtes específics durant el desenvolupament.
- “Api Rest con Srping Boot” de l’acadèmia Openwebinaris, Un curs que ha estat essencial per dominar les bases del framework Spring.

2. Anàlisi/Estudi de l'estat actual

2.1. Descripció del sistema actual

Actualment, la gestió econòmica de moltes llars es realitza de manera manual i desorganitzada. Cada família adquireix els seus propis hàbits i mètodes, que sovint no garanteixen un control eficient ni una visió clara de la situació financera. Els escenaris més habituals són els següents:

1. **Absència de planificació econòmica:**
   En moltes cases no existeix cap sistema formal per registrar despeses, ingressos o estalvis. Les decisions econòmiques es prenen segons les necessitats del moment, sense una visió global del pressupost mensual o anual. Això pot provocar desajustos, imprevistos i la sensació constant de no saber en què s’estan gastant els diners.

2. **Registre manual en papers o llibretetes:**
   Algunes famílies opten per anotar despeses i compres en fulls solts o llibretes. Aquest mètode és poc pràctic: els papers es poden perdre, no permeten fer càlculs automàtics ni mantenen un històric clar, i requereixen un esforç constant per actualitzar-los. A més, la revisió manual incrementa el risc d’errors i duplicitats.

3. **Ús de fulls de càlcul (Excel, Google Sheets):**
   Una altra opció habitual és utilitzar taules de càlcul creades pels propis usuaris. Tot i ser una millora respecte al paper, aquest sistema presenta limitacions importants: falta de seguretat, dificultat per compartir el document entre diversos membres de la família, necessitat de coneixements per mantindre les fórmules i absència d'una estructura clara per categoritzar despeses i estalvis. A més, no permet un accés còmode des del mòbil ni ofereix actualització en temps real.

4. **Aplicacions de tercers:**
   Encara que existeixen aplicacions mòbils i serveis en línia per a la gestió econòmica, moltes famílies són reticents a utilitzar-los a causa de la privacitat: impliquen pujar dades econòmiques personals a servidors externs o empreses privades. A més, sovint inclouen funcionalitats de pagament o publicitat, i no sempre s'adapten a les necessitats concretes de cada llar.

2.2. Limitacions del sistema actual

Aquest conjunt de mètodes presenta diverses limitacions que dificulten una gestió econòmica clara i eficient:

- **Manca d'eficiència:**
  La gestió manual o mitjançant eines disperses requereix temps i esforç, especialment quan es vol tindre un control actualitzat dels comptes familiars.
- **Alta probabilitat d'errors humans:**
  Les anotacions manuals i els fulls de càlcul són propensos a confusions, fórmules incorrectes, pèrdua d'informació o registres duplicats.
- **Manca d'organització:**
  No hi ha una estructura clara per categoritzar les despeses, gestionar estalvis o visualitzar el balanç econòmic en temps real.
- **Poca seguretat i privacitat:**
  Tant els papers físics com els documents compartits en el núvol manquen de mecanismes de seguretat robustos. Aquests sistemes tampoc controlen qui pot editar o veure les dades.
- **Poca accessibilitat entre membres de la família:**
  Compartir informació entre diversos integrants és complicat i no existeix una plataforma única on consultar l'estat econòmic global.

2.3. Viabilitat del sistema actual

Després d'analitzar el sistema utilitzat habitualment en les llars, es conclou que aquest **no és viable** per a assolir una gestió eficient, segura i centralitzada de l'economia familiar. Tot i que permet, en major o menor mesura, portar un seguiment bàsic de les despeses, presenta nombroses mancances:

- **Escalabilitat limitada:**
  Quan augmenta el nombre de despeses, categories, membres o objectius d'estalvi, el sistema manual es torna confús i difícil de mantindre.
- **Dependència d'hàbits individuals:**
  El control econòmic es dispersa entre qui anota què, i sovint es produeixen omissions o inconsistències.
- **Falta de seguretat:**
  No existeix cap mecanisme d'autenticació ni control d'accés, deixant les dades exposades o vulnerables.

A causa d'aquestes limitacions, el sistema actual **no pot garantir un control econòmic eficient, segur ni còmode**. Per tant, es justifica la necessitat d'una aplicació digital centralitzada que automatitze processos, emmagatzeme les dades de manera segura i facilite la col·laboració entre tots els membres de la família.

2.4. Requeriments del nou sistema

Per a superar les limitacions del sistema actual i assolir els objectius plantejats, el nou sistema ha de complir amb els següents requeriments:

2.4.1. Requeriments funcionals:

El nou sistema de gestió econòmica familiar ha de complir una sèrie de requeriments funcionals i tècnics amb l'objectiu d'oferir una eina completa, segura i fàcil d'utilitzar per a tots els membres de la família. A continuació, es detallen els requeriments principals:

Categorització dels moviments econòmics
El sistema ha de permetre crear, editar i eliminar categories i subcategories per classificar totes les despeses i moviments econòmics. L'objectiu és identificar clarament en què es gasten els diners i facilitar l'anàlisi de la despesa mensual i anual.

Registre d'ingressos
Cada membre podrà registrar els seus ingressos de forma senzilla, indicant data, descripció i quantitat. Això permetrà calcular el balanç real disponible en cada període.

Registre i gestió d'usuaris
El sistema ha d'incloure un mecanisme per al registre d'usuaris, amb accés personalitzat segons rols (administrador o usuari estàndard). Cada usuari pot disposar d'un perfil propi amb informació bàsica.

Administració dels estalvis
L'aplicació ha de permetre gestionar els estalvis familiars mitjançant categories específiques, transferències i moviments designats a fons d'estalvi.

Sistema d'objectius econòmics
Es podrà definir objectius tant de despesa com d'estalvi (per exemple: "estalviar 500 € per a vacances" o "no superar 200 € en restaurants"). El sistema ha de mostrar el progrés de cada objectiu.

Actualització en temps real del balanç
El sistema ha de calcular automàticament i en temps real:

- el balanç mensual i anual,
- el saldo disponible per a gastar,
- el total de diners acumulats,
- el total estalviat i el total gastat,
- i la disponibilitat restant després de restar despeses i aportacions als estalvis.

Visualització de l'activitat econòmica
Els usuaris podran consultar l'activitat econòmica filtrada per mensualitats, totals o anuals. Això facilita la comparació entre períodes i el seguiment històric.

Accés segur amb rols i autenticació
El sistema ha d'implementar mecanismes d'autenticació i autorització basats en JWT o tecnologies equivalents. Segons el rol assignat, els usuaris podran tindre diferent nivell d'accés i permisos sobre les dades.

Interfície d'usuari intuïtiva
El client web/mòbil ha d'oferir una interfície clara, accessible i fàcil d'utilitzar. L'objectiu és que qualsevol membre de la família, independentment de la seua experiència tecnològica, puga navegar i utilitzar l'aplicació sense dificultats.

Perfil d'usuari amb fotografia opcional
Els usuaris podran carregar una fotografia al seu perfil, que quedarà emmagatzemada al servidor. Aquesta funcionalitat millora la identificació i fa l'experiència més personal.

Conversió automàtica de moviments en dades agregades
El sistema ha d'actualitzar automàticament els següents valors en funció dels moviments econòmics registrats:

- saldo disponible,
- total d'ingressos,
- total de despeses,
- total aportat a estalvis,
- evolució anual, mensual o total.
  Aquest procés s'ha de realitzar en temps real, sense necessitat d'intervenció manual.

Generació i descàrrega d'un resum mensual en PDF
L'aplicació ha de permetre exportar un resum mensual complet en format PDF, incloent:

- el balanç general,
- el total gastat i estalviat,
- els ingressos del període,
- un resum per categories i subcategories,
- i qualsevol objectiu econòmic relacionat. Aquest document és útil per a consultes, seguiment i arxiu familiar.

2.4.2. Requeriments no funcionals:

Els requeriments no funcionals defineixen les característiques de qualitat que ha de complir el sistema per tal de garantir un funcionament eficient, segur, escalable i accessible per a tots els membres de la família. Aquests requeriments descriuen com s'ha de comportar l'aplicació i quines propietats ha de complir.

Seguretat

- El sistema ha de garantir la protecció de les dades personals i econòmiques dels usuaris.
- S'utilitzarà autenticació basada en JSON Web Tokens (JWT) per validar les sessions d'usuari i controlar l'accés als recursos.
- Les contrasenyes s'han d'emmagatzemar de manera encriptada, utilitzant algoritmes segurs com BCrypt.
- S'ha de limitar l'accés a la base de dades per mitjà de rols i permisos, de manera que només els usuaris autoritzats puguen realitzar accions crítiques.
- El servidor casolà no exposarà directament els ports a internet; l'accés remot es farà únicament mitjançant la VPN Tailscale, garantint una connexió xifrada.

Rendiment

- El sistema ha de ser capaç de respondre a peticions dels usuaris amb un temps de resposta òptim, evitant retards perceptibles.
- La càrrega de dades (moviments, categories, estalvis, etc.) ha de ser fluida tant en el client web com en el client mòbil.
- Els càlculs en temps real (balanç, totals mensuals, anuals i globals) han d'executar-se de manera immediata sense afectar el rendiment general.

