Alexandria University
Faculty of Computer and Data Science
Computing and Data Science Department

AI - Powered Personal Finance Management Application

Graduation Project II - A.Y. 2024-2025

Submitted By

Basmala Salama Hassan	20221060328
Elham Hamed Mahmoud	20221468745
Irinie Magued Sabry	20221440868
Menna Allah Saeed	20221371722
Menna Allah Samy Said	20221031698

Project Supervisor

Dr. Reem Essameldin

Acknowledgments

We would like to express our sincere thanks and appreciation to our families for their support and encouragement throughout our journey, their belief in us, and the power they gave us to overcome challenges.

Special thanks to our supervisor, Dr. Reem Essameldin, who has given us this bright opportunity to engage and guide us along the way of our graduation project in FCDS, for being helpful, supportive, kind, and patient.

Abstract

Managing personal finances is a vital skill for long-term financial well-being. individuals especially those with inconsistent or limited income, often face challenges in budgeting, saving, and tracking their expenses. These difficulties are compounded by the lack of personalized, accessible tools that can adapt to individual financial behaviors and goals.

To address this gap, we developed **Savvy**, an AI-powered personal finance application that helps users monitor their financial activity, forecast future spending, and receive recommendations. The system analyzes cash flow and spending trends using machine learning algorithms, enabling it to deliver tailored insights through intuitive visualizations and goal-tracking tools.

Unlike traditional budgeting apps, Savvy dynamically adapts to user behavior and financial conditions, promoting sustainable money management practices. It empowers users to gain a clearer understanding of their financial situation, improve planning, and build long-term financial stability.

Through the integration of intelligent analytics and user-centric design, Savvy aims to close the gap between financial awareness and financial action, transforming raw data into meaningful, personalized guidance.

SAVVY

Table of Contents

List of Figures

Chapter 1: Introduction

In today's fast-paced world, the increasing complexity of financial products, variable income streams, and rising living costs have made personal financial management more difficult for many individuals. While digital financial tools have proliferated, most lack the intelligence and adaptability to address users' specific financial situations or behaviors.

By integrating intelligent prediction with user-friendly design, the application not only encourages financial discipline but also fosters informed decision-making. The goal is to build a practical and accessible platform that helps users develop sustainable financial habits and achieve their long-term objectives. and empowering users to gain greater awareness of their expenses. By analyzing historical spending data, the tool aims to forecast future expenses and provide actionable advice

1.1 Problem Statement

Many individuals struggle with managing their finances due to a lack of visibility into their spending patterns, inadequate financial planning, and the absence of tools that offer personalized recommendations. Without clear insights and accurate predictions, users are often unable to make informed financial decisions, which can lead to overspending, debt accumulation, and missed financial goals.

This project aims to address these challenges by developing a smart personal finance tool that leverages machine learning to analyze spending behavior, predict future expenses, and provide actionable recommendations.[1]

1.2 Objectives

Our project seeks to enhance individuals' financial management by providing a comprehensive, data-driven overview of their economic status. This enables users to make informed, strategic decisions that optimize financial control and planning. We analyze users' cash flow and financial objectives through advanced machine learning algorithms, offering actionable insights to support

effective business planning for future profits and expenses. Additionally, we assist users in setting realistic goals and managing expenses based on personalized spending patterns. Our focus remains on creating an intuitive, accessible interface that simplifies complex financial information, making it easily navigable.

1.4 Survey Construction and Analysis

1.4.1 Survey Objectives

Effective financial management is a challenge many face, particularly in Egypt, where the current economic conditions add additional pressures. For young adults balancing academic responsibilities, part-time jobs, and personal aspirations, the ability to save for goals, monitor daily expenses, and maintain financial discipline often proves especially difficult. Rising living costs, limited job opportunities, and fluctuating income levels have further complicated their financial journeys.

Recognizing the importance of addressing these challenges, we decided to develop an application designed to assist users in managing their finances more effectively. While our goal was to create a tool suitable for a broad audience, most responses to our initial survey came from our classmates—students aged 18 to 25. Their insights, gathered against the backdrop of Egypt's economic realities, provided valuable input, shaping our application's core features and highlighting this age group's unique financial priorities.

