Hochschule Reutlingen
Reutlingen University

SS 2025

Prof. Dr. Natividad Martínez Madrid

- Project Documentation-

UNIWALLET

- ONE APP FOR ALL YOUR BANK ACCOUNTS -

Submitted by:

Óscar Gómez González

Yeray Navascués Trincado

Contact address:

oscar.gomez_gonzalez@student.reutlingen-university.de

yeray.navascues_trincado@student.reutlingen-university.de

Submitted on: 28.05.2025

Abstract: *The goal of this project is to design and develop a mobile application that serves as a multi-bank accounts manager. The app aims to consolidate and display information from multiple bank accounts in a single, unified interface, providing users with an efficient way to monitor and manage their finances across different banks.*

TABLE OF CONTENTS

1. Introduction

This chapter covers the idea and motivation as well as the goals of our project.

1.1 Idea and Motivation

In today’s world, individuals often maintain multiple bank accounts—checking, savings, credit—and yet lack a single, unified view of their complete financial picture. Switching between different bank apps or websites to track balances, monitor spending, or move money is cumbersome and time-consuming. Users struggle to understand their overall cash flow and net worth immediately, and small transfers between accounts can easily be forgotten or delayed.

UniWallet tackles these pain points by aggregating disparate banking relationships into one sleek mobile interface for iOS and Android. With UniWallet, you can:

- View aggregated balances across all linked accounts in real time.
- Analyse total spending and income trends across all banking relationships.
- Transfer funds instantly between any of your linked accounts without switching apps.
- Track your net worth—including incomes, and expenses—on a single screen.

The result is a seamless, transparent experience that helps users regain control of their finances, make smarter decisions, and save time by handling everything from one place.

1.2 Goals

The primary goal of this project is to provide users with a single, seamless mobile platform that aggregates all their bank accounts into one unified view, empowering them to effortlessly monitor balances, track net worth, analyse spending, and move money between accounts without ever leaving the app.

2. Requirements Engineering

2.1 Storyboard

Defining the needs and expectations of users was a crucial step in shaping the application. Managing multiple bank accounts and gaining a clear understanding of one's overall financial situation is often unnecessarily complex—a challenge for individuals across all ages and backgrounds. UniWallet is designed to simplify this process, delivering an intuitive and seamless experience that serves a diverse audience, regardless of technical skill or familiarity with financial platforms.

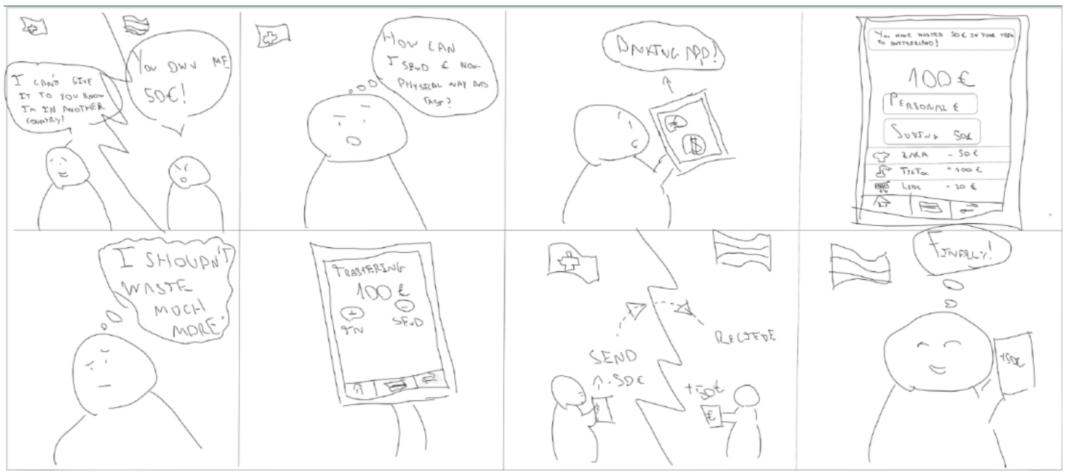

I CAN'T GIVE IT TO YOU KNOW I'M IN ANOTHER COUNTRY!
YOU OWN ME 50€!
HOW CAN I SEND € NON-PHYSICAL WAY AND FAST?
BANKING APP!
YOU HAVE WASTED 50€ IN YOUR TRIP TO SWITZERLAND!
100 €
PERSONAL €
SAVING 50€
ZARA - 50 €
TIKTOK + 100 €
LIDL - 30 €
I SHOULDN'T WASTE MUCH MORE
TRANSFERING
100 €
IN
SEND
SEND
-50€
RECIEVE
+50€
FINALLY!
+50€

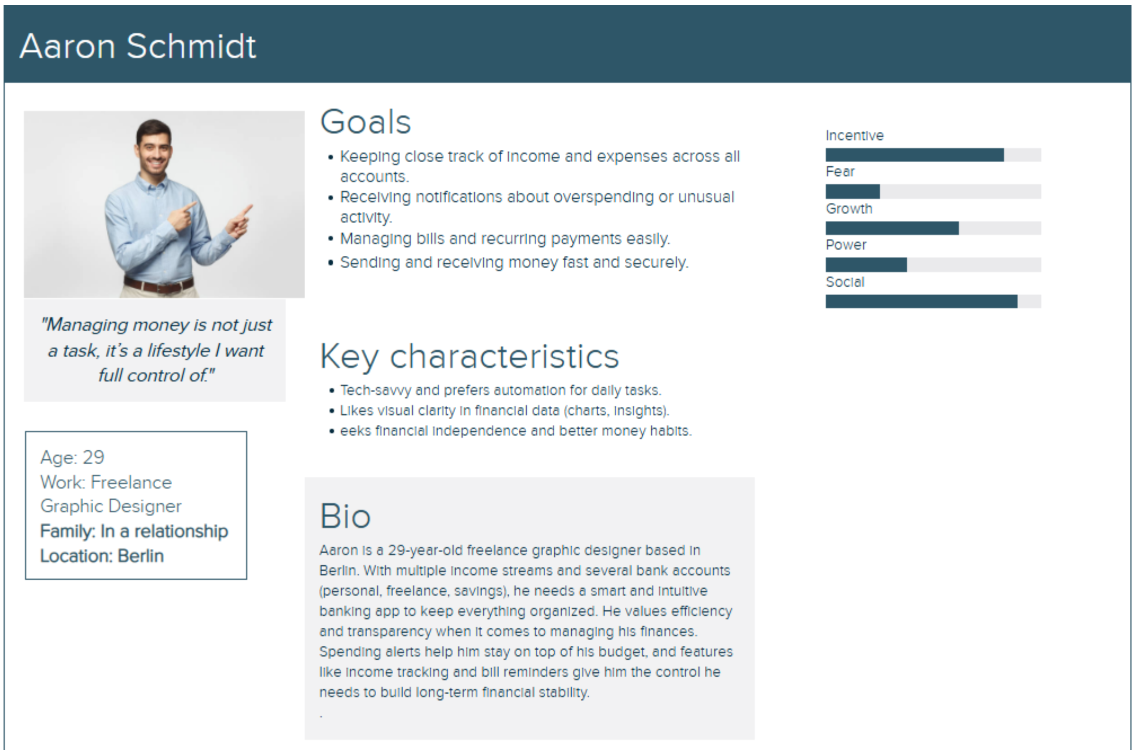
FIGURE 1 STORYBOARD

2.2 PERSONAS

Personas allow to gain a better understanding of the needs and restrictions of the user group. The application has two primary target groups.

PERSONA 1

Aaron Schmidt works as a freelance graphic designer in Berlin. With multiple income streams and bank accounts, he's busy managing his finances most of the time. He doesn't like disorganized money management, preferring smart tools that track everything automatically. Aaron wants clear alerts about his spending and values features that help him build better financial habits without extra effort.



Aaron Schmidt

"Managing money is not just a task, it's a lifestyle I want full control of."

Age: 29
Work: Freelance Graphic Designer
Family: In a relationship
Location: Berlin

Goals

- Keeping close track of income and expenses across all accounts.
- Receiving notifications about overspending or unusual activity.
- Managing bills and recurring payments easily.
- Sending and receiving money fast and securely.

Key characteristics

- Tech-savvy and prefers automation for daily tasks.
- Likes visual clarity in financial data (charts, insights).
- eeks financial independence and better money habits.

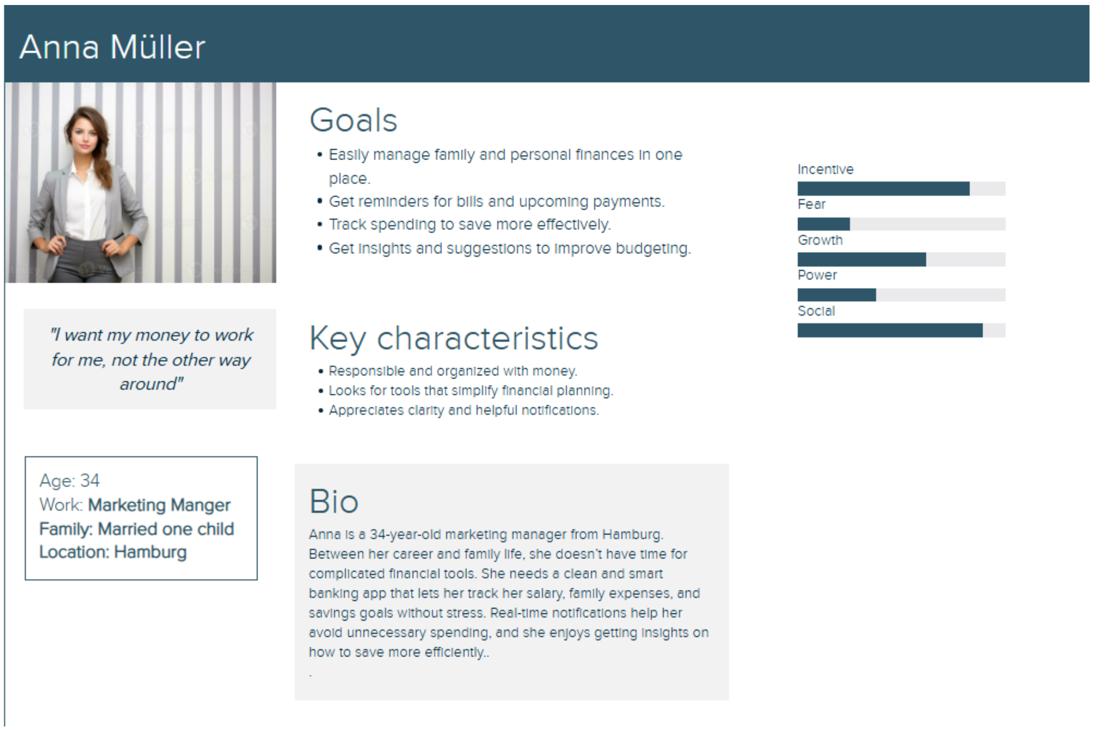
Bio

Aaron is a 29-year-old freelance graphic designer based in Berlin. With multiple income streams and several bank accounts (personal, freelance, savings), he needs a smart and intuitive banking app to keep everything organized. He values efficiency and transparency when it comes to managing his finances. Spending alerts help him stay on top of his budget, and features like income tracking and bill reminders give him the control he needs to build long-term financial stability.