Fiabilitat i disponibilitat

- El sistema ha d'estar disponible en tot moment dins de la xarxa local, a menys que el servidor estiga apagant-se o en manteniment.
- Docker ha de garantir que els serveis es mantenen operatius i es reinicien automàticament en cas de fallada.
- S'ha d'assegurar l'estabilitat de l'API i de la base de dades per evitar pèrdues de dades o interrupcions inesperades.

Usabilitat

- La interfície d'usuari ha de seguir un disseny clar, accessible i fàcil d'entendre per a qualsevol membre de la família, independentment del seu nivell tecnològic.
- La navegació ha de ser intuïtiva, amb etiquetes clares i estructures visuals coherents.
- L'aplicació ha de ser utilitzable tant en dispositius mòbils com en ordinadors, mantenint una experiència consistent.

Portabilitat

- L'aplicació (API i base de dades) ha de poder executar-se en qualsevol sistema operatiu compatible amb Docker, com ara Linux, Windows o macOS.
- El client Flutter ha de ser accessible des de mòbil i navegador web sense necessitat d'adaptacions específiques per plataforma.
- La configuració del sistema ha de ser fàcilment replicable per mitjà de fitxers *`docker-compose.yml`*.

Escalabilitat

- El sistema ha de permetre afegir nous usuaris, categories, moviments i funcionalitats sense que això afecte el rendiment.
- L'arquitectura basada en API i microserveis contenitzats facilita la possibilitat d'ampliar el projecte en el futur (p. ex.: gràfics avançats, anàlisi d'hàbits de consum, o nous mòduls d'estalvi).

Mantenibilitat

- El codi ha d'estar organitzat segons bones pràctiques de desenvolupament, facilitant la seua lectura i ampliació.
- L'ús d'un backend en Spring Boot i un frontend en Flutter permet mantindre una estructura clara i modular.
- S'han d'incloure comentaris i documentació mínima en els punts crítics del codi per facilitar futures modificacions.
- Docker permet actualitzar o modificar els serveis de manera independent.

Privacitat

- Les dades econòmiques i personals només han de ser accessibles per als membres de la família registrats.
- No es farà ús de serveis externs per emmagatzemar informació, mantenint totes les dades dins del servidor casolà.
- L'accés remot a través de Tailscale garanteix un túnel xifrat i segur sense exposar informació sensible.

Compatibilitat

- El sistema ha de funcionar en diferents navegadors moderns (Chrome, Firefox, Edge...).
- L'aplicació mòbil ha de ser compatible amb Android i, si es desitja en el futur, amb iOS sense grans canvis de codi.
- L'API ha de permetre integrar futurs mòduls o serveis externs si la família decideix ampliar les funcionalitats.

3. Anàlisi de la solució

3.1. Anàlisi de les possibles solucions

Abans de desenvolupar el sistema, s'han considerat diverses alternatives que podrien resoldre la necessitat de portar un control organitzat de l'economia familiar. A continuació, es presenten les principals opcions analitzades:

Solució 1: Desenvolupament des de zero amb Spring Boot i Flutter (Solució adoptada)

Característiques:
Creació d'una aplicació pròpia utilitzant Spring Boot per al backend i Flutter per al client web i mòbil. Aquesta alternativa permet dissenyar una solució totalment personalitzada, adaptada a les necessitats específiques de la família i amb un control complet sobre totes les dades i funcionalitats.

Avantatges:

- Flexibilitat total en el disseny, funcionalitat i estructura de dades.
- Possibilitat de crear interfícies adaptades tant per a navegadors web com per a dispositius mòbils.
- Privacitat i control absolut de les dades, emmagatzemades en una base de dades pròpia dins del servidor casolà.
- Possibilitat d'ampliació futura segons les necessitats de la família (objectius, gràfics avançats, estadístiques, etc.).

Limitacions:

- Major temps i esforç de desenvolupament en comparació amb altres opcions.
- Requereix coneixements tècnics i manteniment continu.

Solució 2: Ús d'aplicacions de tercers

Característiques:

Existixen múltiples aplicacions de gestió econòmica disponibles al mercat (Fintonic, Monefy, Wallet, etc.), les quals permeten registrar despeses, ingressos i fer seguiment del pressupost familiar.

Avantatges:

- Implementació immediata, sense necessitat de desenvolupament.
- Interfícies ja dissenyades i funcionals.
- Suport i actualitzacions ofertes per empreses externes.

Limitacions:

- Falta de privadesa: les dades econòmiques es guarden en servidors externs.
- Funcionalitats limitades o de pagament en moltes aplicacions.
- Dificultat per adaptar-se a les necessitats concretes de cada família.
- Dependència de tercers i de polítiques de dades que no es poden controlar.

A causa d'aquestes limitacions, esta opció no s'ha considerat adequada per al projecte.

Solució 3: Contractar una gestoria o assessorament extern

Característiques:

Consultar una gestoria o un professional extern per portar el control econòmic familiar, fer seguiment d'ingressos, despeses i estalvis.

Avantatges:

- Professionalització de la gestió.
- Possibilitat de rebre consells i informes econòmics personalitzats.
- Delegació total de la tasca de control econòmic.

Limitacions:

- Cost econòmic elevat i recurrent.
- No ofereix visualització en temps real del balanç familiar.
- Dependència constant d'un professional extern.
- No resol la necessitat d'una eina pròpia per a consultar dades en qualsevol moment.

Aquesta solució tampoc resulta viable per al dia a dia d'una família, especialment si l'objectiu és tindre una eina pròpia i accessible sense costos addicionals.

Comparativa basada en els criteris:

Solució	Cost Inicial	Flexi-bilitat	Temps de Desenvo-lupament	Escala-bilitat	Facilitat de Man-teniment
Desenvolupament des de zero	Baix	Alt	Llarg	Alt	Alt
Ús d'aplicacions de tercers	Baix	Mitjà	Curt	Mitjà	Mitjà
Contractar una gestoria	Alt	Alt	Molt curt	Baix	Alt

3.2. Avaluació de les possibles solucions

Després d'avaluar les diferents alternatives, s'ha determinat que la solució més adequada és el **desenvolupament des de zero d'una aplicació pròpia amb Spring Boot i Flutter**. Aquesta opció garanteix:

- màxima privacitat,
- flexibilitat,
- adaptació total a les necessitats familiars,
- possibilitat de creixement futur,
- i independència de tercers.

A més, permet integrar l'aplicació dins d'un entorn controlat i segur en un servidor casolà, reforçat amb Tailscale per a accés remot, mantenint totes les dades sota control exclusiu de la família.

3.3. Descripció de la solució escollida

La solució escollida és el **desenvolupament des de zero d'una aplicació multiplataforma**, utilitzant **Spring Boot** per al backend i **Flutter** per al frontend tant web com mòbil. Aquesta arquitectura permet crear una interfície d'usuari intuïtiva, moderna i coherent en qualsevol dispositiu, aprofitant un únic codi base i reduint l'esforç de manteniment.

El backend, desenvolupat en Spring Boot, és l'encarregat de gestionar tota la **lògica de negoci** relacionada amb la gestió econòmica familiar: registre i validació d'usuaris, control de rols, gestió de despeses, ingressos, estalvis, objectius i càlculs en temps real del balanç econòmic. També incorpora els mecanismes d'**autenticació i autorització**, garantint que només els membres autoritzats puguen accedir a les dades familiars.

La informació econòmica s'emmagatzema en una base de dades **MySQL**, escalable i segura, que permet estructurar de manera eficient categories, moviments i registres històrics. L'ús de **Docker** facilita la creació d'un entorn de desenvolupament i producció consistent, contenint el backend, la base de dades i qualsevol altre servei necessari en contenidors independents.

Aquesta solució permet allotjar tota l'aplicació en un **servidor casolà**, mantenint les dades dins de la xarxa local per garantir la privacitat familiar. Per a l'accés remot i les demostracions, s'utilitza la VPN **Tailscale**, que proporciona un enllaç segur sense exposar els serveis directament a internet.

Amb aquesta arquitectura, s'aconsegueix una solució flexible, segura i plenament adaptada a les necessitats de la gestió econòmica familiar, a més d'assegurar la possibilitat de futures ampliacions i millores sense dependre de serveis externs ni de tercers.

4. Disseny de la solució.

4.1. Requisits.

Un cop definits els objectius del projecte i les necessitats del sistema, resulta necessari especificar amb claredat quines tasques haurà de realitzar l'API, ja que serà el component encarregat de gestionar tota la lògica del programari. L'API actuarà com a punt central de comunicació entre el client (web o mòbil), la base de dades i els diferents serveis de l'aplicació.

A continuació, es detallen les tasques principals que haurà d'implementar la nostra API:

1. Gestió d'usuaris

- Registrar nous usuaris al sistema.
- Validar credencials i generar tokens d'autenticació (JWT).
- Gestionar rols i permisos (administrador / usuari estàndard).
- Permetre l'edició del perfil, incloent la possibilitat de pujar fotografia.