1.4.2 Survey overview

1.4.2.1 Demographics

- Age Groups: Most respondents are likely in the 18–34 age range, which suggests that our app's primary audience will be younger individuals, likely students or early-career professionals.
- Gender and Occupation: A mix of respondents includes students, freelancers, and employed individuals, highlighting the need for versatile features that cater to both structured (employed) and flexible (students/freelancers) financial needs.

1.6.2.2 Current Finance Management Methods

- “No specific system; I manage it as I go” highlights a gap in structured finance management for many users.
- “Manual tracking (notebooks, spreadsheets)”: Suggests the need for a user-friendly digital transition.
- “Bank apps”: Indicates a familiarity with technology and digital tools.

1.4.2.3 Financial Goals

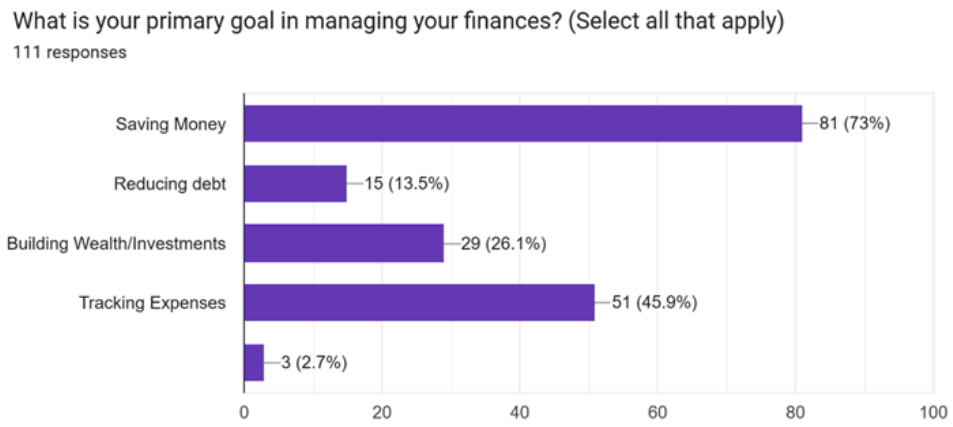


Figure 1: primary finance goals

- Saving Money: Universally desired.
- Tracking Expenses: Strong demand for monitoring financial flows.
- Reducing Debt: Less common but essential for certain users.

1.4.2.4 Frequency of expense tracking

- Sometimes (33.3%): The majority of respondents indicated that they only track their expenses occasionally. They may recognize the importance of tracking finances but lack consistency or a structured system.

- Monthly (22.5%): A significant portion of participants review their finances at the end of the month.
- Weekly (16.2%): A smaller group is highly organized and tracks their expenses weekly, reflecting a more disciplined approach to financial planning.
- Daily (13.5%): The smallest group of respondents track their expenses daily. This indicates a high level of commitment to financial management, possibly due to specific financial goals or habits.
- Not Very Often (14.4%): A notable portion of respondents do not track their expenses often, highlighting a potential gap in financial awareness or interest.

Insight: A significant portion of users are inconsistent in tracking their expenses ("Sometimes" and "Not Very Often"). The system should include features that encourage regular engagement, such as reminders or gamification elements.

For users who track expenses monthly, the system can provide monthly summaries or automated reports to simplify their process