FIGURE 2 PERSONA 1

Persona 2

Anna Müller works as a marketing manager in Hamburg. Between her job and family life, she's busy most of the time. She doesn't like complicated financial tools, preferring simple solutions that organize everything for her. Anna wants clear notifications about her money so she can avoid overspending, and she appreciates smart suggestions to help her save more effectively.



Anna Müller

Goals
- Easily manage family and personal finances in one place.
- Get reminders for bills and upcoming payments.
- Track spending to save more effectively.
- Get insights and suggestions to improve budgeting.

"I want my money to work for me, not the other way around"

Key characteristics
- Responsible and organized with money.
- Looks for tools that simplify financial planning.
- Appreciates clarity and helpful notifications.

Age: 34
Work: Marketing Manger
Family: Married one child
Location: Hamburg

Bio
Anna is a 34-year-old marketing manager from Hamburg. Between her career and family life, she doesn't have time for complicated financial tools. She needs a clean and smart banking app that lets her track her salary, family expenses, and savings goals without stress. Real-time notifications help her avoid unnecessary spending, and she enjoys getting insights on how to save more efficiently..
.

FIGURE 3 PERSONA 2

2.3 User Stories

User Stories specify software requirements from the perspective of a user of the application. We defined personae, that represent the typical user role. According to those typical user roles, we have described the requirements each user has. For each of the requirements we have defined, there are acceptance criteria, that must be met to fulfil the user's needs.

1. As a user, I want to send money to another IBAN so I can pay individuals or transfer between accounts.
2. As a user, I want to add a description to each transaction so I can remember what it was for.
3. As a user, I want to add/delete/modify multiple bank accounts so I can manage all my finances in one place.
4. As a user, I want to view a combined balance of all accounts, so I understand my total financial position.
5. As a user, I want to switch between accounts easily so I can manage each one individually.
6. As a user, I want to view a list of my income and expenses so I can track my financial activity.
7. As a user, I want to categorize my transactions (e.g., food, transfers, transport) to better understand my spending.
8. As a user, I want to search and filter my transactions.
9. As a user, I want to see graphical reports (e.g. bar charts) to visualize how much I spent.
10. As a user, I want to set a monthly spending limit so I can control my budget.
11. As a user, I want to see the percentage of budget reached, so I can reduce expenses.
12. As a user, I want to receive a warning if I exceed my budget so I can adjust my future spending.
13. As a user, I want to change the account holder.
14. As a user, I want to see the accounts with most expense.
15. As a user, I want to receive a 2% cashback.

2.4 Use Cases

Use Cases describe the interaction of a user with the application to reach a specific goal. It is used to document functionalities from the perspective of the user and consists out of a sequence of actions that take place in a prescribed order. We described our Use Cases with the User Roles and a detailed description.

Fetch total balance

Actor: Logged-in user with linked accounts.

Steps:

1. User opens the app and lands on the Home Page.
2. App fetches:
    a. Balances of all linked accounts.
    b. Recent transactions (income and outcome).

3. App calculates and displays:
    a. Total Balance (sum of all accounts).
    b. Monthly Income and Outcome.
    c. A list of recent transactions with type, amount, and date.

Result:
User sees their overall financial status at a glance via a widget.

MAKE A TRANSACTION

Actor: Logged-in user with linked accounts and sufficient balance.

Steps:

1. User taps the "+" button on the Home Page or Transactions screen.
2. A transaction form opens.
3. User fills in:
    a. Recipient IBAN
    b. Amount
    c. Concept/Description
    d. Selects the account to send money from.
4. User taps "Send".
5. App validates the input and sends a transfer request to the backend.
6. On success, app shows a confirmation, updates the balance, and adds the transaction to the list.

Result:
The transaction is completed and reflected in the user's account and transaction history.

ASSOCIATE ACCOUNT

Actor: Logged-in user.

Steps:

1. User goes to the "Wallet" tab.
2. User taps the "+" button to add an account.
3. A form appears where the user enters:
    a. IBAN
    b. Beneficiary name
    c. BIC
4. User taps "Add Account".
5. App validates the data and sends it to the backend.
6. On success, the account is added to the user's wallet and is visible on the home and wallet screens.

Result:
The new account is linked and ready for balance checks and transactions.

Setting a limit

Actor: Logged-in user managing expenses.

Steps:

1. User navigates to the “Expenses” tab.
2. User taps “Set Limit” or a similar button.
3. User enters a spending limit amount and confirms.
4. App saves the limit and monitors expenses in real time.
5. When user’s expenses exceed the limit, app shows an alert.
6. User sees the alert and can review their spending.

Result: User stays informed about their budget and avoids overspending.

2.5 Requirement Specification

From the definitions of the Personas, User Stories and Use Cases, we defined functional and non-functional requirements. Every requirement consists of a name, ID, description, success criteria and the priority. We grouped them into “Must have requirement” and “May have requirement” to priorities the requirements for the planning and development process

Functional Requirements

Requirement	Registration user
ID	UR-001
Description	The system shall allow new users to register by providing required Information, such as their phone number, and subsequently their email and password. The system must validate the inputs, send a verification code to the phone number, and require users to complete their username and password
Success Criteria	Registration is successful only when the user provides valid inputs; the system sends a verification code which the user enters on another screen;
Priority	Must Have

TABLE 1 FUNCTIONAL REQUIREMENT: REGISTRATION USER

Requirement	User login
ID	UL-001
Description	The system shall allow registered users to log in using their phone number, or their username and password. The system must validate credentials, provide error messages for invalid login attempts, support password recovery, and maintain session security during the user’s activity.
Success Criteria	Login is successful only with valid credentials; incorrect credentials show an error message; users can recover passwords; sessions remain secure and active until logout or timeout.
Priority	Must Have

TABLE 2 FUNCTIONAL REQUIREMENT: USER LOGIN

Requirement	Associating an account
ID	UR-003
Description	The system shall allow users to associate external bank accounts by entering the IBAN, beneficiary name, and BIC. The system must validate the entered data, confirm successful association, and display the new account in the user's wallet.
Success Criteria	Users can successfully add external accounts with valid IBAN, beneficiary, and BIC; invalid inputs are rejected with error messages; associated accounts appear in the wallet for management.
Priority	Must Have

TABLE 3 FUNCTIONAL REQUIREMENT: ASSOCIATING AN ACCOUNT

Requirement	Changing personal info
ID	UR-001
Description	The system shall allow users to update their personal information such as their first or last name. The system must validate the new data and save the changes successfully.
Success Criteria	Users can edit their personal info (e.g., name), changes are validated and saved, and the updated information is reflected in their profile.
Priority	Must Have

TABLE 4 FUNCTIONAL REQUIREMENT: CHANGE PERSONAL INFORMATION

Requirement	Registration user with phone number
ID	UR-002
Description	The system shall allow users to register with their email and password, and optionally enter their phone number during registration. The phone number field is not mandatory and can be skipped.
Success Criteria	Users can complete registration without providing a phone number; if a phone number is entered, it is validated and saved correctly.
Priority	Optional

TABLE 5 FUNCTIONAL REQUIREMENT: REGISTER USER WITH PHONE NUMBER

Requirement	Visualizing transactions
ID	TR-001
Description	The system shall allow users to view a list of all transactions (income and outcome) made from their associated accounts, including details like amount, date, concept, and account used.
Success Criteria	Users can access a transaction history screen that displays accurate and complete information for each transaction, sorted by date and updated in real time.
Priority	Must-have

TABLE 6 FUNCTIONAL REQUIREMENT: VISUALIZING TRANSACTIONS

Requirement	Fetch total balance
ID	TR-001
Description	The system shall calculate and display the total balance by summing the balances of all associated accounts. This information shall be shown on the home screen in a widget.
Success Criteria	The user sees a total balance on the home screen that accurately reflects the sum of all linked account balances, updated in real time or on refresh.
Priority	Must-have

TABLE 7 FUNCTIONAL REQUIREMENT: FETCH TOTAL BALANCE

Requirement	Create transaction
ID	TX-001
Description	The system shall allow users to create a new transaction by entering the recipient's IBAN, amount, concept, and selecting the source account.
Success Criteria	A transaction is successfully created when valid data is entered; funds are deducted from the selected account, and the transaction appears in the history.
Priority	Must-have

TABLE 8 FUNCTIONAL REQUIREMENT: CREATE TRANSACTION

Requirement	Expense alert
ID	EA-001
Description	The system shall allow users to set a monthly spending limit and notify them when their expenses exceed this limit, including how much they went over.
Success Criteria	The user can define a limit in the Expenses tab; when the total spending exceeds it, a notification is triggered showing the exceeded amount.
Priority	Must-have

TABLE 9 FUNCTIONAL REQUIREMENT: EXPENSE ALERT

NON-FUNCTIONAL REQUIREMENTS

Requirement	Performance Response Time

ID	NFR-001
Description	The system shall respond to user actions (e.g., login, viewing balance) within 2 seconds.
Success Criteria	95% of user interactions complete in ≤2 seconds under normal network conditions.
Priority	Must-have

TABLE 10 NON-FUNCTIONAL REQUIREMENT: PERFORMANCE RESPONSE TIME

Requirement	System Availability (In production)
ID	NFR-002
Description	The system shall be available 99.9% of the time, excluding scheduled maintenance.
Success Criteria	Uptime is monitored monthly and does not fall below 99.9%.
Priority	Must-have

TABLE 11 NON-FUNCTIONAL REQUIREMENT: SYSTEM AVAILABILITY (IN PRODUCTION)

Requirement	Data Security (In production)
ID	NFR-004
Description	All user data shall be encrypted in transit and at rest.
Success Criteria	Communications use HTTPS; database uses encryption protocols for storage.
Priority	Must-have

TABLE 12 NON-FUNCTIONAL REQUIREMENT: DATA SECURITY (IN PRODUCTION)

Requirement	Platform Compatibility
ID	NFR-005
Description	The mobile app shall be compatible with Android 10+ and iOS 13+.
Success Criteria	The app runs without major bugs on supported OS versions and passes store certification.

Priority	Must-have

TABLE 13 NON-FUNCTIONAL REQUIREMENT: PLATFORM COMPATIBILITY

Requirement	Scalability (In production)
ID	NFR-006
Description	The app shall support at least 10,000 concurrent users without performance degradation.
Success Criteria	The app shall support at least 10,000 concurrent users without performance degradation.
Priority	Must-have

TABLE 14 NON-FUNCTIONAL REQUIREMENT: SCALABILITY (IN PRODUCTION)

Requirement	Fluent Navigation
ID	NFR-007
Description	The user will be able to navigate fluently through the user interface, even at his first time using the app
Success Criteria	The user-experience is positive
Priority	Must-have

TABLE 15 NON-FUNCTIONAL REQUIREMENT: FLUENT NAVIGATION

3. Conceptual model

The basis of our development is the conceptual model. It was created to describe and visualize the application and its functions on paper. Different diagrams and especially Mock-up's were used for this step.