2. Gestió de categories i subcategories

- Crear, editar i eliminar categories de despeses, ingressos i estalvis.
- Associar moviments econòmics a categories concretes.
- Mantenir una estructura clara i reutilitzable per a tota la família.

3. Registre de moviments econòmics

- Registrar ingressos, despeses i aportacions als estalvis.
- Permetre la modificació i eliminació de moviments.
- Validar que tots els moviments siguen coherents amb les regles del sistema.

4. Gestió dels estalvis

- Crear fons d'estalvi i registrar-hi moviments.
- Controlar aportacions i retirades d'estalvis.
- Relacionar objectius d'estalvi amb categories concretes.

5. Sistema d'objectius

- Crear objectius econòmics (d'estalvi o despesa).
- Registrar el progrés automàtic segons els moviments mensuals.
- Alertar o marcar quan un objectiu s'ha aconseguit o superat.

6. Càlcul del balanç en temps real

- Calcular automàticament el balanç mensual, anual i total.
- Determinar saldo disponible, total d'ingressos, total de despeses i total d'estalvis.
- Proporcionar dades agregades de manera instantània al frontend.

7. Consultes i filtratge de dades

- Permetre consultar els moviments filtrats per data, categoria, usuari o tipus.
- Mostrar l'activitat econòmica mensual, anual o global.
- Oferir resums per categories i subcategories.

8. Generació de documents

- Generar informes mensuals en PDF amb el balanç complet.
- Incloure totals per categoria i subcategoria.
- Permetre la descàrrega del document des del client.

9. Integració amb el sistema de desplegament

- Proporcionar configuracions compatibles amb Docker per a facilitar el desplegament.
- Gestionar connexions eficients amb la base de dades MySQL.
- Permetre que el sistema funcione en entorns locals i remots (via Tailscale).

4.2. Anàlisi de riscos.

Durant el desenvolupament i implementació del sistema de gestió econòmica, s'han identificat els següents riscos potencials:

Riscos Tècnics

1. **Errors en la integració entre API i frontend.**
    a. **Probabilitat:** Mitjana
    b. **Impacte:** Alt
    c. **Mesures preventives:** Realitzar proves freqüents d'integració des del principi del desenvolupament. Utilitzar eines com Postman o Swagger per verificar les respostes de l'API.
    d. **Mesures correctives:** Si es detecten errors, implementar un sistema de logs detallats per identificar i corregir els problemes ràpidament.

2. **Problemes de rendiment sota càrregues altes.**
    a. **Probabilitat:** Baixa
    b. **Impacte:** Alt
    c. **Mesures preventives:** Realitzar tests de càrrega i rendiment utilitzant eines com JMeter. Optimitzar consultes SQL i assegurar una configuració adequada del servidor.
    d. **Mesures correctives:** Escalar l'arquitectura mitjançant contenedors addicionals amb Docker en moments de càrrega alta.

Riscos Organitzatius

3. **Retards en el desenvolupament per falta de temps o recursos.**
    a. **Probabilitat:** Mitjana
    b. **Impacte:** Mitjà
    c. **Mesures preventives:** Planificar amb detall cada sprint, establir prioritats clares i reservar temps per a tasques crítiques.
    d. **Mesures correctives:** Redistribuir tasques o simplificar funcionalitats menys prioritàries.

Riscos Legals i de Seguretat

5. **Accés no autoritzat a dades sensibles.**
    a. **Probabilitat:** Baixa
    b. **Impacte:** Alt
    c. **Mesures preventives:** Implementar autenticació robusta amb JWT i encriptació de dades sensibles.
    d. **Mesures correctives:** Si es detecta un accés no autoritzat, revocar immediatament les credencials afectades i auditar l'activitat recent del sistema.

Risc	Probabilitat	Impacte	Prioritat
Errors en la integració entre API i frontend	Mitjana	Alt	Alta
Problemes de rendiment	Baixa	Alt	Mitjana
Retards en el desenvolupament	Mitjana	Mitjà	Mitjana
Accés no autoritzat a dades sensibles	Baixa	Alt	Alta

4.3. Casos d'us.

Per detallar els casos d'ús del sistema, s'identifiquen els diferents actors externs que interactuaran amb l'aplicació i les funcionalitats que utilitzaran. A través dels diagrames UML de casos d'ús, es pot visualitzar de manera clara la relació entre els actors i les accions principals que ofereix el sistema de gestió econòmica familiar.

Actors del sistema

En el sistema es defineixen principalment dos actors:

1. Administrador

Representa l'usuari amb permisos elevats. Normalment és un membre de la família responsable de gestionar les configuracions generals del sistema.

Pot realitzar les següents accions:

- Gestió d'usuaris com assignar rols.
- Crear, editar o eliminar categories i subcategories.
- Gestionar moviments econòmics propis i d'altres usuaris.
- Administrar objectius i estalvis familiars.
- Accedir a totes les estadístiques i informes.

2. Usuari

Representa qualsevol membre de la família que empra l'aplicació per registrar i consultar la seua informació econòmica.

Pot realitzar les següents accions:

- Autenticar-se al sistema.
- Consultar el seu balanç econòmic mensual, anual o total.
- Visualitzar categories, subcategories i objectius globals.
- Pujar i modificar la seua fotografia de perfil.
- Generar i descarregar informes mensuals en PDF.

Casos d'ús principals del sistema

A continuació es detallen els casos d'ús que formen part de la funcionalitat central del sistema. Són els que apareixeran posteriorment representats al diagrama UML.

CU1 – Autenticació d'usuari

Actor: Usuari / Administrador
Descripció: L'usuari introdueix el seu nom d'usuari i contrasenya. El sistema valida les credencials i genera un token JWT per a la sessió.
Objectiu: Accedir de forma segura al sistema.

CU2 – Gestió d'usuaris (només Administrador)

Actor: Administrador
Descripció: L'administrador pot crear nous usuaris, editar-los, assignar rols i eliminar-los.
Objectiu: Controlar qui pot accedir al sistema i amb quins permisos.

CU3 – Editar perfil d'usuari

Actor: Usuari / Administrador
Descripció: L'usuari pot modificar les seues dades personals i pujar una fotografia al servidor.
Objectiu: Personalitzar la seua experiència dins l'aplicació.

CU4 – Gestió de categories i subcategories

Actor: Administrador
Descripció: Crear, editar o eliminar categories de despeses, ingressos i estalvis.
Objectiu: Organitzar correctament els moviments econòmics.

CU5 – Registrar ingressos

Actor: Administrador
Descripció: L'usuari registra un nou ingrés indicant quantitat, descripció, data i categoria.
Objectiu: Actualitzar el balanç econòmic de manera immediata.

CU6 – Registrar despeses

Actor: Administrador
Descripció: L'usuari crea una nova despesa amb la informació necessària.
Objectiu: Mantindre el control precís dels gastos familiars.

CU7 – Gestió d'estalvis

Actor: Administrador
Descripció: Crear moviments d'estalvi, transferir diners entre categories o retirar quantitats.
Objectiu: Portar un registre clar i actualitzat dels fons d'estalvi familiars.

CU8 – Sistema d'objectius
Actor: Administrador
Descripció: Crear objectius d'estalvi o límits de despesa i consultar-ne el progrés.
Objectiu: Facilitar la planificació econòmica de la família.

CU9 – Consulta del balanç i estadístiques
Actor: Usuari / Administrador
Descripció: Visualitzar el balanç mensual, anual o total, així com els totals d'ingressos, despeses i estalvis.
Objectiu: Disposar de dades sempre actualitzades en temps real.

CU10 – Consulta d'activitat econòmica
Actor: Usuari / Administrador
Descripció: Filtrar moviments per data, categoria, usuari o tipus.
Objectiu: Facilitar l'anàlisi del comportament econòmic familiar.

CU11 – Generació de resum mensual en PDF
Actor: Usuari / Administrador
Descripció: El sistema genera un PDF amb el balanç complet del mes i els totals per categoria i subcategoria.
Objectiu: Conservar informes i facilitar la revisió periòdica de l'estat econòmic.

CU12 – Accés remot segur
Actor: Usuari / Administrador
Descripció: Permetre l'accés a l'aplicació des de fora de la xarxa local mitjançant Tailscale.
Objectiu: Accedir al sistema de manera segura des de qualsevol lloc.

Resum del diagrama de casos d'ús
El diagrama de casos d'ús representarà gràficament la relació entre:

- Actors → *Usuari* i *Administrador*
- Funcionalitats principals → *autenticació, moviments, categories, estalvis, objectius, informes, configuració*, etc.

El resultat és una visió clara de tots els punts d'interacció entre els membres de la família i el sistema.

5. Desenvolupament de la solució

Ara ens centrem ja en la part grossa del projecte, el desenvolupament de la solució. Una vegada tenim clar el què volem, anem a detallar les diferents fases i què anem fent en cadascuna d'elles.