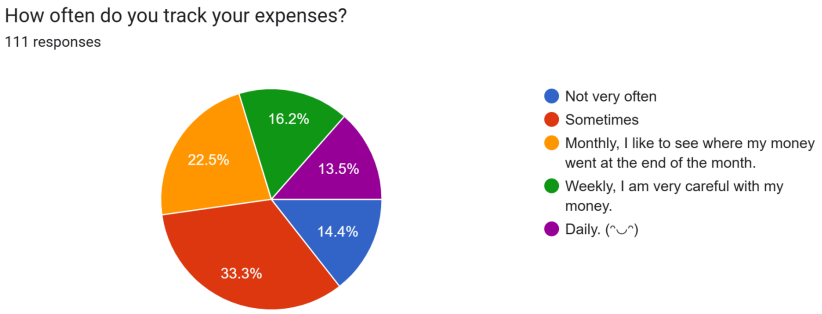


Figure 2: Tracking Expenses

1.4.2.5 Preferred features in a budget tracker

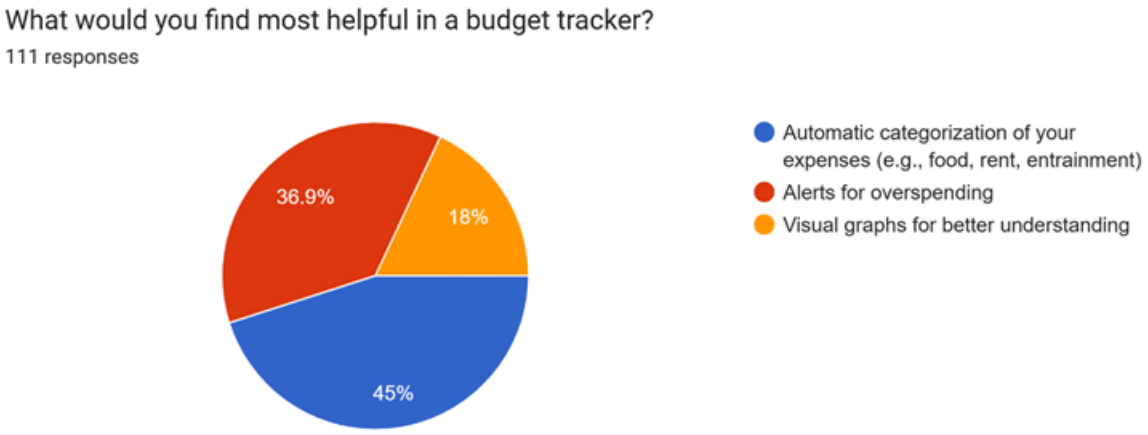


Figure 3: Tracking Budget

- Automatic categorization of expenses (e.g., food, rent, entertainment) emerged as the most desired feature, with 45% of respondents selecting it.
- Alerts for overspending were the second most popular feature, chosen by 36.9% of respondents. This reflects a strong need for tools to help users stay within their budgets and avoid financial pitfalls.
- Visual graphs for better understanding were favored by 18% of respondents, showing that some users prefer graphical representations for analyzing their financial data.

1.4.2.6 Importance of visual spending representation

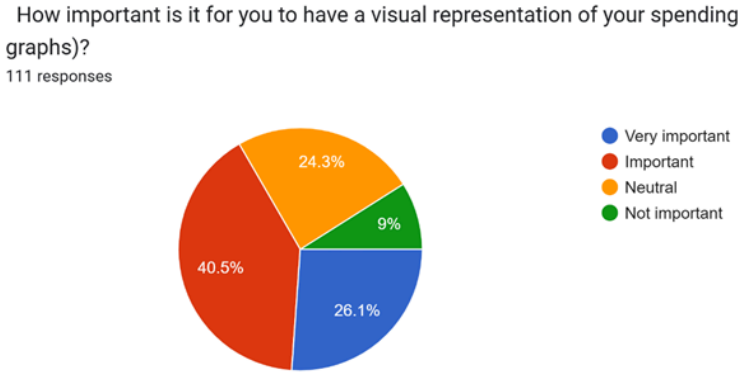


Figure 4: Visualization importance

- Majority Preference (40.5% - "Important"): A significant proportion finds visual representation important. This suggests that clear, visually appealing charts and graphs are a crucial feature.
- Strong Preference (26.1% - "Very Important"): More than a quarter of respondents highly value visual tools, emphasizing that such features will improve user satisfaction and engagement.
- Moderate Interest (24.3% - "Neutral"): While neutral respondents might not prioritize visuals, they are likely to still benefit from them as part of a well-rounded app.
- Minimal Disinterest (9% - "Not Important"): A small minority does not find visuals important, indicating a marginal trade-off for users who prefer other features.

Insights:

- Prioritizing Visuals: Include charts, and graphs to display spending trends, budget progress, and savings goals. Examples: Bar graphs for monthly expenses.
- Customizable Options: Allow users to select which data to visualize, such as weekly trends or specific categories.