3.1 Sequence Diagram

Create User

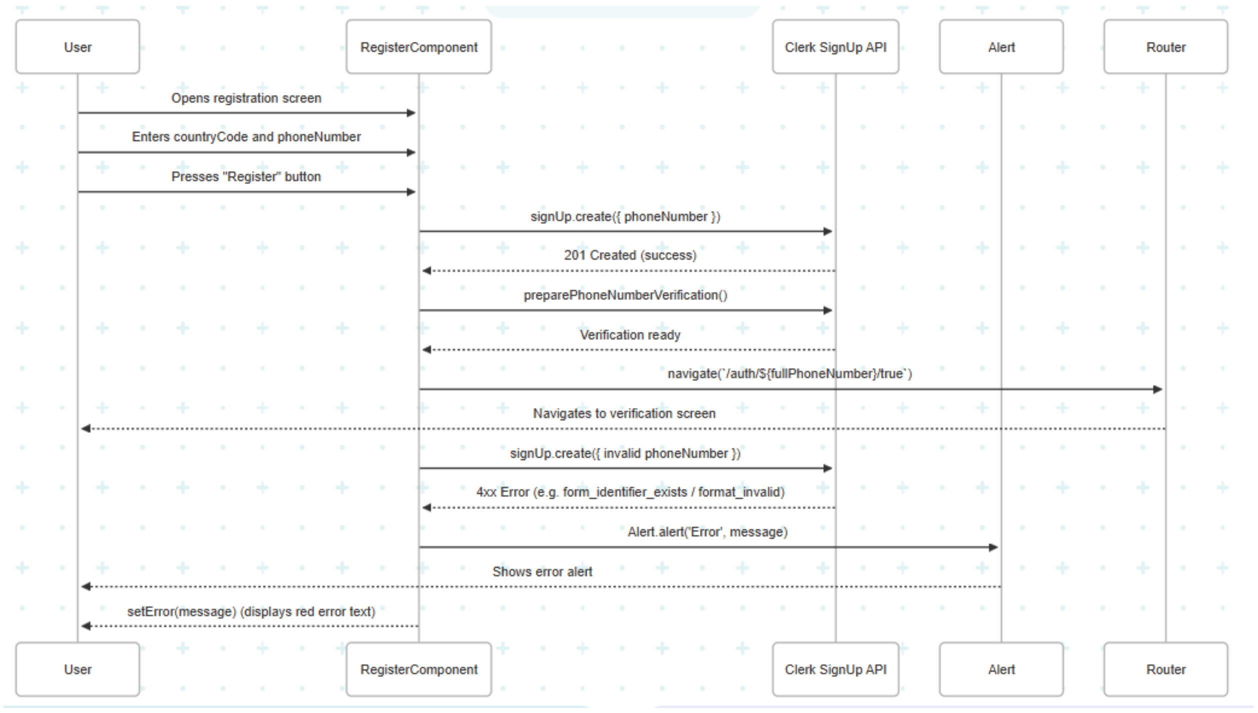

User
RegisterComponent
Clerk SignUp API
Alert
Router
Opens registration screen
Enters countryCode and phoneNumber
Presses "Register" button
signUp.create({ phoneNumber })
201 Created (success)
preparePhoneNumberVerification()
Verification ready
navigate(`/auth/${fullPhoneNumber}/true`)
Navigates to verification screen
signUp.create({ invalid phoneNumber })
4xx Error (e.g. form_identifier_exists / format_invalid)
Alert.alert('Error', message)
Shows error alert
setError(message) (displays red error text)
User
RegisterComponent
Clerk SignUp API
Alert
Router

FIGURE 4 SEQUENCE DIAGRAM: CREATE USER

LOGIN USER

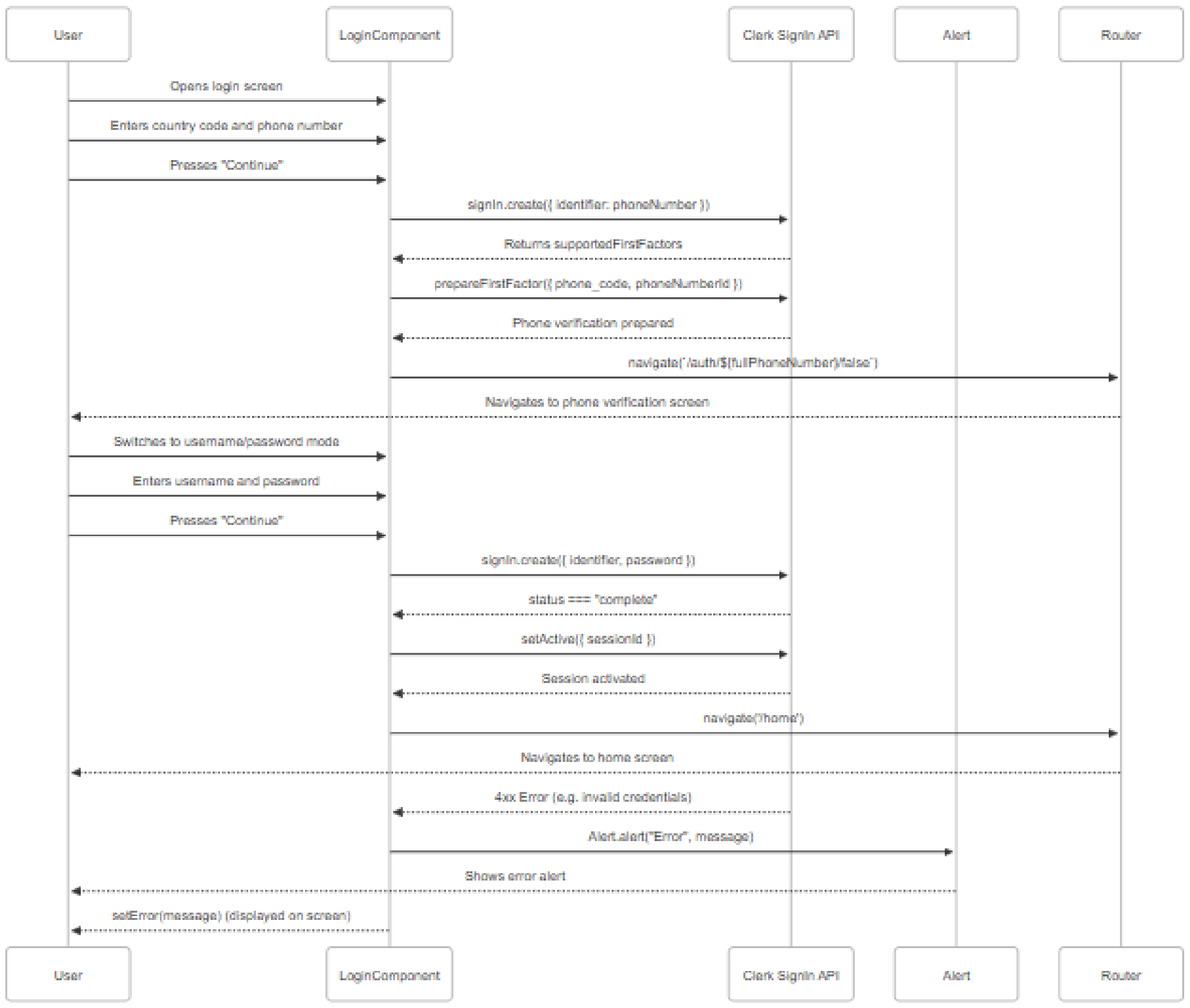

User
LoginComponent
Clerk SignIn API
Alert
Router
Opens login screen
Enters country code and phone number
Presses "Continue"
signIn.create({ identifier: phoneNumber })
Returns supportedFirstFactors
prepareFirstFactor({ phone_code, phoneNumberId })
Phone verification prepared
navigate(`/auth/${fullPhoneNumber}/false`)
Navigates to phone verification screen
Switches to username/password mode
Enters username and password
Presses "Continue"
signIn.create({ identifier, password })
status === "complete"
setActive({ sessionId })
Session activated
navigate('/home')
Navigates to home screen
4xx Error (e.g. invalid credentials)
Alert.alert("Error", message)
Shows error alert
setError(message) (displayed on screen)
User
LoginComponent
Clerk SignIn API
Alert
Router

FIGURE 5 SEQUENCE DIAGRAM: LOGIN USER

CREATE TRANSACTION

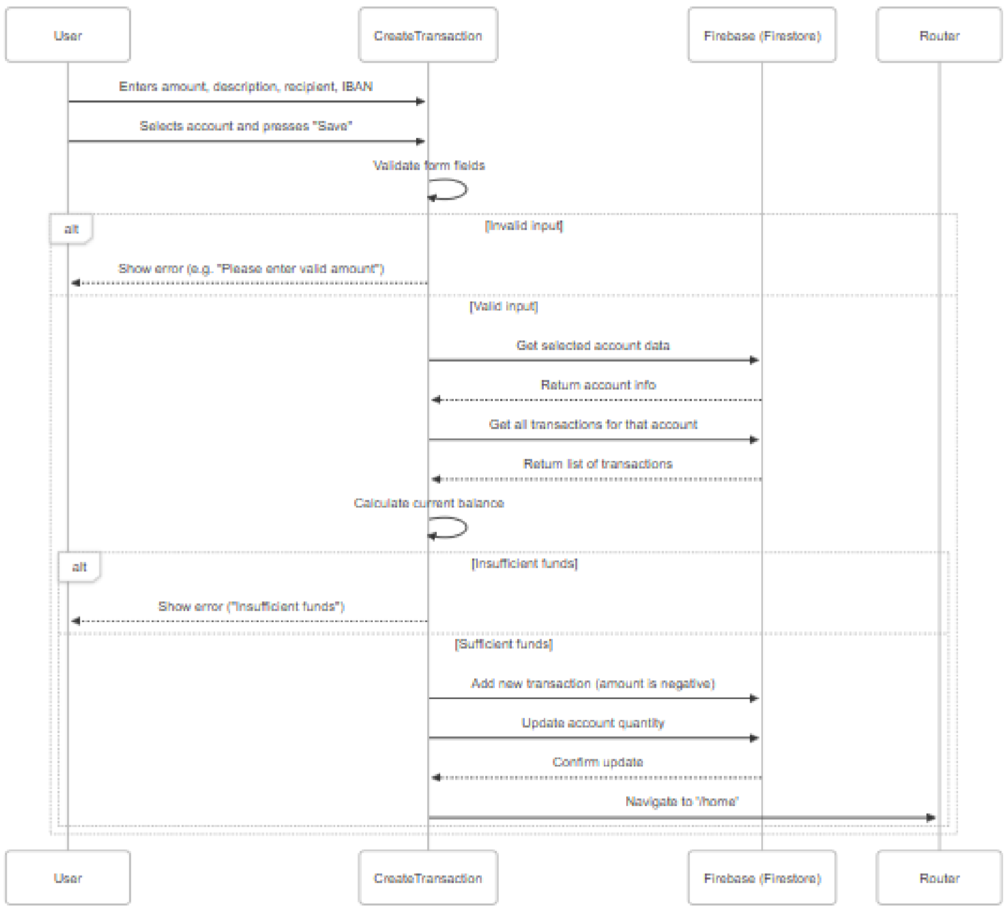

User
CreateTransaction
Firebase (Firestore)
Router
Enters amount, description, recipient, IBAN
Selects account and presses "Save"
Validate form fields
alt
[Invalid input]
Show error (e.g. "Please enter valid amount")
[Valid input]
Get selected account data
Return account info
Get all transactions for that account
Return list of transactions
Calculate current balance
alt
[Insufficient funds]
Show error ("Insufficient funds")
[Sufficient funds]
Add new transaction (amount is negative)
Update account quantity
Confirm update
Navigate to '/home'