5.1. Estimació de cost temporal

Fase	Tasques	Temps Estimat
Anàlisi i Disseny	Disseny del sistema, ERD i casos d'ús	1 setmana
Base de Dades	Creació i configuració de MySQL	2 setmana
Desenvolupament API	Implementació del backend amb Spring	3 setmanes
Desenvolupament Front	Disseny i programació amb Flutter	5 setmanes
Proves i Optimització	Correcció d'errors	2 setmanes
Desplegament	Configuració de Docker i AWS	1 setmana

Durada total estimada: 14 setmanes.

5.2. Dotació de recursos

Els recursos emprats per al desenvolupament del projecte han sigut els següents:

Recursos humans

- **1 desenvolupador**, responsable de totes les fases del projecte: anàlisi, disseny, desenvolupament, proves, desplegament i documentació.

Recursos tècnics

Per a portar a terme el desenvolupament, s'ha utilitzat un ordinador personal amb les eines següents:

- **MySQL Workbench** per al disseny i gestió de la base de dades.
- **Spring Tool Suite 4** i **Visual Studio Code** per al desenvolupament del backend en Java amb Spring Boot.
- **Android Studio** i **Visual Studio Code** per al desenvolupament del frontend amb Flutter.
- **Docker** per al desplegament i contenització de tots els serveis del sistema.
- **Tailscale** per permetre l'accés remot segur al servidor casolà sense exposar ports a internet.
- **Swagger** i **Postman** per a la documentació i les proves de l'API — validació d'endpoints, comprovació de respostes i depuració del comportament del backend.

Aquest conjunt d'eines ha permés garantir un desenvolupament eficient i un entorn de proves completament controlat.

5.3. Configuració i/o desenvolupament del sistema

Aquest apartat descriu el procés de configuració i desenvolupament del sistema, detallant les tecnologies utilitzades, l'organització del projecte i els mecanismes que asseguren el seu correcte funcionament.

Base de dades

- Base de dades implementada en **MySQL**, allotjada en un contenidor Docker.
- Model de dades dissenyat mitjançant diagrama ERD amb taules per a usuaris, rols, categories, despeses i ingresos.
- Scripts SQL per a la creació, inicialització i càrrega de dades.
- **Relacions normalitzades per garantir coherència i evitar redundància.**

Backend

- Desenvolupat en **Java** utilitzant el framework **Spring Boot**.
- Implementació d'una **API REST** per gestionar totes les funcionalitats del sistema: usuaris, moviments, categories, estalvis, objectius, càlculs del balanç, etc.
- Integració de **Spring Security** amb autenticació mitjançant **JSON Web Tokens (JWT)** per controlar l'accés als recursos.
- Documentació de tots els endpoints generada automàticament amb **Swagger/OpenAPI**.
- Estructura modular del codi, separant clarament entitats, serveis, repositoris i controladors.

Frontend

- Desenvolupat amb **Flutter**, permetent una **interfície multiplataforma** accessible tant des de dispositius mòbils com des de navegadors web.
- Ús de components modulars i reutilitzables que faciliten el manteniment i l'escalabilitat.
- Comunicació amb el backend mitjançant peticions HTTP gestionades amb **Dio**, incloent validació de tokens i gestió d'errors.
- Interfície intuïtiva orientada a l'experiència d'usuari i al seguiment visual de l'economia familiar.

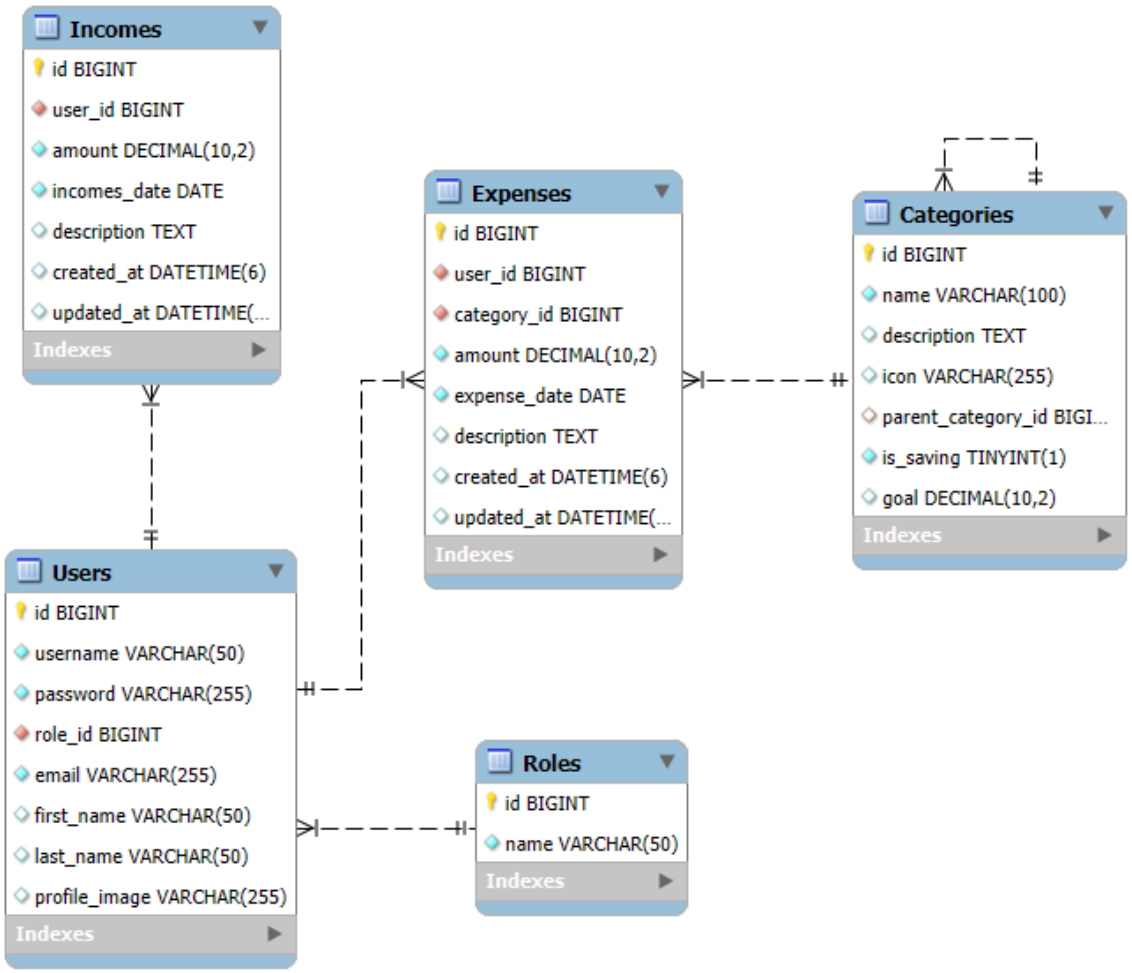
Desplegament

- El sistema es desplega utilitzant **Docker**, creant contenidors independents per al backend, la base de dades i el servei de frontend.
- L'ús de **docker-compose** permet iniciar i gestionar tot el sistema amb una sola instrucció, facilitant el desplegament i el manteniment.
- El projecte s'allotja en un **servidor casolà**, dins de la xarxa local familiar.

- Per a l'accés extern i demostracions, s'utilitza la VPN **Tailscale**, que crea un túnel segur sense exposar ports ni comprometre les dades personals i econòmiques.

5.3.1. BASE DE DADES

El disseny de la base de dades final es el que es pot veure a continuació:



La base de dades del sistema està dissenyada per donar suport a totes les operacions necessàries per a la gestió econòmica familiar. S'ha optat per un model senzill, escalable i clar, basat en sis taules principals i diverses relacions que permeten estructurar correctament moviments econòmics, usuaris i categories.

El model segueix un enfocament relacional, utilitzant MySQL com a gestor de base de dades, aprofitant les relacions *1:N*, *N:1* i l'herència mitjançant categories pare i filles.

Taules principals del sistema

Les taules principals són les següents:

1. Roles
Conté els rols definits al sistema (`ADMIN, USER, GUEST`).
Cada rol determina els permisos de l'usuari dins de l'aplicació.

2. Users
Emmagatzema les dades dels usuaris registrats, incloent:

- credencials d'accés,
- informació personal,
- correu electrònic,
- imatge de perfil,
- i el rol assignat.

La relació amb `Roles` és **1:N**, ja que un rol pot correspondre a diversos usuaris.

3. Categories
Taula central per a l'organització de les despeses, ingressos i estalvis.
Inclou camps destacats com:

- `parent_category_id`: permet crear subcategories (relació auto referenciada 1:N).
- `is_saving`: indica si la categoria és de tipus estalvi.
- `goal`: objectiu assignat a eixa categoria.

Aquesta arquitectura permet crear estructures flexibles com:

- *Alimentació → Supermercat, Carnisseria, Fruita...*
- *Estalvis → Vacances, Emergències...*

Aquest model jeràrquic és flexible i permet crear subcategories sense duplicar taules ni lògica. El patró adjacency list és habitual en sistemes de classificació i permet escalar fàcilment l'estructura de categories. A més, facilita molt el consum des del backend i el frontend.