1.4.2.7 Setting financial goals

Asking this question allowed us to measure whether users already set financial goals and how regularly they do so.

We found that a majority of respondents—**54.1%**—reported setting financial goals "occasionally," while **22.5%** indicated that they "rarely" set goals, and only **20.7%** stated they do so "regularly." A small minority of **2.7%** never set financial goals at all.

These findings emphasize the need for features that encourage and simplify goal-setting. By providing tools that make the process engaging and trackable, we aim to foster more consistent goal-setting habits among users. Additionally, the data underlines the importance of integrating motivational features, such as progress trackers and reminders, to help users stay committed to their financial objectives.

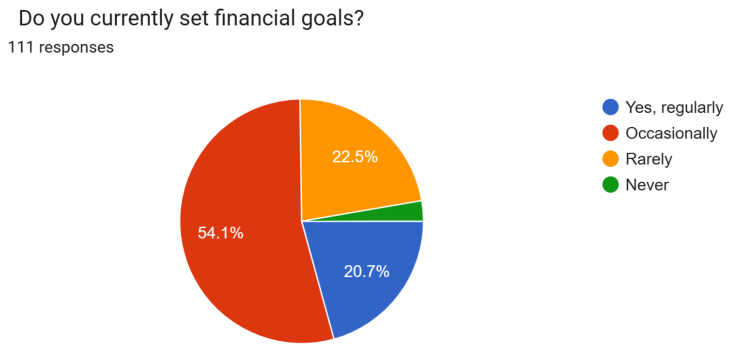


Figure 5: Current financial goals

1.4.2.8 Interest in reminders, chatbots, and AI-powered features

- Would you use a feature that tracks your progress toward financial goals? When we asked if users would benefit from a progress-tracking feature, the responses reflected an overwhelming interest. This affirmed the need for a dynamic, visual tool that motivates users to achieve their financial aspirations.

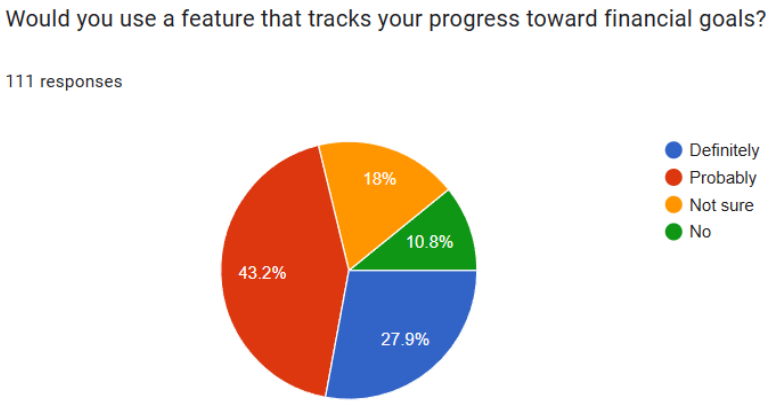


Figure 6: Tracking Expenses

- Would you trust AI to provide insights into your spending habits? We were curious about users' openness to leveraging AI for personalized insights. The majority of responses said yes, this indicates that a significant portion of the audience is open to the idea of using AI for

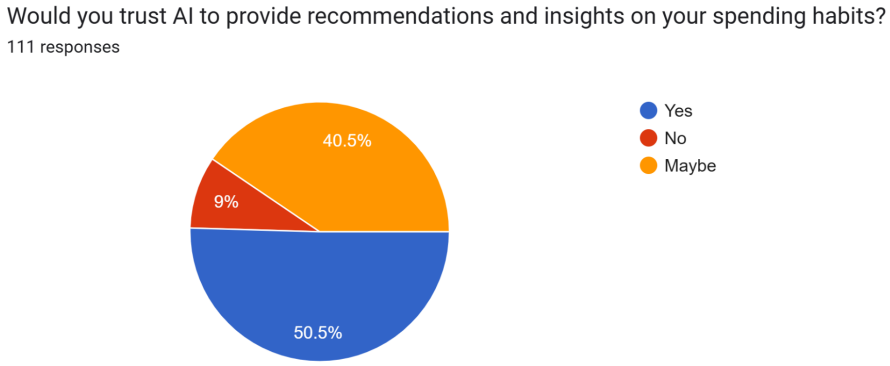


Figure 7: AI recommendations

financial guidance, although some remain uncertain. Only a small percentage (9%) expressed a lack of trust in AI.