FIGURE 6 SEQUENCE DIAGRAM: CREATE TRANSACTION

Associate account

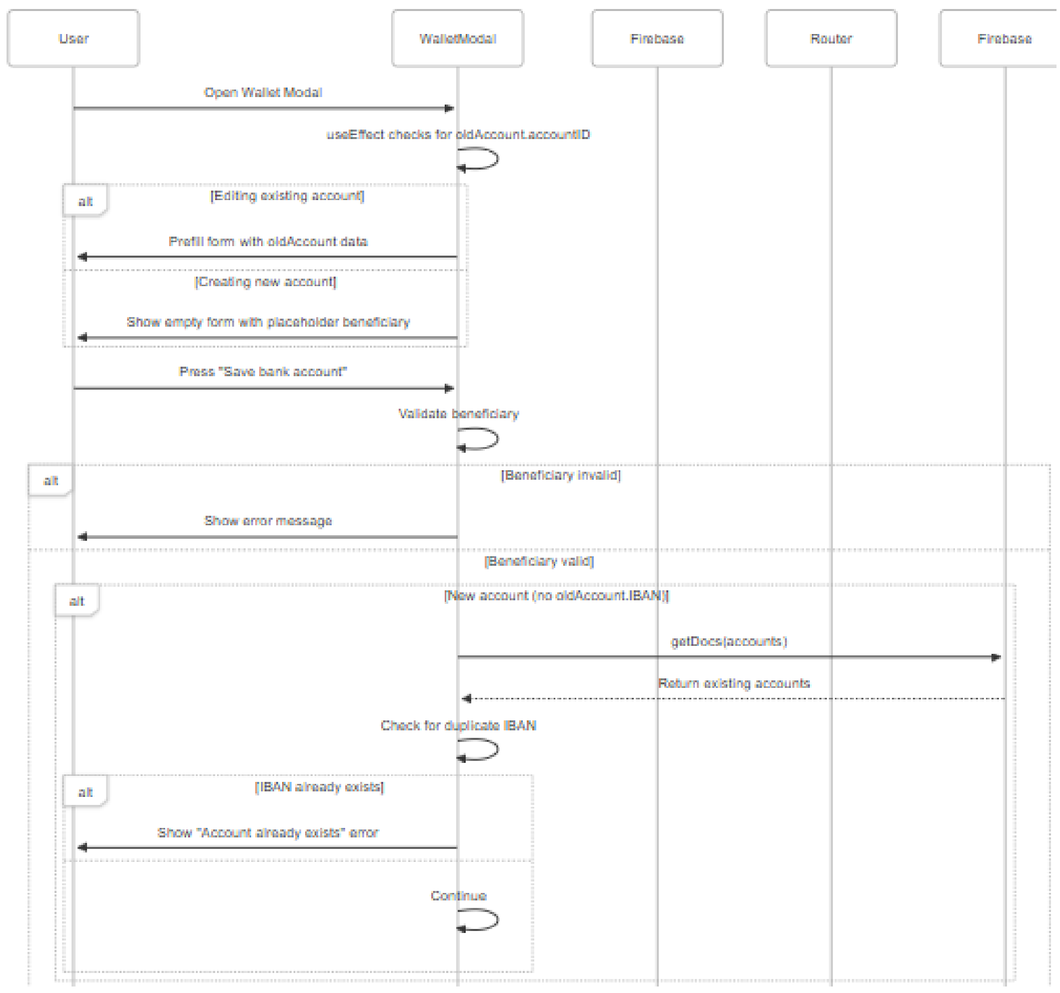


FIGURE 7 SEQUENCE DIAGRAM: ASSOCIATE ACCOUNT

3.2 States Diagram

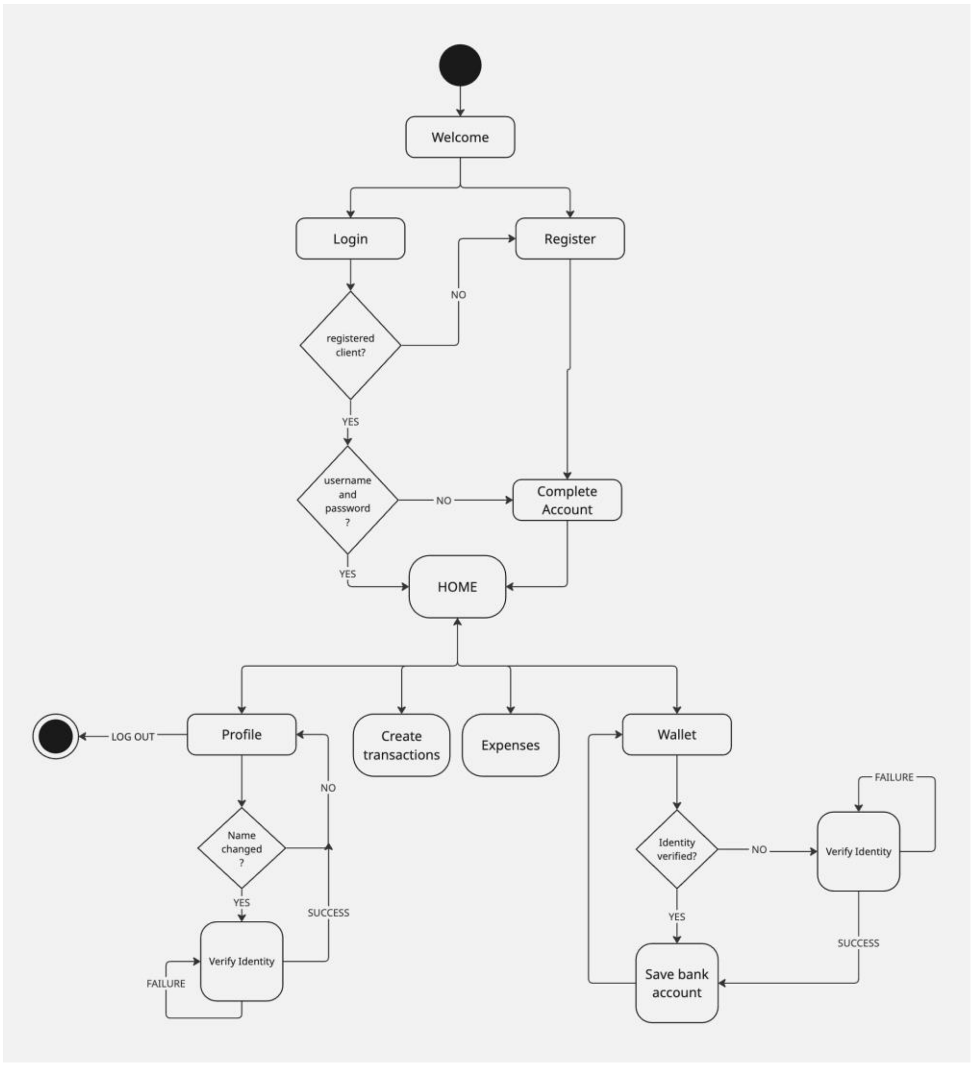


FIGURE 8 STATES DIAGRAM

4. Software Architecture

UniWallet is built using React Native, a JavaScript framework developed by Meta for creating cross-platform mobile applications on iOS and Android from a single codebase. React Native enables the development of high-performance native apps while leveraging the familiar syntax and structure of React.

To streamline development and reduce the need for complex native configuration, the app utilizes Expo, a robust platform that simplifies the React Native development process. Notably, the project makes extensive use of expo-router, a file-based routing system built on top of react-navigation. This approach enhances scalability and simplifies navigation logic, especially for implementing deep linking and nested routes.

For user management, Clerk—a modern authentication and user management solution for web and mobile applications—plays a central role in the app. It is seamlessly integrated with Firestore, which functions as the primary backend for securely storing and managing user-inherited data like transactions or bank accounts.

4.1 Architecture Pattern

The application follows a single-tier architecture based on a serverless model using Backend-as-a-Service (BaaS) through Firebase Firestore. The React Native frontend interacts directly with Firestore using the Firebase software development kit (SDK), eliminating the need for a traditional backend. This approach was chosen to simplify development, reduce infrastructure overhead, and take full advantage of Firebase's scalable and real-time capabilities. State management within the app is handled using React's built-in hooks (useState, useEffect), while user authentication is managed through Clerk Authentication, ensuring secure access without additional backend logic.

This architecture offers a lightweight yet powerful solution ideal for mobile apps that benefit from rapid development and cloud services.

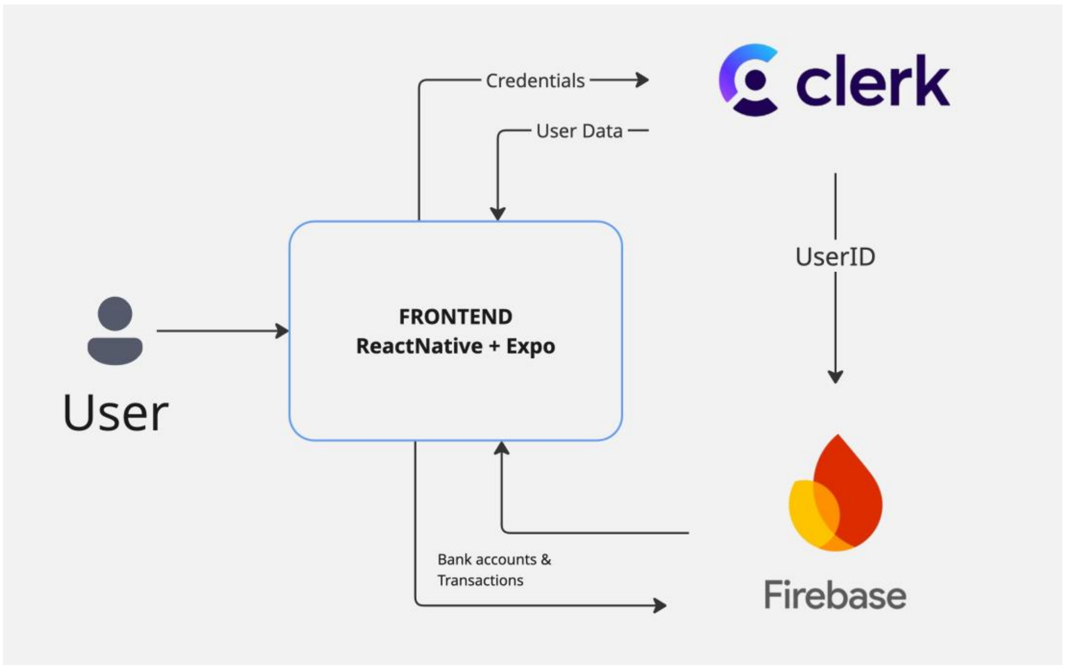


FIGURE 9 ARCHITECTURE PATTERN

4.2 Dependencies

Dependency	Version	Description
@clerk/clerk-expo	^2.9.6	Expo SDK integration for Clerk authentication.
date-fns	^4.1.0	Modern, lightweight library for parsing, formatting, and manipulating dates in JS.
expo	~53.0.0	Meta-package providing the core Expo SDK (build tools, APIs, and CLI).
expo-asset	~11.1.5	Manages and serves static assets (images, fonts, etc.) in Expo apps.
expo-av	~15.1.4	Audio & video playback and recording utilities for Expo.
expo-blur	~14.1.4	Provides a native "blur" view (Frosted-glass effect) on Android and