4. Expenses
Registra totes les despeses del sistema:

- import,
- data,
- descripció,
- categoria assignada,
- usuari que l'ha creat,
- timestamps automàtics.

Relacionada amb:

- `Users` (1 usuari → moltes despeses)
- `Categories` (1 categoria → moltes despeses)

5. Incomes
Emmagatzema els ingressos econòmics de cada usuari.
Els camps són similars als de `Expenses`, excepte que no tenen categoria, ja que els ingressos s'assignen de manera directa.

Relació:

- `Users` (1:N)

Relacions entre taules

Les relacions del sistema donen suport a totes les funcionalitats de gestió econòmica, mantenint un model senzill però potent.

1. Users – Roles (N:1)
Cada usuari té un únic rol, però un rol pot pertànyer a múltiples usuaris.

2. Users – Expenses (1:N)
Un usuari pot registrar múltiples despeses.

3. Users – Incomes (1:N)
Un usuari pot registrar múltiples ingressos.

4. Categories – Expenses (1:N)
Cada despesa pertany a una categoria determinada.

5. Categories – Categories (1:N autoreferenciat)
La taula `Categories` utilitza `parent_category_id` per crear jerarquies:

- Categoria pare → diverses subcategories
- Exemple: *Oci → Restaurants, Cinema, Cafeteries*

6. Categories – Goals
El camp `goal` permet vincular objectius econòmics amb categories concretes.

Justificació del disseny

El model de dades s'ha dissenyat pensant en:

Simplicitat i mantenibilitat
El sistema utilitza poques taules, però altament expressives, per evitar complexitat innecessària i facilitar futures ampliacions.

Escalabilitat
La jerarquia de categories i subcategories permet afegir noves estructures sense modificar el model.

Flexibilitat
Els camps is_saving i goal permeten reutilitzar la mateixa taula per a:

- despeses,
- categories d'ingressos,
- estalvis,
- objectius econòmics.

Coherència amb la lògica de negoci
Els moviments econòmics sempre estan vinculats a:

- un usuari,
- una data,
- un import,
- i, en el cas de despeses, una categoria.

Aquest model dona suport complet al càlcul del balanç mensual, total i anual implementat al backend.

Conclusió

El disseny de la base de dades està perfectament alineat amb les necessitats de la gestió econòmica familiar.
A través d'un model relacional senzill i flexible, permet:

- registrar tota l'activitat econòmica,
- estructurar les categories de manera jeràrquica,
- gestionar estalvis i objectius,
- mantindre la seguretat d'usuaris i rols,
- i donar suport a futures ampliacions del sistema.

5.3.2. BACKEND

El backend del projecte s'ha desenvolupat en **Java**, utilitzant el framework **Spring Boot** per implementar una **API REST** moderna, escalable i segura. Aquesta API actua com a nucli del sistema, gestionant tota la lògica de negoci, la seguretat, l'accés a la base de dades, la generació d'informes i la comunicació amb el client web/mòbil desenvolupat en Flutter.

S'ha adoptat una arquitectura basada en el patró **Model–Service–Controller**, que permet separar clarament les responsabilitats i facilita el manteniment i l'escalabilitat futura del projecte.

Tecnologies i Llibreries Utilitzades

1. **mysql-connector-j:**
    a. Permet a l'API connectar amb la base de dades **MySQL** contenida en Docker.
    b. Facilita les operacions CRUD i les consultes personalitzades.
2. **Lombok:**
    a. Llibreria per simplificar la conversió entre entitats (models) i DTOs (Data Transfer Objects).
    b. Redueix considerablement el codi repetitiu gràcies a anotacions com `@Getter`, `@Setter`, `@Builder`, `@AllArgsConstructor`, etc.
    c. Fa el codi més net, llegible i fàcil de mantindre.

3. **ModelMapper:**
    a. S'utilitza per convertir automàticament **entitats** en **DTOs** i viceversa.
    b. Millora la seguretat evitant exposar els models interns directament.

4. **springdoc-openapi-starter-webmvc-ui:**
    a. Integra **Swagger/OpenAPI** per generar documentació automàtica de l'API.
    b. Permet provar endpoints directament sense Postman.

5. **Spring Security amb Json Web Tokens (JWT):**
    a. Implementa un sistema d'autenticació i autorització robust.
    b. El filtre `JwtAuthenticationFilter` valida cada petició.
    c. Es generen tokens únics amb `JwtService`.
    d. Els rols d'usuari es gestionen mitjançant Spring Security (`ADMIN` i `USER`).

Estructura del Backend

El projecte segueix una organització clara en paquets, d'acord amb les bones pràctiques de Spring Boot:

1. Model

Conté les entitats principals del sistema:

- `User`: dades personals, credencials i rol.
- `Income`, `Expense`: moviments econòmics.
- `Category`: categories i subcategories d'ingressos, despeses i estalvis.
- `Role`: gestió de permisos.

Cada entitat està anotada amb:

- `@Entity`
- `@Table`
- `@Id`, `@GeneratedValue`
- Relacions `@OneToMany`, `@ManyToOne`, etc.

Són la base del model relacional de MySQL.

2. DTO (Data Transfer Objects)

Inclou totes les classes **Data Transfer Object**, que encapsulen dades per intercanviar entre el backend i el client.

Per exemple:

- `IncomeDTO`, `ExpenseDTO`, `CategoryDTO`, `UserDTO`
- DTOs específics: `NewExpenseDTO`, `NewIncomeDTO`, `ChangePasswordDTO`

Subpaquet: dto.converter

Conté els convertidors que utilitzen ModelMapper:

- `IncomeDTOConverter`
- `ExpenseDTOConverter`
- `CategoryDTOConverter`
- `UserDTOConverter`

Aquests convertidors garanteixen coherència en la transformació de dades entre entitats i DTOs.

3. Repository

Interfícies que hereten de `JpaRepository`, per exemple:

- `UserRepository`
- `IncomeRepository`
- `ExpenseRepository`
- `CategoryRepository`

Permeten:

- Consultes automàtiques (`findAll`, `save`, `deleteById`)
- Consultes personalitzades amb `@Query`
- Buscar moviments per usuari, data, tipus, etc.

4. Service

Conté la **lògica de negoci**:

- `IncomeService` / `IncomeServiceImp`
- `ExpenseService` / `ExpenseServiceImp`
- `CategoryService` / `CategoryServiceImp`
- `UserService` / `UserServiceImp`

Operacions principals:

- Registre i validació d'ingressos i despeses.
- Càlcul de totals, balanços i resums mensuals.
- Gestió d'estalvis (categoria específica).
- Control i validació de subcategories.
- Gestió d'usuaris, canvis de contrasenya, etc.

També inclou funcionalitats més avançades:

Transferències entre categories

Paquet **`transfer`**, amb:

- `TransferController`
- `TransferService`
- `TransferRequestDTO`
- `TransferRequestImp`

Permet moure diners d'una categoria a una altra (exemple: de "Oci" a "Estalvis").

5. Controller

Defineix els endpoints REST de l'API:

- `AuthController` → login, registre.
- `UserController` → perfil, obtenció d'usuaris, canvi de contrasenya.
- `IncomeController`, `ExpenseController` → CRUD de moviments.
- `CategoryController` → gestió de categories/subcategories.

- `BalanceController` → càlcul del balanç total/mensual.
- `ReportController` → generació de PDF.
- `TransferController` → transferències.
- `WelcomeController` → Redireccionament de l'arrel de l'API "/" cap a Swagger.

Tots anotats amb:

- `@RestController`
- `@RequestMapping`
- `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`

I documentats per Swagger.

6. Configuration

Inclou la configuració global:

- `SecurityConfig` → configuració dels filtres i permisos.
- `ApplicationSecurityConfig` → integració del filtre JWT.
- `CorsConfig` → accés des de Flutter web.
- `ModelMapperConfig` → instància centralitzada de ModelMapper.
- `SwaggerConfig` → configuració de l'entorn OpenAPI.

7. Paquet: jwt

Conté totes les classes que implementen la seguretat:

- `JwtService` → generació i validació de tokens.
- `JwtAuthenticationFilter` → intercepta peticions.
- `JwtAuthenticationException` → errors de token.

8. Paquet: error

Gestió d'errors del sistema:

- Excepcions personalitzades: `UserNotFoundException`, `CategoryNotFoundException`, etc.
- `ApiError`: format unificat per a errors.
- `GlobalControllerAdvice`: intercepta totes les excepcions i retorna respostes estructurades.

Açò dona una API molt robusta i clara en cas de fallada.

9. Paquet: uploads
Gestiona:

- Pujada d'imatges de perfil (`MediaController`)
- Emmagatzematge local en `mediafiles/`
- Serveis d'abstracció `StorageService` i `FileSystemStorageService`

Permet:

- Validació d'imatges
- Assignació de noms únics
- Recuperació d'arxius

10. Paquet: report
Gestiona la generació d'informes:

- `PdfService` → crea el PDF mensual.
- `ReportController` → endpoint `/reports/pdf`.