- The question, “How helpful would reminders for bill payments or savings goals be?” is crucial as it addresses a common challenge in personal finance management. Many people struggle to remember important financial tasks due to busy lifestyles, leading to missed

payments and delayed savings. By incorporating timely reminders, users can stay on track with bill payments and savings goals, improving their financial management.

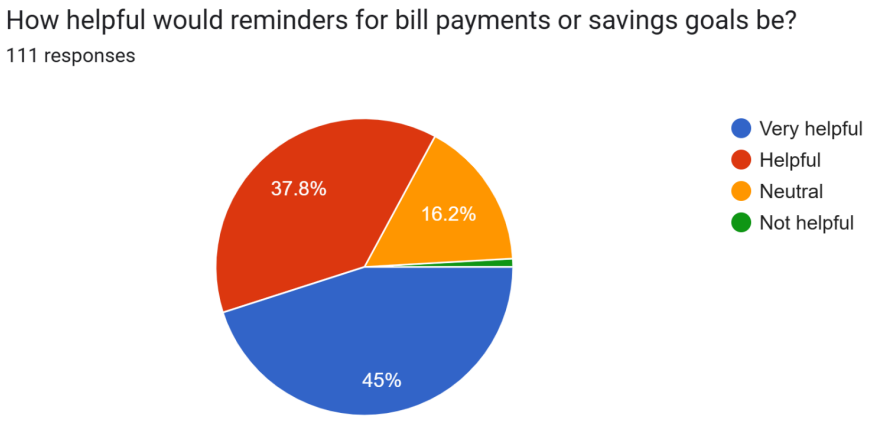


Figure 8: Reminders

- The question, "How useful would chatbot be for answering queries like 'What did I spend on food last month?' be?" is important as it addresses the convenience and efficiency of using a chatbot to quickly access personal finance data. A chatbot can provide immediate answers, saving time and improving user experience. A chatbot can streamline this process by instantly retrieving relevant data, making it easier for users to track their spending. Feedback and usage data can reveal how much users value having such instant access to information,

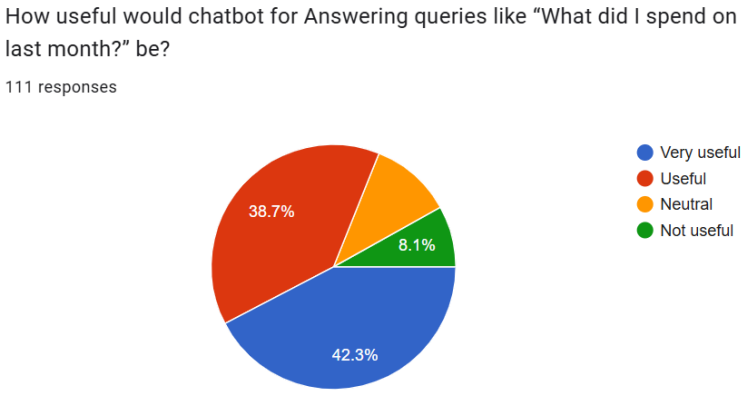


Figure 9: Answering queries

- leading to improved financial awareness and decision-making.

1.4.2.9 Features importance rank

The visualized data represents how respondents rank various features of a financial tool based on their importance.