		iOS.
expo-constants	~17.1.6	Access to system constants (app manifest, device info, platform, etc.)
expo-linking	~7.1.4	Utilities for handling deep links and URL schemes in Expo apps.
expo-local-authentication	~16.0.4	Biometric (Face ID / Touch ID) and device-PIN authentication support.
expo-router	~5.0.6	File-based routing solution for Expo and React Native projects.
expo-secure-store	~14.2.3	Secure, encrypted key–value storage for sensitive data (tokens, credentials).
expo-splash-screen	~0.30.8	Control over the native splash screen display and hiding behavior.
expo-status-bar	~2.2.3	Simplified API for controlling the status bar style and visibility.
express	^4.21.2	Minimalist web framework for building RESTful APIs and web servers in Node.js.
firebase	^11.6.0	Official Firebase JS SDK for Firestore, Auth, Storage, Analytics, and more.
react	19.0.0	Core React library for building user interfaces.
react-dom	19.0.0	React package for working with the DOM (web), needed for react-native-web.
react-native	0.79.2	Core React Native framework for building native mobile apps.
react-native-circular-progress	^1.4.1	Component for rendering animated circular progress indicators.
react-native-confirmation-code-field	^7.4.0	Input field component optimized for one-time code / OTP entry.
react-native-gesture-handler	~2.24.0	Low-level gesture handling and touch interactions for React Native.
react-native-reanimated	~3.17.4	Highly performant animations and gesture-driven interactions.
react-native-safe-area-context	5.4.0	Provides insets for handling iPhone notch, Android status bar, and other safe areas.
react-native-size-matters	^0.4.2	Utility to scale sizes (margins, fonts) based on screen dimensions for responsive design.
react-native-web	^0.20.0	Makes React Native components renderable in web browsers.

TABLE 16 DEPENDENCIES

4.3 DATABASE

We decided to use Firestore, a NoSQL real-time database developed by Google as part of Firebase. It is highly scalable, flexible, and commonly used for web and mobile applications. Instead of using tables and rows like in SQL databases, Firestore organizes data into collections and documents.

It works together with Clerk users' management, which provides the userID from the useUser hook. Thus, this userID is stored in Firestore, more specifically in the Users collection, which contains another nested collection called accounts, which hold the information of the bank account of each corresponding user. Finally, these nested collections contain at the same time other sub-collections called transactions, which provide information about the movements of each corresponding account, corresponding to the specific user. This is described in the diagram below:

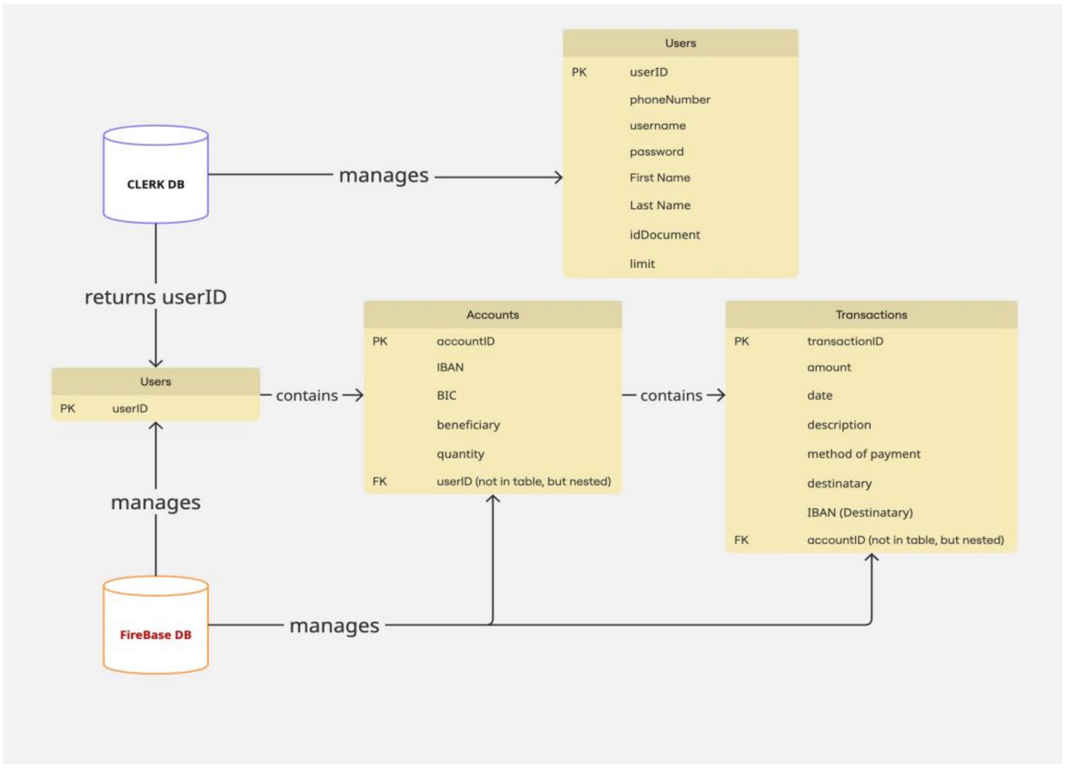


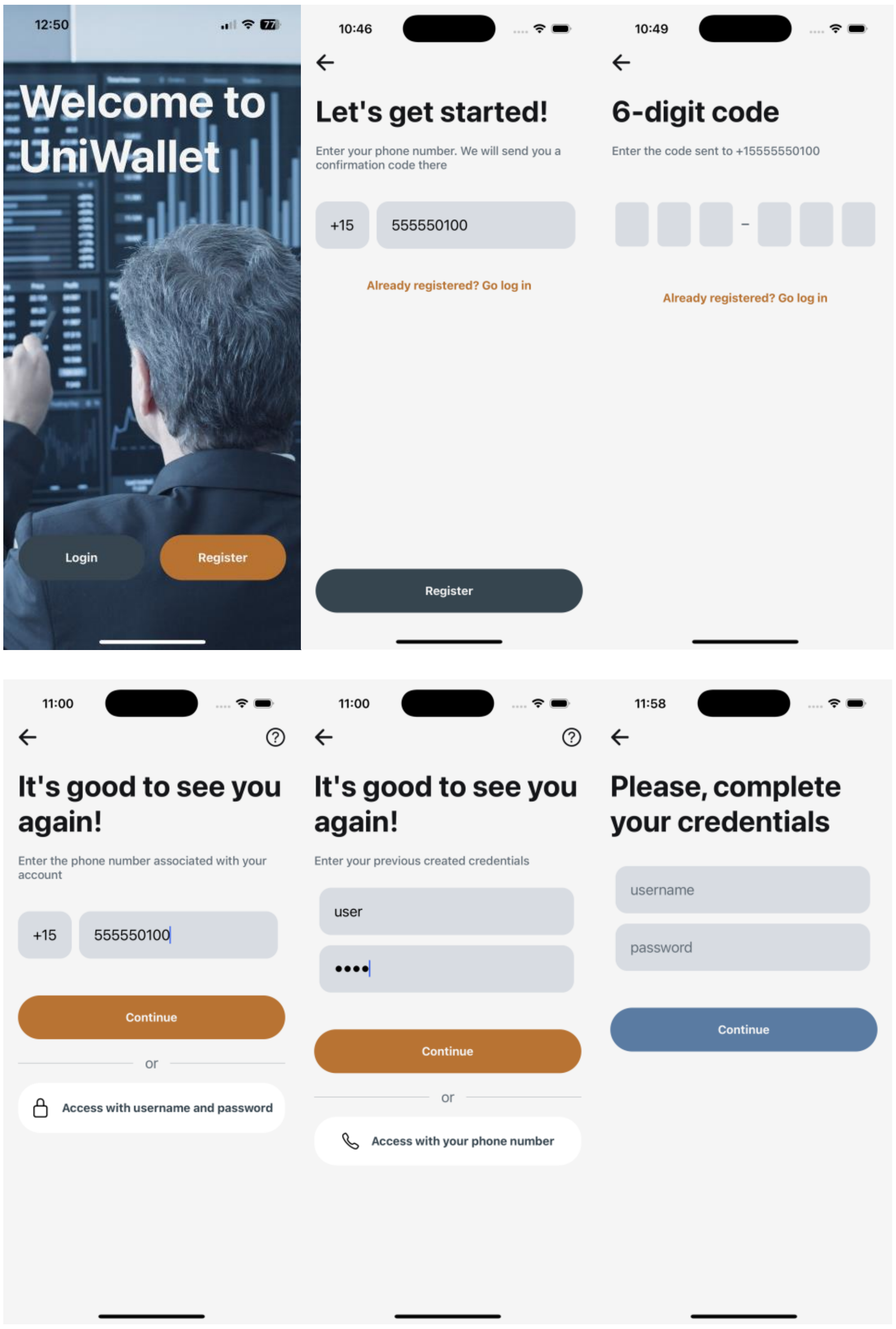
FIGURE 10 DATA MODELING

5. Result

5.1 Features

Register and Login process

First of all, user is presented a welcome page, where he can register or login. User registers with his phone number, to which a temporal code will be sent. On the other hand, user can login with his phone number, or with a username and password he is required to fulfil after registering.



12:50

Welcome to UniWallet

Login
Register

10:46

Let's get started!

Enter your phone number. We will send you a confirmation code there

+15
555550100

Already registered? Go log in

Register

10:49

6-digit code

Enter the code sent to +15555550100

Already registered? Go log in

11:00

It's good to see you again!

Enter the phone number associated with your account

+15
555550100

Continue

or

Access with username and password

11:00

It's good to see you again!

Enter your previous created credentials

user
••••

Continue

or

Access with your phone number

11:58

Please, complete your credentials

username
password

Continue

FIGURE 11 SCREEN: WELCOME, REGISTER AND LOGIN

ADD/DELETE/MODIFY BANK ACCOUNTS FROM THE WALLET

The total balance will show the sum of the balance through all accounts. Once user identity is verified, he can add a bank account by pressing the orange round button, when he is redirected to the wallet modal, where he can fulfil bank account data. Then, by pressing an existing account, user can delete it or modify its BIC or beneficiary.