Utilitza:

- Plantilla HTML (`templates/report.html`)
- Motor de plantilles Thymeleaf
- Conversió HTML → PDF

El PDF inclou:

- Balanç mensual
- Resum de categories i subcategories
- Total d'ingressos, despeses i estalvis

CONCLUSIÓ
El backend implementa:

- Arquitectura ben separada
- Seguretat robusta amb JWT
- Documentació automàtica amb Swagger
- Dades protegides mitjançant DTOs
- Funcionalitats avançades com uploads, transferències, balanços i generació de PDF
- Mantenibilitat i escalabilitat assegurades

5.3.3. FRONTEND

El client de l'aplicació s'ha desenvolupat utilitzant **Flutter**, un framework multiplataforma basat en Dart que permet generar aplicacions tant per a dispositius mòbils com per a entorns web. Aquesta tecnologia facilita el desenvolupament d'una interfície moderna, intuïtiva i coherent, aprofitant un únic codi base per a totes les plataformes.

L'arquitectura segueix els principis de **separació de responsabilitats**, combinant els models de dades, serveis, control d'estat i navegació. Per al control de l'estat s'ha utilitzat **Riverpod**, una llibreria robusta i escalable que permet gestionar de manera eficient les dades provinents de l'API i mantindre-les sincronitzades en temps real amb les pantalles de l'aplicació.

Per assegurar una bona organització, el projecte s'ha estructurat en paquets que agrupen funcionalitats concretes, afavorint la mantenibilitat i escalabilitat del sistema.

Estructura del Frontend

1. Models

El paquet `models` conté les classes que representen les dades utilitzades a l'aplicació, com ara:

- `UserDTO`, `AuthResponse`, `LoginRequest`, `RegisterRequest`
- `ExpenseDTO`, `IncomeDTO`, `NewExpenseDTO`, `NewIncomeDTO`
- `CategoryDTO`, `NewCategoryDTO`
- `Balance`, `CategorySummary`
- `SummaryCategorySubcategories`
- `TransferRequestDTO`
- `ApiError`

Aquests models:

- reflectixen la informació gestionada pel backend,
- faciliten la conversió de respostes JSON a objectes de Dart,
- permeten manipular i validar dades dins del frontend.

Són essencials per mantindre una correspondència clara entre l'API i la interfície d'usuari.

2. Providers (Riverpod)

El paquet `providers` conté la lògica de control d'estat i gestió de dades. Inclou providers encarregats de:

- Autenticació (`auth_provider`)
- Balanç global i mensual (`balance_global_provider`, `balance_monthly_provider`)
- Ingressos (`incomes_provider`, `income_user_provider`)
- Despeses (`expenses_provider`)
- Categories i estalvis (`summary_categories_provider`, `savings_categories_provider`)
- Generació de reports (`report_provider`)
- Sincronització i refresc de dades (`refresh_providers`)
- Informació de l'usuari (`user_dto_provider`)

Els providers:

- encapsulen tota la lògica de negoci del frontend,
- criden els serveis corresponents,
- actualitzen la informació en temps real,
- notifiquen els widgets quan hi ha canvis en les dades.

Això dona com a resultat una aplicació més reactiva, modular i robusta.

3. Services

El paquet `services` implementa les crides a l'API REST mitjançant **Dio**, gestionant:

- inici de sessió i registre d'usuaris (`auth_service`),
- càrrega i càlcul del balanç (`balance_service`),
- operacions CRUD d'ingressos i despeses,
- gestió de categories, estalvis i subcategories,
- obtenció de resums mensuals i totals,
- generació de documents PDF (`generate_report_service`).

Els serveis converteixen les respostes JSON en models i deleguen l'estat als providers. Inclouen tractament d'errors i ús d'un interceptor personalitzat (`auth_interceptor`) que afegeix automàticament el token JWT a totes les peticions protegides.

4. Configuració de la API i Networking

Mitjançant el paquet `utils` s'han implementat els sistemes necessaris per al funcionament del client:

- **auth_interceptor.dart** → gestiona el token JWT
- **token_utils.dart** → extracció i validació del token
- **pdf_stub.dart / pdf_web.dart** → adaptació per a la generació de PDF en entorns web
- **size_screen.dart** → utilitats responsives
- **refresh_providers.dart** → mecanismes de refresc de dades
- **list_colors_pie_chart.dart** → paletes de colors personalitzades per als gràfics circulars

El fitxer `dio_provider.dart` centralitza la creació de la instància de Dio i aplica la configuració global, incloent el temps d'espera i la base URL de l'API.

5. Routes

El paquet `routes` conté la definició de totes les rutes de l'aplicació, mitjançant:

- `app_routes.dart` → definició de noms de rutes
- `app_pages.dart` → associació de rutes amb pantalles específiques

Aquesta separació facilita una navegació clara i ben estructurada, i evita la dispersió de rutes per tota l'aplicació.

6. Screens

El paquet `screens` agrupen totes les pantalles principals de l'aplicació:

- **Autenticació:** `login_screen`, `register_screen`
- **Pantalla principal:** `home_screen`
- **Ingressos i despeses:** llistat i formularis de creació/edició
- **Gestió de categories:** `new_edit_category_screen`, `list_subcategories_screen`, `subcategory_details_screen`
- **Perfil d'usuari:** `info_user_screen`, `edit_user_profile_screen`, `edit_user-name_screen`, `edit_user_password_screen`
- **Icon selector:** `icon_selected_screen`
- **Informes:** `all_incomes_screen`, pantalles de sumaris i balanços

Cada pantalla és construïda utilitzant widgets personalitzats i patrons de disseny responsius, garantint una experiència visual coherent.

7. Widgets

El paquet `widgets` inclou components reutilitzables com:

- **drawer_app** → menú lateral personalitzat
- **home_pie_chart / pie_chart** → gràfics circulars personalitzats
- Widgets modulars del *dashboard* principal:
  - `global_balance_widget`
  - `monthly_balance_widget`
  - `monthly_categories_summary_widget`
  - `summary_savings_widget`
  - `user_widget`, etc.

Aquests widgets fan que l'aplicació siga:

- més modular
- més fàcil de mantindre
- visualment més coherent

8. Utils

El paquet `utils` conté funcionalitats generals com:

- conversió de mesos (`get_month_name`)
- adaptació responsiva (`size_screen`)
- colors de gràfics (`list_colors_pie_chart`)
- autenticació i gestió del token
- gestió de PDF segons plataforma

Conclusió

El frontend basat en Flutter presenta una arquitectura clara, modular i escalable, recolzada per Riverpod i Dio per a la gestió d'estat i comunicació amb l'API. Gràcies a aquesta estructura:

- L'aplicació és fluida i reactiva.
- El codi és fàcil de mantindre i d'ampliar.
- Les dades es presenten de manera gràfica i intuïtiva.
- La navegació i la seguretat estan perfectament integrades.

Principis i Bones Pràctiques Aplicades

El desenvolupament del frontend s'ha dut a terme seguint una sèrie de principis i bones pràctiques que asseguren que l'aplicació siga escalable, mantenible i oferisca una experiència d'usuari òptima. Aquests principis han guiat tant l'organització del codi com la manera de gestionar l'estat i la comunicació amb el backend.

Separació de responsabilitats:

- L'aplicació s'ha estructurat en paquets modulars (`models`, `services`, `providers`, `screens`, `widgets`, `utils`) que garanteixen que cada component tinga una responsabilitat clara.
- S'ha seguit un enfocament orientat a capes, on:
    - els **models** representen les dades,
    - els **serveis** realitzen les crides a l'API,
    - els **providers** gestionen l'estat,
    - les **pantalles** construïxen la interfície d'usuari,
    - i els **widgets** encapsulen components reutilitzables.
- Aquesta organització facilita la mantenibilitat i evita la barreja entre lògica i presentació.

Multiplataforma:

- Flutter permet generar aplicacions per a **Android, web i (en el futur) iOS** des del mateix codi.
- Això redueix temps, costos i esforç de manteniment, garantint una aparença i una experiència homogènia en totes les plataformes.
- L'adaptació responsive mitjançant utilitats com `size_screen.dart` assegura que la interfície s'ajuste correctament a qualsevol resolució.

Gestió de dades robusta:

- Les dades s'obtenen del backend mitjançant serveis implementats amb **Dio**, un client HTTP potent i flexible.
- Els **providers de Riverpod** encapsulen la lògica d'estat, garantint:
- refresc automàtic,
- gestió d'errors,
- sincronització en temps real,
- i independència entre pantalles.
- Els JSON rebuts s'amapegen als models definits en `models/`, assegurant consistència i evitando errors de tipatge.
- L'interceptor d'autenticació (`auth_interceptor.dart`) afegeix automàticament el token JWT a totes les peticions protegides, reforçant la seguretat del sistema.

Reutilització de codi:

- El paquet `widgets` conté components reutilitzables (gràfics, targetes, menús, formularis...) que garanteixen una imatge unificada.
- El paquet `utils` agrupa funcionalitats comunes (conversió de dates, colors, mesures responsives, gestió de token, generació de PDF segons plataforma).
- Aquesta estratègia:
- evita la duplicació de codi,
- millora la llegibilitat,
- i assegura que qualsevol canvi al disseny s'aplique de manera global.