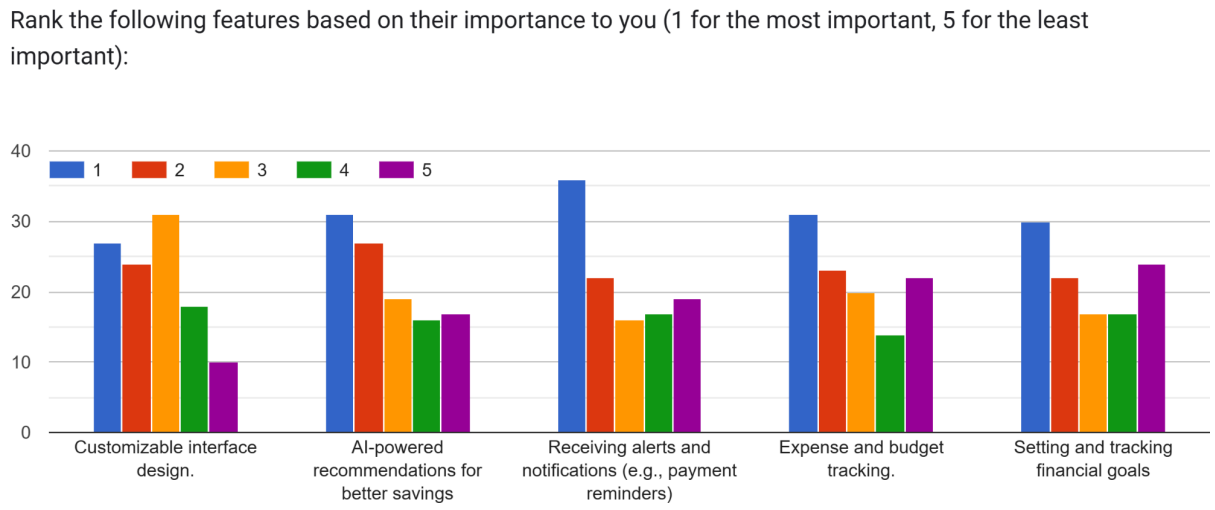


Figure 10: Features importance rank

Each bar indicates the number of respondents who assigned a specific rank (1 being the most important, 5 being the least important) to the following features:

- **Receiving Alerts and Notifications**: Dominantly ranked as the most important, this demonstrates the critical role of timely reminders in managing finances effectively.
- **Expense and Budget Tracking**: Similar to notifications, this feature garners significant importance, showcasing users' reliance on tools that help monitor and control spending habits.
- **AI-Powered Recommendations for Better Savings**: This feature shows a balanced distribution, highlighting a diverse perception of its usefulness, likely influenced by individual comfort with AI-driven tools.

- **Customizable Interface Design**: A significant number prioritize this feature moderately, indicating that while personalization is valued, it is not the top priority for most users.
- **Setting Financial Goals**: While not as heavily prioritized as notifications or budget tracking, goal setting still holds relevance, particularly for long-term financial planning.

1.4.2.10 Suggestion questions

The challenges users face with managing finances reveal pain points that a financial app can aim to solve effectively.

Based on the feedback, Challenges:

- **Difficulty in Budgeting and Predicting Expenses**: Many users struggle to set budgets for upcoming months or estimate their future expenses accurately. This highlights a need for tools that provide **flexible budgeting options** and **predictive analytics** based on past spending habits to help users prepare better for the future.
- **Economic Pressures in Egypt**: The high prices and inflation make financial management even more challenging. Users are likely looking for solutions that help them **track spending**, **identify savings opportunities**, and **prioritize essential expenses** to cope with economic uncertainty.
- **Complexity of Existing Financial Apps**: A recurring concern is that many apps are not user-friendly. They lack simplicity, making it harder for people to navigate and utilize their features effectively. This highlights a significant gap where an app offering **intuitive design**, **clear workflows**, and **minimal steps** can stand out.

1.4.2.11 Additional features users ordered

The written responses provide deeper insight into user preferences and emphasize critical factors for designing a successful financial app. Users consistently highlight ease of use and an understandable interface as top priorities, indicating their preference for a straightforward and intuitive experience. They want an app that simplifies financial management rather than complicates it.

Additionally, security emerges as a crucial factor, as users want to feel confident sharing sensitive financial data. This underscores the importance of robust security measures and transparent communication about data protection practices.

Alert notifications are frequently mentioned as an essential feature to keep users informed about deadlines, payments, or progress toward their goals. Beyond that, users express the need for features that motivate and encourage them to stay engaged, such as progress tracking, rewards, or insights that inspire continued usage.

Chapter 2: Related work

2.1 Introduction

This section reviews prior studies and techniques employed in developing finance management applications. The focus is identifying methodologies and technologies that enhance user financial planning, budgeting, and goal achievement.