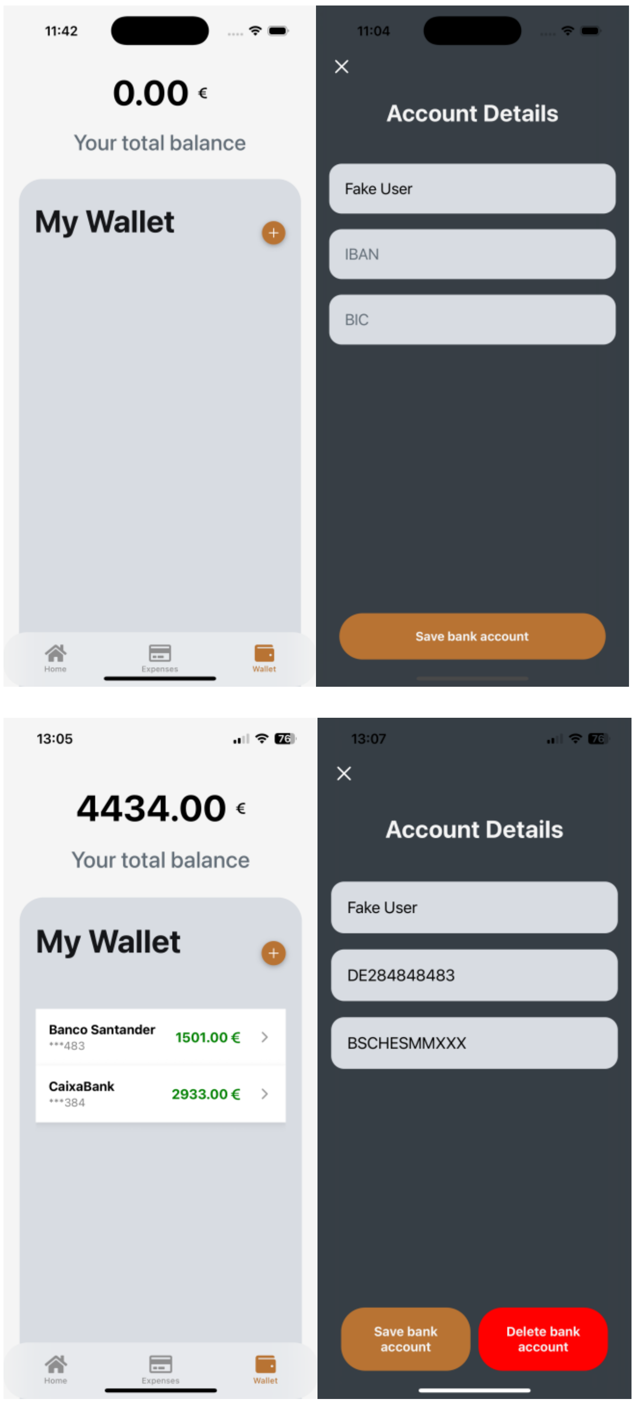


FIGURE 12 SCREEN: ADD/DELETE/MODIFY BANK ACCOUNTS

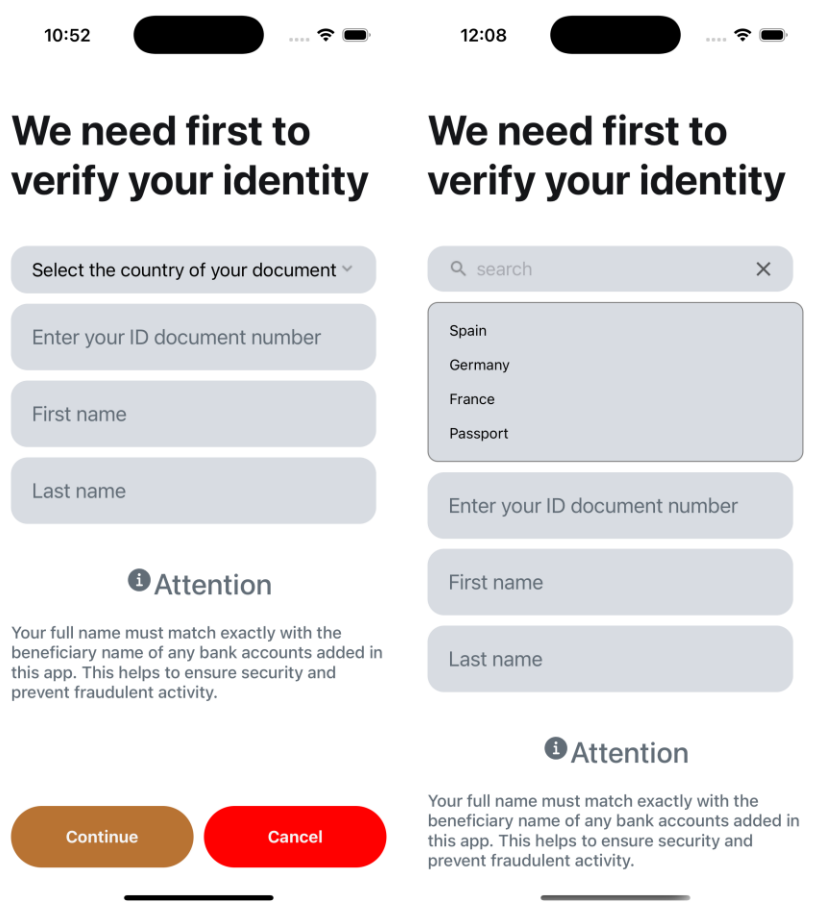
Verify User Identity

As mentioned above, user must verify his identity just once before adding a bank account, but also every time he changes his first or last name in the profile modal.

User can select the origin of his documentation from the three countries we are currently working with, or for a more general verification, the passport option. Each country adjusts to a different regulation for their documentation; thus, the corresponding check will be done.

There is an attention message which describes the restriction applied to the full name given to verify: it must match the beneficiary of all accounts inside the app.

This is for ensuring security and preventing fraudulent activity.


10:52
We need first to verify your identity
Select the country of your document
Enter your ID document number
First name
Last name
Attention
Continue
Cancel
12:08
We need first to verify your identity
search
Spain
Germany
France
Passport
Enter your ID document number
First name
Last name
Attention

Your full name must match exactly with the beneficiary name of any bank accounts added in this app. This helps to ensure security and prevent fraudulent activity.

Your full name must match exactly with the beneficiary name of any bank accounts added in this app. This helps to ensure security and prevent fraudulent activity.

FIGURE 13 SCREEN: VERIFY IDENTITY

CHECK/EDIT PROFILE AND LOG OUT

User can check his profile, where his full name will be fetched, besides another content rows such as Consents and rights, Privacy policy, Learn and Inbox. At the bottom, there is a logout button for finishing the user session.

Moreover, user can modify his first or last name, taking into account he must verify his new identity then.

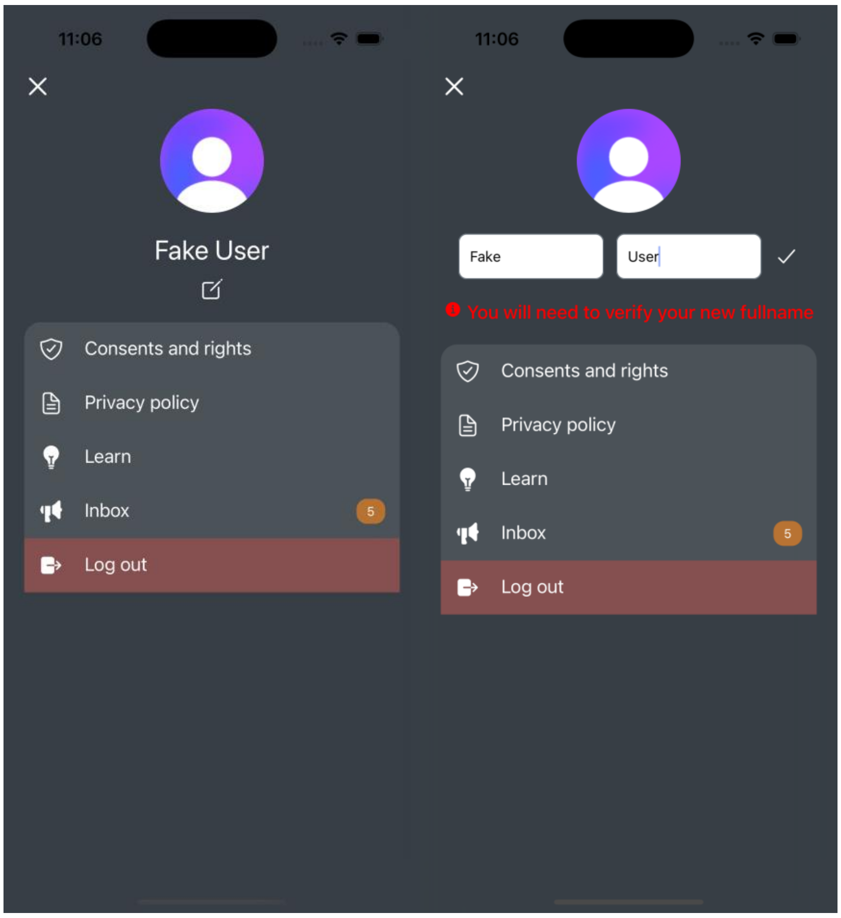
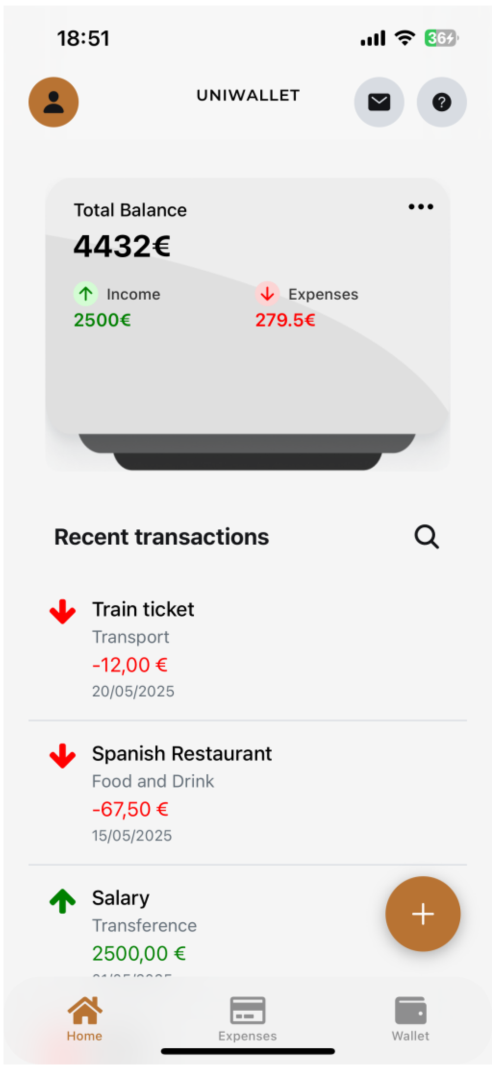


FIGURE 14 SCREEN: PROFILE

HOME

The user is presented the total balance from all his accounts, considering incomes and expenses. Besides, there is a section of transactions sorted by the earliest date, where the user can check his more recent expenses or incomes, as well as the category of the transaction, its amount, and its date of operation.

Moreover, it is possible to execute several operations from this tab like going to the profile modal, help, or correspondences. Also, users can send money by pressing the button with the '+' icon.