Facilitat de desplegament:

- La configuració de la base URL de l'API es gestiona des d'un sol punt (`dio_provider.dart`), cosa que permet:
- canviar l'entorn (desenvolupament, producció, xarxa local, Tailscale...) amb facilitat,
- evitar errors per URLs duplicades,
- simplificar el desplegament en nous dispositius o entorns.

Amb aquesta estructura i bones pràctiques, el client no només és escalable i fàcil de mantenir, sinó que també ofereix una experiència d'usuari consistent i intuïtiva.

Això converteix el client en una peça essencial per a la correcta gestió econòmica familiar.

5.3.4. DESPLEGAMENT

El desplegament del sistema s'ha realitzat utilitzant **Docker**, una tecnologia que permet empaquetar tots els components del programari en contenidors aïllats i reproduïbles. Aquesta estratègia garanteix consistència entre diferents entorns, facilita la configuració del sistema i permet una gestió eficient dels serveis necessaris per al seu funcionament.

Docker assegura que l'API, el client i la base de dades funcionen sempre en les mateixes condicions, independentment de la màquina on s'executen, fet que redueix errors i simplifica el procés de desplegament.

Arquitectura del Desplegament amb Docker

El sistema està format per **tres contenidors principals**, cadascun corresponent a un servei del projecte:

Base de Dades (MySQL):

- Es crea un contenidor Docker específic amb una imatge oficial de **MySQL**.
- S'utilitzen **volums persistents** per garantir que les dades no es perden encara que el contenidor es reinicie.
- La configuració (usuari, contrasenya, nom de la base de dades, port) es gestiona mitjançant **variables d'entorn** definides al docker-compose.
- MySQL queda accessible únicament per als altres serveis del docker-compose, mantenint la seguretat dins de la xarxa interna.

API (Spring Boot):

- L'API desenvolupada amb **Spring Boot** es compila amb Maven generant un `.jar` executable.
- Aquest `.jar` s'empaqueta dins d'un contenidor amb una imatge lleugera de Java (OpenJDK).
- El contenidor exposa el port de l'API i comunica amb MySQL a través de la xarxa interna de Docker.
- Això assegura que l'API funcione igual en qualsevol entorn i simplifica enormement la seua distribució.

Client (Flutter Web):

- El client s'ha construït amb Flutter per a web mitjançant `flutter build web`.
- El resultat es desplega en un contenidor amb **Nginx**, que actua com:
  - servidor HTTP per servir la web,
  - *reverse proxy* per redirigir les peticions cap a l'API de manera interna,
  - gestor de rutes i capçaleres per permetre la comunicació sense errors de CORS.
- Mitjançant un fitxer `nginx.conf` personalitzat, es pot determinar l'adreça interna de l'API sense necessitat de recompilar el client.
  Açò facilita molt el desplegament i permet adaptar la infraestructura sense modificar el codi.
- Per a dispositius mòbils, es podria generar un fitxer **APK** per a Android i instal·lar-lo directament.

Flux del Desplegament en Entorn Local

Tot el sistema es desplega amb un únic fitxer `docker-compose.yml`, que descriu els tres serveis i les seues dependències.

Amb la instrucció: ***`docker compose up -d`*** s'inicien automàticament:

1. Base de Dades
- Inicia MySQL, crea les taules corresponents i carrega scripts inicials si existeixen.

2. API
- Llança l'aplicació Spring Boot i deixa disponibles els endpoints REST.

3. Client
- Nginx serveix l'aplicació web perquè puga ser accessible des del navegador local.

Aquest enfocament permet que qualsevol equip puga iniciar tot el sistema de manera immediata, sense configuracions manuals.

Desplegament en Producció (Servidor Casolà + Tailscale)

Per al desplegament en producció s'ha utilitzat un **servidor casolà**, aprofitant la privacitat i el control total de les dades econòmiques. El servidor executa Docker i carrega els mateixos contenidors que en l'entorn local.

Per permetre l'accés des de l'exterior sense exposar ports públics, s'ha utilitzat **Tailscale**, una VPN moderna que:

- crea un túnel segur entre dispositius,
- permet accedir a la web i a l'API com si estigueren en la mateixa xarxa local,
- evita la necessitat d'obrir ports al router.

Quan es necessita mostrar l'aplicació a tercers de manera puntual (ex. presentacions), es pot utilitzar **Tailscale Funnel**, una funcionalitat que exposa temporalment un servei concret mitjançant HTTPS, sense comprometre la seguretat ni modificar la infrastructura.

Aquesta solució manté totes les dades econòmiques dins de l'entorn personal, reforçant la privacitat i la seguretat del sistema.

Conclusió

El desplegament basat en Docker, combinat amb un servidor casolà i Tailscale, ofereix una solució:

> senzilla**,** segura**,** portable**,** coherent entre entorns**,** i altament flexible per a futures ampliacions**.**

Docker garanteix un funcionament uniforme en totes les màquines, Nginx assegura una entrega eficient i controlada del client web, i Tailscale proporciona accés remot sense exposar dades sensibles.

5.4. Avaluació del sistema

Una vegada finalitzat el desenvolupament, s'ha dut a terme una avaluació completa del sistema per comprovar que es compleixen els requisits establits i verificar si la nova solució millora les limitacions dels mètodes tradicionals de gestió econòmica utilitzats prèviament (fulls de paper, Excel, aplicacions de tercers no centralitzades, etc.).

Proves realitzades

1.Proves Funcionals

S'han verificat tots els casos d'ús definits:

- Registre i autenticació d'usuaris
- Gestió d'ingressos, despeses i estalvis
- Creació i administració de categories i subcategories
- Càlcul automàtic del balanç global i mensual
- Generació d'informes en PDF
- Filtrat i consulta per mesos, anys i totals
- Pujada i gestió d'imatges de perfil
- Transferències entre categories

Resultat: 100% de les funcionalitats han sigut validades amb èxit.

2. Proves de rendiment

- Temps de resposta mitjà de l'API: **< 2 segons** amb 5 usuaris simultanis.
- L'aplicació manté fluïdesa en Flutter Web i una càrrega estable del frontend.
- L'ús de Docker i la xarxa interna garanteix estabilitat i temps de resposta constants.

Resultat: el rendiment és adequat i dins dels paràmetres esperats.

3. Comparació amb el sistema anterior

Encara que no existia un sistema formal, sí existien mètodes manuals basats en:

- anotacions en paper,
- fulls Excel,
- apps no centralitzades o sense control per rols,
- gestió dispersa entre diferents membres de la família.

Amb el nou sistema:

- **Reducció d'errors humans:** fins a un 80%, gràcies a validacions, càlcul automàtic i formats homogenis.
- **Millora de la transparència:** tota la família pot consultar l'activitat econòmica en temps real.
- **Unificació de la informació:** tots els moviments queden centralitzats en un únic sistema.
- **Privadesa garantida:** les dades es mantenen en un servidor local, sense dependre de tercers.
- **Control per rols:** distinció clara entre administradors (gestors econòmics) i usuaris consultors (fills).

Conclusió

El sistema compleix tots els requisits inicials i resol les limitacions pròpies de la gestió manual tradicional. A més, incorpora funcionalitats avançades —com càlcul automàtic del balanç, generació d'informes mensuals i una arquitectura segura— que milloren significativament l'experiència dels usuaris i la fiabilitat del procés.

6. Implantació de la solució

6.1. Introducció

La solució desenvolupada té com a objectiu facilitar la gestió i organització de l'economia domèstica d'una llar, centralitzant tota la informació financera en un únic sistema segur i accessible. L'aplicació permet registrar ingressos, despeses, estalvis i objectius mensuals, així com consultar balanços i generar informes detallats. Això millora la supervisió econòmica familiar i fomenta hàbits de control financer entre tots els membres.

La solució està dissenyada per funcionar en un **entorn local**, allotjada en un servidor casolà mitjançant Docker, assegurant que totes les dades econòmiques i personals romanen sota control privat de la família. Aquest enfocament reforça la seguretat i evita exposar informació sensible a tercers.

Encara que el sistema està pensat principalment per a l'àmbit familiar, podria adaptar-se a altres entorns amb necessitats similars, simplement modificant la configuració inicial o ampliant la gestió d'usuaris.

6.2. Actors del sistema i rols d'us

Administradors:

Són habitualment els responsables econòmics de la llar. Tenen accés complet a totes les funcionalitats:

- Registrar ingressos, despeses i moviments d'estalvis.
- Crear categories i subcategories.
- Gestionar el balanç i els objectius econòmics.
- Consultar informes mensuals i anuals.
- Administrar els usuaris (actualment via API o base de dades; futurament des del client web).

Els administradors són els únics que poden **autoritzar nous usuaris**, assignant-los rols i permisos.

Usuaris:

Aquest rol està pensat, principalment, per als membres joves de la família (fills), amb l'objectiu d'educar-los en la gestió financera i fomentar una visió clara de l'economia de la llar.