2.2 Understanding financial modeling

A financial model applies academic principles to analyze accounting and business data, making predictions based on evidence. These models are used in many fields to perform tasks such as reporting to regulators, managing assets, evaluating mergers and investments, shaping business strategies, and forecasting sales. They help organizations simulate scenarios, plan effectively, and make informed decisions.

However, financial models are not without risks. They depend on assumptions and incomplete or potentially biased data, which can lead to inaccurate predictions and poor business choices. To minimize these risks, finance professionals use model validation. This process thoroughly checks the model for errors and identifies its weaknesses, helping leaders make better, more informed decisions while being aware of the model's limitations.[2]

2.3 Techniques Used in Financial Management Applications

2.3.1 Monte Carlo simulation

The Monte Carlo simulation–also known as the multiple probability simulation–uses statistical analysis to predict every possible outcome of an uncertain scenario that involves one or more undefined variables. The simulation assigns many different random values to these variables and runs the scenario repeatedly, generating numerous outcomes. Finally, the simulation takes the average of these outcomes to estimate the most likely result.[2]

The Monte Carlo method has many practical applications. For example, investors can use this technique to predict the likelihood of earning a profit from an investment. Finance professionals can also use the Monte Carlo simulation to estimate the future value of portfolios.

2.3.2 Scenario analysis

Evaluates how future events could impact a business's operations and performance over a long-term period. This method allows finance professionals to predict the effects of a wide range of deterrents, such as cyber threats, government collapse, hiring shortages, new technological developments, and terrorism. Business leaders can use scenario analysis to plan how they would respond to possible situations to achieve the best possible outcome for their company. This technique also allows companies to identify their strengths and weaknesses so they're better prepared to act in case of crisis.[2]

2.3.3 Neural Networks in Financial Forecasting

Neural networks are particularly effective because they can capture the non-linear and complex relationships between various financial indicators. For example, consider the task of predicting stock prices. Traditional methods might use linear regression, which assumes a linear relationship between the independent variables (such as historical prices, trading volumes, and interest rates) and the dependent variable (the future stock price). A neural network, on the other hand, does not assume any specific functional form for these relationships. Instead, it learns from the data, allowing it to model the intricate dependencies that exist between the input variables. For instance, the network might learn that a combination of rising interest rates, increasing trading volumes, and negative sentiment in news articles can signal a potential decline in stock prices. By capturing such complex patterns, neural networks can generate more accurate and reliable forecasts.[2]

2.3.4 Time series analysis

Time series analysis is a powerful tool for examining non-stationary data—data that fluctuates over time or is influenced by temporal factors. This analytical method is particularly valuable in industries such as finance, retail, and economics, where variables like currency exchange rates, sales figures, and market trends are constantly changing. A notable application of time series analysis is in stock market analysis, where it forms the foundation of automated trading algorithms designed to predict price movements and optimize trading strategies. In the retail sector, companies like Amazon utilize time series forecasting to prepare for significant events, such as Black Friday. By analyzing historical sales data, they can anticipate demand surges, allowing them to strategically manage inventory and logistics. This approach ensures that stock levels are adequate and resources are available, thereby optimizing operations and efficiently meeting customer demand.[2]

2.4 Main features most Personal Finance Applications have

Personal finance applications have become a critical tool for individuals seeking to manage their financial health. These applications typically provide functionalities such as expense tracking, budgeting, investment management, and debt reduction strategies. With the rapid advancements in artificial intelligence (AI), many such apps have incorporated AI-driven features, transforming them into intelligent financial assistants. The integration of AI not only enhances the user experience but also offers personalized insights, predictive analytics, and automated financial management, which are reshaping the landscape of financial technology (FinTech). Most personal finance applications share a core set of features designed to help users manage their money effectively. The following features are commonly found:[3]

2.4.1 Expense Tracking and Budgeting

AI has significantly improved the accuracy and usability of expense tracking and budgeting. Applications leverage AI to automatically categorize transactions and sync with bank accounts and credit cards or offer manual entry. Research indicates that automation reduces user effort and increases engagement by providing tailored suggestions for budget adjustments based on past

behaviors [4]. AI-driven alerts and nudges also play a critical role in preventing overspending and promoting savings habits.