Total Balance
4432€
Income 2500€
Expenses 279.5€

Recent transactions

Train ticket
Transport
-12,00 €
20/05/2025

Spanish Restaurant
Food and Drink
-67,50 €
15/05/2025

Salary
Transference
2500,00 €

FIGURE 15 SCREEN: HOME

SEND MONEY

In this modal, the user can send money by fulfilling the required data and selecting the source account between the ones associated with him. This will create a transaction with the predefined category 'Transference', which will substract the selected amount from the source account.

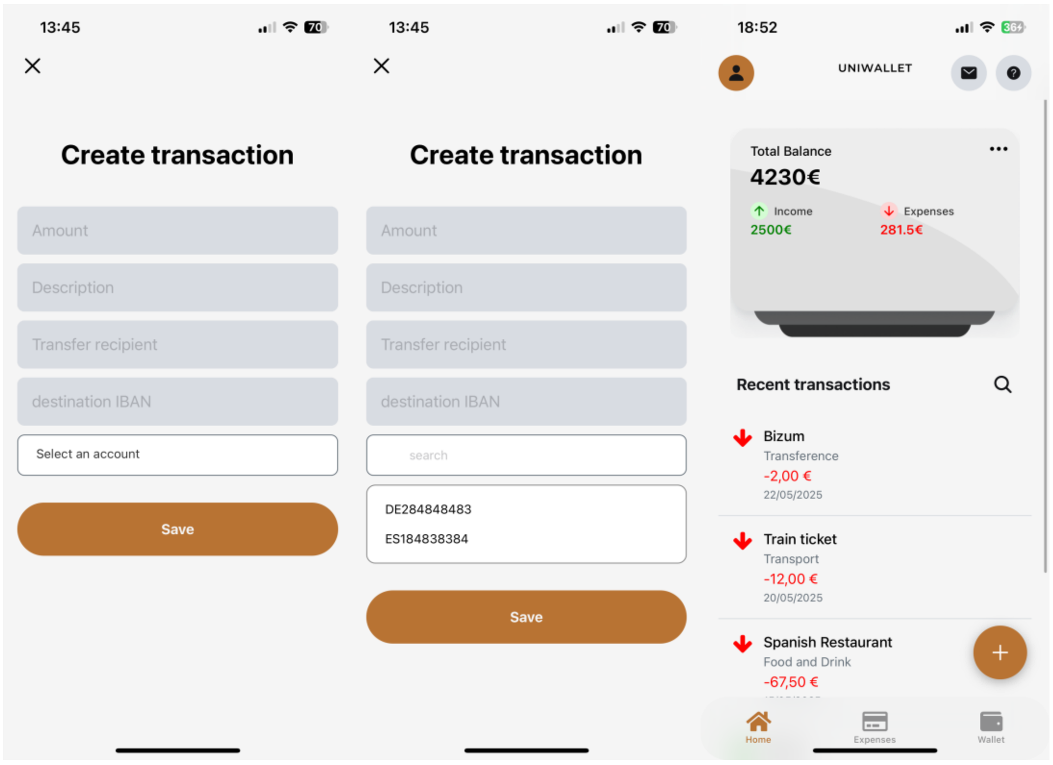
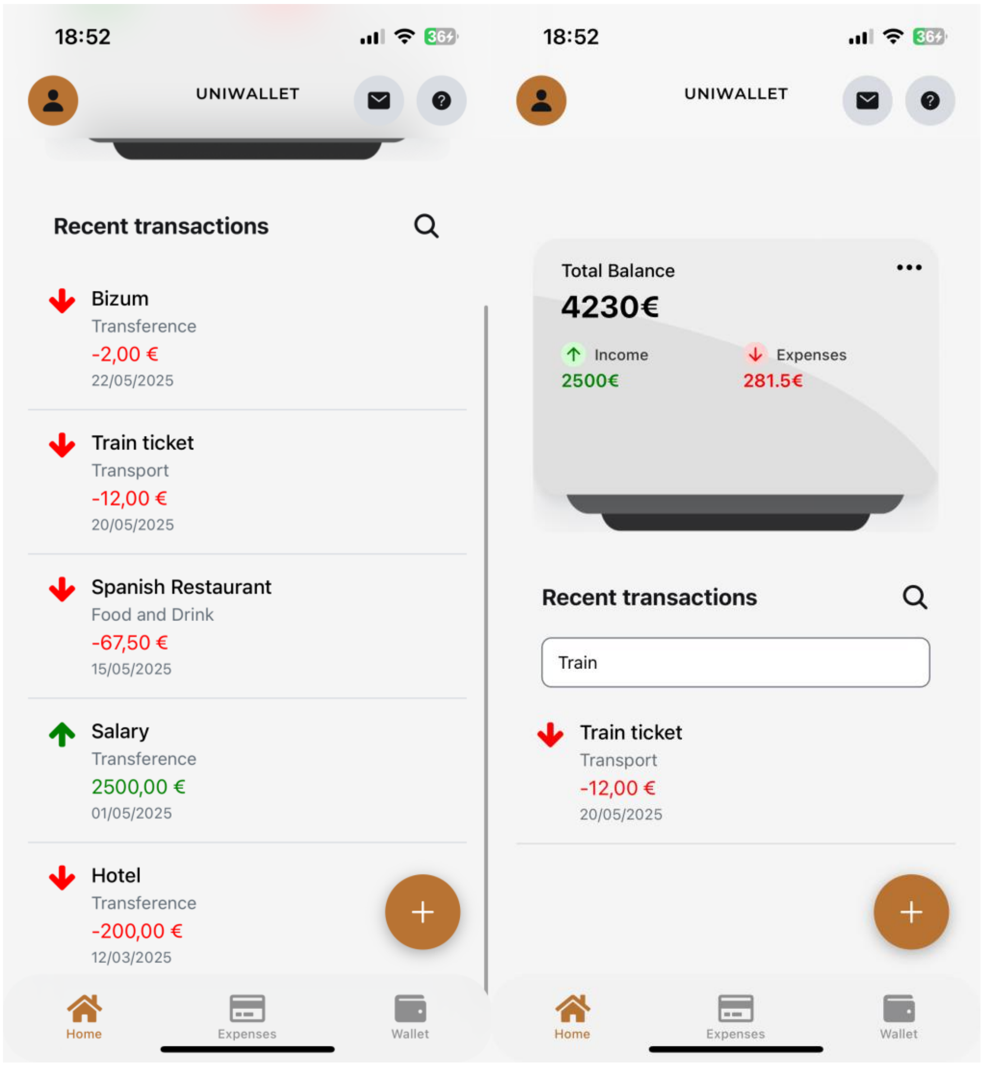


FIGURE 16 SCREEN: SEND MONEY

SEARCH TRANSACTIONS

Inside the home page, user can search into his transactions by pressing the magnifying glass icon, which will display an input text for filtering transactions by this input value.


18:52
UNIWALLET
Recent transactions
Bizum
Transference
-2,00 €
22/05/2025
Train ticket
Transport
-12,00 €
20/05/2025
Spanish Restaurant
Food and Drink
-67,50 €
15/05/2025
Salary
Transference
2500,00 €
01/05/2025
Hotel
Transference
-200,00 €
12/03/2025

18:52
UNIWALLET
Total Balance
4230€
Income
2500€
Expenses
281.5€
Recent transactions
Train
Train ticket
Transport
-12,00 €
20/05/2025

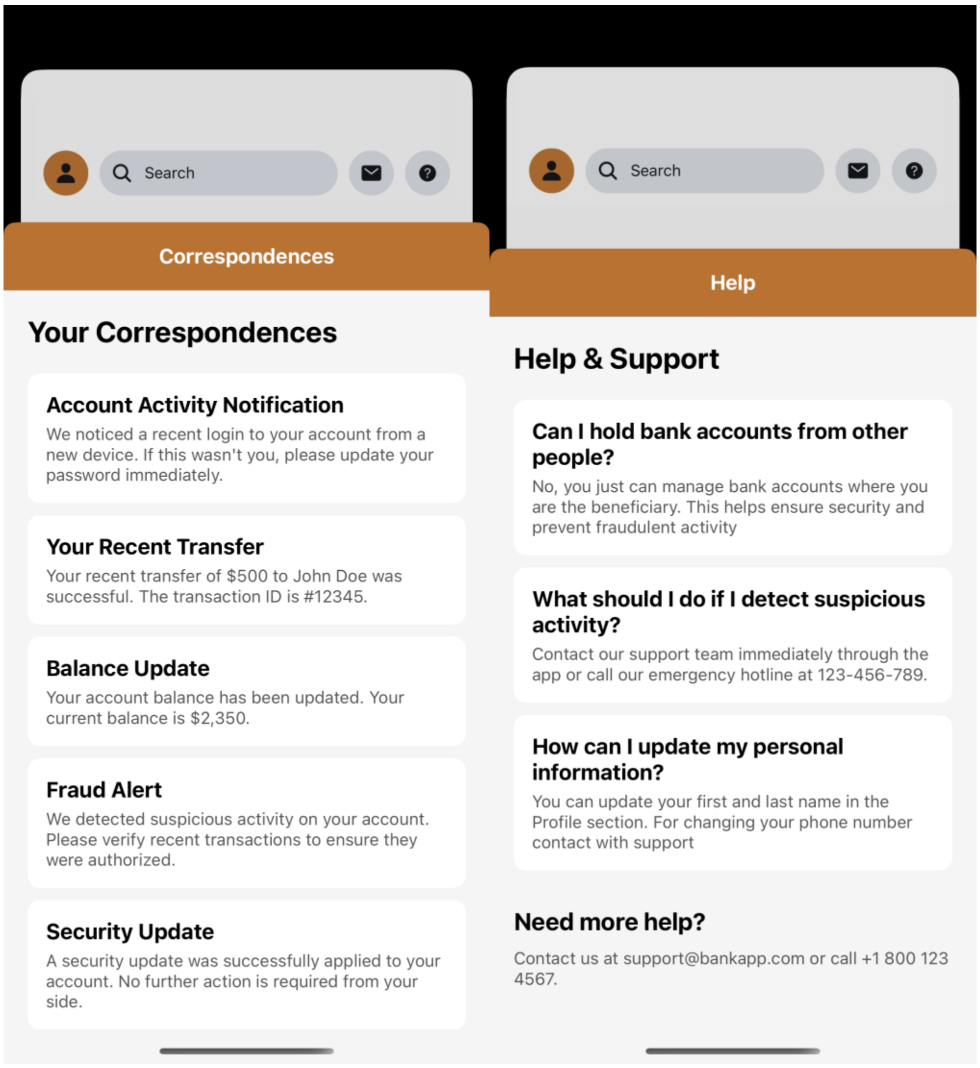
FIGURE 17 SCREEN: SEARCH INTO TRANSACTIONS

Ask for Help and check Correspondences

From the same header as previous point, user can check the profile, help and correspondences windows, by clicking their corresponding icons.

In Correspondences, the user is communicated with the most relevant up to date information regarding his account.

Furthermore, in Help, there are some answers to the most frequent questions, and a guide of how to contact support.



Your Correspondences

Account Activity Notification
We noticed a recent login to your account from a new device. If this wasn't you, please update your password immediately.