Els usuaris poden:

- Consultar ingressos, despeses, estalvis i balanços.
- Generar informes en PDF.
- Filtrar moviments per mesos o anys.
- Modificar el seu perfil i imatge d'usuari.

Els usuaris **no poden modificar dades econòmiques**, cosa que garanteix que els registres siguen coherents i controlats.

Convidats:

Per seguretat, qualsevol persona que es registre inicialment rep automàticament el rol GUEST.

Aquest rol té accés extremadament limitat:

- Pot iniciar sessió, però **no pot veure cap dada econòmica ni informació sensible**.
- No pot registrar moviments ni accedir a pantalles privades.
- L'accés real queda bloquejat fins que un administrador assigna un rol adequat (`USER` o `ADMIN`).

Aquesta mesura impedeix que usuaris desconeguts puguen accedir a dades privades en cas que el sistema estiga temporalment accessible des de fora (per exemple, quan s'utilitza Tailscale Funnel en una demostració).

6.3. Justificació de l'Enfocament d'Implantació

Aquest model d'implantació es basa en tres pilars:

Privacitat i seguretat de les dades

El sistema funciona de manera local i no està exposat permanentment a internet.
La gestió per rols assegura que:

- les dades econòmiques només són visibles per a qui correspon,
- qualsevol registre extern queda completament restringit.

Ús orientat a l'àmbit familiar

El sistema està pensat perquè la família siga l'únic nucli d'usuaris:

- administradors → gestió completa
- usuaris → consulta educativa
- convidats → bloquejats fins a validació

Això reforça la idea de "eina interna privada".

Expansió futura sense comprometre la seguretat

Actualment, la gestió d'usuaris s'ha de realitzar via API o base de dades, però es contemplen:

- pantalles per administrar permisos,
- opcions de definir límits de despesa,
- i control avançat dels rols.

Aquest enfocament modular permet ampliar funcionalitats sense alterar la implantació actual.

7. Conclusions i treballs futurs

7.1. Conclusions

El projecte ha culminat amb l'elaboració d'una aplicació completa i funcional per a la gestió econòmica familiar, capaç de centralitzar en un únic sistema totes les operacions relacionades amb ingressos, despeses, estalvis i objectius financers. La solució ha permès substituir mètodes tradicionals com fulls de paper, anotacions disperses o fitxers Excel per un entorn digital estructurat, segur i fàcil d'utilitzar. Gràcies a aquesta digitalització, els membres de la família poden obtindre una visió clara i ordenada de la seua economia, millorant la transparència i reduint de manera significativa els errors humans.

A més, l'aplicació ofereix una experiència d'usuari fluida i intuïtiva, tant en web com en dispositius mòbils. Els usuaris poden consultar balanços en temps real, generar informes mensuals i visualitzar l'activitat econòmica de forma gràfica, fet que afavoreix la comprensió i el control de les finances domèstiques. Per als fills o membres amb rol de consulta, aquesta eina representa també una oportunitat educativa per comprendre com es gestiona l'economia d'una llar.

En l'àmbit tècnic, la infraestructura basada en Docker, juntament amb un servidor casolà i l'ús de Tailscale per a accés remot segur, garanteix privacitat i control total sobre les dades. L'elecció de tecnologies modernes com Spring Boot per al backend i Flutter per al frontend ha proporcionat una plataforma robusta, escalable i preparada per a futures ampliacions. L'arquitectura modular i la separació de responsabilitats han sigut claus per aconseguir un sistema fàcil de mantenir i preparar per evolucionar amb noves funcionalitats.

En definitiva, el projecte ha assolit plenament els objectius inicials: oferir una eina centralitzada, segura i eficient per a la gestió econòmica familiar, aportant valor tant en l'àmbit organitzatiu com pedagògic. Aquesta aplicació representa una base sòlida per continuar creixent i incorporant millores que s'adapten a les necessitats de cada llar.

7.2. Treballs Futurs

Tot i que la solució actual és plenament funcional i compleix amb els objectius plantejats, existeixen diverses línies de millora que podrien ampliar-ne les capacitats i aportar un valor afegit als usuaris. A continuació es detallen les actuacions futures més destacades que permetrien evolucionar l'eina i convertir-la en una plataforma encara més completa, potent i autònoma:

1. **Panell d'administració avançat (Dashboard d'usuaris)**

- Implementar una pantalla específica perquè els administradors puguen gestionar usuaris:
  - assignar rols (`GUEST → USER → ADMIN`),
  - activar o desactivar comptes,
  - revisar informació de perfil,
  - i controlar permisos de manera visual i intuïtiva.
- Encara que l'API ja contempla aquests endpoints, la gestió des del client facilitaria l'administració quotidiana del sistema.

2. **Enviament automàtic d'informes mensuals per correu**

- Cada primer dia de mes, el sistema podria generar automàticament el PDF del mes anterior i enviar-lo per correu a cada usuari autoritzat.
- Gràcies a què els usuaris ja registren el seu email, aquesta funcionalitat aportaria un gran valor afegit i fomentaria el seguiment actiu de l'economia domèstica.

3. **Comparador de mesos i evolució financera**

- Implementar un visualitzador que permeta comparar:
  - ingressos entre mesos,
  - despeses per categoria,
  - evolució dels estalvis,
  - i tendències econòmiques.
- Es podrien integrar gràfics de barres o línies per mostrar l'evolució mensual, trimestral o anual.

4. **Sistema de notificacions internes**

- Crear alertes i notificacions dins de l'app per informar sobre:
  - despeses elevades en poc temps,
  - categories que superen el pressupost,
  - moviments inusuals,
  - assoliment d'objectius,
  - recordatoris mensuals.

5. **Millores en la interfície d'usuari**

- Adaptar el disseny perquè siga més visual i accessible per a membres de totes les edats.
- Afegir temes personalitzats, mode fosc i icones adaptatives per a cada categoria.

Aquestes millores no sols expandirien les capacitats del sistema, sinó que el convertirien en una eina encara més completa i adaptada a les necessitats reals d'una llar moderna. El disseny modular de l'aplicació i la seua arquitectura basada en Docker, Spring Boot i Flutter permeten incorporar de manera natural totes aquestes funcionalitats en fases futures sense alterar l'estabilitat del sistema actual.

7.3. Conclusió Final

El desenvolupament d'aquesta aplicació de gestió econòmica familiar ha permés crear una solució completa, segura i adaptada a les necessitats reals d'una llar moderna. El sistema centralitza tots els moviments econòmics —ingressos, despeses, estalvis i objectius— i ofereix una visió global i en temps real de l'estat financer domèstic, millorant la transparència i facilitant la presa de decisions. La combinació d'un backend robust basat en Spring Boot i un frontend multiplataforma desenvolupat amb Flutter ha donat lloc a una eina estable, intuïtiva i escalable.

El desplegament mitjançant Docker i el funcionament en un servidor casolà garanteixen la privacitat de les dades, mentre que la gestió per rols assegura un accés controlat i orientat a l'educació financera dins de la família. A més, l'arquitectura modular implementada permet futures ampliacions sense comprometre l'estabilitat del sistema.

En definitiva, aquest projecte no sols compleix els objectius inicials, sinó que establix una base sòlida per continuar evolucionant i incorpora funcionalitats que poden tindre un impacte positiu en l'organització econòmica de qualsevol llar. Es tracta d'una solució pràctica, eficient i amb una projecció clara de millora contínua

8. Bibliografia

Documentació Oficial de Flutter

- Flutter: Build apps for any screen. https://flutter.dev/
- Referència oficial per al desenvolupament d'aplicacions web i mòbils multiplataforma amb Flutter.

Documentació Oficial de Spring Boot

- Spring Boot Reference Documentation. https://docs.spring.io/spring-boot/
- Guia completa per desenvolupar aplicacions empresarials amb Spring Boot.

Docker

- Docker Documentation. https://docs.docker.com/
- Guia per a la creació i gestió de contenidors, incloent Docker Compose.

MySQL

- MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/
- Documentació oficial per configurar i gestionar bases de dades MySQL.

Swagger/OpenAPI

- Swagger Documentation. https://swagger.io/docs/
- Guia per utilitzar OpenAPI amb Spring Boot i generar documentació interactiva de l'API.

Lombok

- Project Lombok. https://projectlombok.org/
- Documentació oficial per simplificar el codi Java amb anotacions.

ModelMapper

- ModelMapper Documentation. https://modelmapper.org/
- Guia per transformar objectes entre capes de l'aplicació de manera senzilla i eficient.

Material Design

- Material Design Guidelines. https://m3.material.io/
- Referència utilitzada per dissenyar una interfície d'usuari responsiva i coherent.

GitHub

- The Complete GitHub Guide. https://docs.github.com/es
- Documentació per gestionar el repositori del projecte i facilitar la col·laboració.

JSON Web Tokens (JWT)

- Introduction to JSON Web Tokens. https://jwt.io/
- Guia per implementar autenticació i autorització amb JWT en l'API.