Your Recent Transfer
Your recent transfer of $500 to John Doe was successful. The transaction ID is #12345.

Balance Update
Your account balance has been updated. Your current balance is $2,350.

Fraud Alert
We detected suspicious activity on your account. Please verify recent transactions to ensure they were authorized.

Security Update
A security update was successfully applied to your account. No further action is required from your side.

Help & Support

Can I hold bank accounts from other people?
No, you just can manage bank accounts where you are the beneficiary. This helps ensure security and prevent fraudulent activity

What should I do if I detect suspicious activity?
Contact our support team immediately through the app or call our emergency hotline at 123-456-789.

How can I update my personal information?
You can update your first and last name in the Profile section. For changing your phone number contact with support

Need more help?
Contact us at support@bankapp.com or call +1 800 123 4567.

FIGURE 18 SCREEN: CORRESPONDENCES AND HELP

FETCH EXPENSES

Monthly expenses from all the attached bank accounts will be shown in this window. The information is organized into four distinct widgets, each serving a specific purpose:

- **Total Spending:** Displays the total amount spent during the selected month.
- **Cashback Overview:** Shows the amount of cashback earned, calculated in accordance with the applicable European cashback.
- **Top Spending Categories:** Highlights the three categories in which the user has spent the most, along with the expenditure.
- **Top Spending Accounts:** Presents the three bank accounts with the highest spending during the month, detailing the amount spent.

In order to improve the user experience, the order of these widgets can be altered by the user by long-pressing them and dropping them into the preferred position.

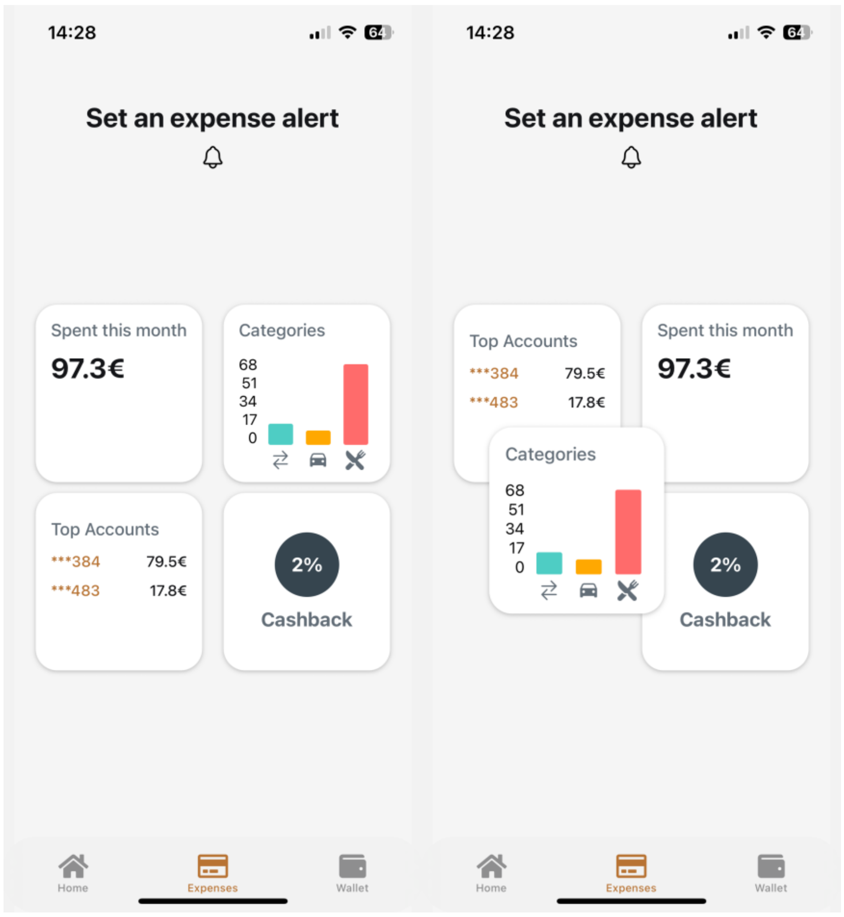


FIGURE 19 SCREEN: EXPENSES DISPLAYMENT

Set Expense Alert

In the same screen as the previous point, user can set an expense alert by clicking the text or icon at the top.

1. He is asked to insert a limit budget.
2. It is shown the current percentage of the total limit spent.
3. User can modify the fixed budget by clicking the 'modify button'
4. If the current quantity spent exceeds the fixed budget, an alert will be displayed, so that the user is aware of his economic activity.

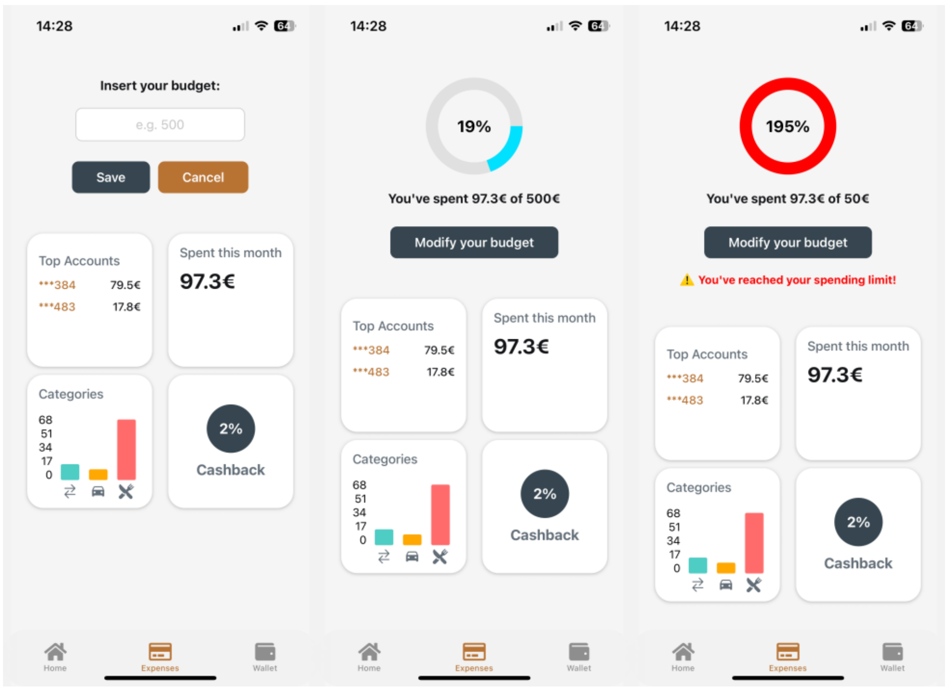


FIGURE 20 SCREEN: EXPENSE ALERT

5.2 Restrictions and Validations

For ensuring the app behaves logically and remains secure, there are some restrictions and validations that need to be applied.

- Users related: Users must provide a both valid and unique phone number to sign up, always maintaining one on their account; Usernames must be unique and contain from 4 to 64 characters; Passwords must contain 8 or more characters, and they are rejected if compromised, powered by HaveIBeenPwned; Identity must be verified if first name or last name change; Identity document must coincide with the selected country regulations.

- Accounts related: The identity of the user must be verified before adding any bank account; 5 accounts at most per user; Beneficiary of all bank accounts added must match the verified user´s full name; It will not be possible to add an account with an already registered IBAN; When modifying a bank account, the IBAN must remain the same as in the origin; Currency is always the euro.

- Expenses related: Expenses tab just refers to expenses issued in the current calendar month (1st – 31st); Widgets show at most the 3 accounts/categories with the biggest associated expenses; There are 4 possible categories: “Food and Drink”, “Transport”, “Transference”, “default”.

- Transactions creation related: Amount must be a positive number; Source account must match any of the ones associated with the user; Source account must contain more money than the selected amount of the transaction

5.3 Testing

To verify that the core functionality of our app meets the specified requirements, we carried out a combination of manual and automated evaluation steps:

- **Exception-Driven Debugging:**Throughout development, we wrapped all asynchronous calls (Firebase reads/writes, Clerk user lookups, navigation hooks) in try…catch blocks.
  Whenever an exception was thrown—whether due to network timeouts, missing user IDs, or malformed data—we logged both the error message and stack trace to the console.
  By iteratively reproducing failure scenarios (e.g. navigating away mid-fetch, simulating permission denials) and inspecting the console output, we were able to pinpoint and correct edge-case bugs in our data-fetching and state-update logic.

- **Console-Based Trace Testing:**We added informative console statements at key lifecycle events: on focus/blur of each screen, before and after each Firestore query, and around our formatting routines.

Besides, for testing Clerk functionality, we used the credentials provided by this solution:

- Test verification code: 424242
- Test phone number: +15 5555501xx

6. Conclusion

We successfully delivered a first-version multi–bank accounts app that centralizes users' financial data in one place. At its core, the application allows authenticated users to view the aggregated balance across all their bank accounts and to browse, search, and filter recent transactions. While these basic functions meet our initial objectives, there remains substantial work to transform this prototype into a fully polished product that satisfies real-world user needs.

As a team, we gained significant practical experience with modern mobile-app tooling and patterns such as NoSQL databases (Firestore), React Native's component model, Expo routing and Clerk's Expo SDK for secure, client-side user management.

Working collaboratively on this project also sharpened our soft skills. We adhered to Git-based workflows, carefully structured pull requests, and held weekly stand-ups to align on progress and roadblocks. Consistent communication helped us avoid merge conflicts and enabled rapid iteration on UI/UX feedback.

In summary, this project was a highly valuable learning experience. We achieved all core functionality and laid a strong foundation for future enhancements. With further refactoring for maintainability, expanded automated testing, and deeper UX polish, the app can mature into a robust tool that empowers users to manage multiple real bank accounts effortlessly.

6.1 Limitations

This first version of the app has some limitations, which should be considered in future development versions.

First, user identity verification is only implemented locally, which restricts the app's use in regulated financial environments. This is because it would need to be implemented by third-party providers such as Onfido or Jumio, which cost money and add a complexity layer.

Additionally, the app does not fetch real bank account data through official banking APIs, limiting its ability to provide real-time or verified financial information. This decision was made primarily due to the technical and legal complexity of integrating with open banking APIs, such as PSD2-compliant interfaces in Europe. These APIs often require developer registration, bank-specific onboarding procedures, and strict compliance with data protection regulations (e.g., GDPR). Given the scope and time constraints of this university project, implementing such integrations was not feasible. Instead, the app uses manually entered data to represent account balances and transactions, which, while useful for demonstration purposes, cannot replace verified financial data in a production environment.

Currently, the app supports verification for Spain, France and Germany identity documents, besides the passport option. It would be interesting to expand the service to other countries, considering differences in national ID formats (e.g., DNI, SSN).

Something similar happens when looking at the supported banks. It would be crucial for future versions to include more banks, which requires adapting to a wide variety of BIC/SWIFT codes and banking systems.

7. TABLE OF FIGURES

8. TABLE DIRECTORY

9. Declaration LLM´s usage

Throughout the development process, we leveraged artificial intelligence tools to assist us in identifying and resolving bugs more efficiently, as well as to enhance the visual styling and overall user interface of the application. These AI-driven solutions helped streamline our workflow, reduce manual effort, and improve both the functionality and design quality of our project.