Examples of Applications that have this feature: are Mint, Lunch Money, and PocketGuard.

2.4.2 Budgeting tools

Allows users to create budgets and track how their spending aligns with predefined categories. Users can set monthly budgets tailored to their spending habits, or the application will offer a suggested budget category based on past spending.[5]

Examples of Applications that have this feature:

You Need A Budget (YNAB) which uses Zero-Based Budgeting for allocating users' money in which every dollar has a job and another application similar to it called EveryDollar focuses on budgeting.[6]

2.4.3 Financial Insights and Reports

AI models integrated into personal finance apps enable predictive financial analytics and insights. Tools use machine learning to forecast future expenses and suggest optimal savings plans. These models rely on historical data and external factors such as economic trends to deliver actionable insights. Users of AI-enabled forecasting tools demonstrate better financial outcomes, such as increased savings rates and reduced debt.[7]

Examples of Applications that have this feature: are Albert and Clarity Money.

2.4.4 Investment and Net Worth Tracking

Investment and net worth tracking apps help users monitor their financial health by aggregating and analyzing their assets and liabilities. These tools provide insights into portfolio performance, asset allocation, and hidden investment fees, along with retirement and growth projections. For net worth, they consolidate accounts, and track assets (e.g., savings, properties, investments), and liabilities

(e.g., loans, credit card debt) to calculate total net worth, displaying trends over time. Examples of Applications that have this feature:

The Personal Capital (Empower) application Offers paid advisory services for personalized financial planning and investment management. It tracks net worth by consolidating all assets and liabilities, offering a clear view of financial health.

Range application is a financial advisory platform focused on investment management, financial planning, and wealth management. The platform provides users access to a team of advisors and tools to support various financial aspects, including investment strategies, retirement planning, and equity compensation.

Also, the rise of robo-advisors demonstrates the role of AI in investment management. These systems utilize algorithms to create and manage diversified portfolios based on individual risk tolerance and financial goals. The incorporation of Natural Language Processing (NLP) enables apps to offer user-friendly explanations of complex investment strategies. Literature suggests that AI-based investment tools outperform traditional methods by optimizing returns while minimizing risk through continuous data-driven analysis.[8]

Examples of Applications that have this feature: are Wealthfront, Betterment, Acorns, and Robinhood.

2.4.5 Goal setting and tracking

Users specify their financial goals, such as saving for a vacation, building an emergency fund, paying off debt, or planning for retirement. This is often done through a simple, guided process where the user selects the type of goal and sets a target amount and time frame.

They automatically track and categorize transactions to ensure that each expense and income is accounted for within the context of the user's goal. The apps visually display goal progress, often through graphs, progress bars, or percentage completions. As users contribute funds or reach spending milestones, these tools update dynamically.

A lot of applications have this feature like YNAB, and Mint.

2.4.6 Multi-Currency and International Features

Multi-currency support and international features are becoming increasingly important in financial apps, especially for users who deal with multiple currencies, travel frequently, or have international income sources.

An application like Mint is primarily designed for the U.S. and Canadian markets. It links to financial institutions, credit cards, loans, and investment accounts in these countries so it does not support financial institution integration or currency outside the U.S. and Canada.

YNAB also focuses on users in specific countries, primarily the U.S. and Canada, and it does not provide built-in multi-currency support. However, YNAB users can manage multi-currency budgets by manually adding foreign bank accounts or adjusting conversion rates in their budget categories. Third-party integrations are also available for currency exchange calculations.

Personal Capital supports the tracking of multiple accounts and investments, but its international features are more limited. It mainly serves U.S.-based users, though international assets can be tracked manually. Multi-currency functionality is not a key feature, making it less suitable for global users who need automatic currency conversions.

Lunch Money supports both multi-currency and cryptocurrency tracking, making it versatile for international users and those with digital asset portfolios. The app allows users to track financial data across over 90 currencies, ensuring that transactions in different currencies are accurately accounted for.

2.4.7 Data Security and User Privacy

Despite their benefits, AI-driven personal finance apps face challenges such as data security, user privacy, and algorithmic bias. The reliance on sensitive financial data necessitates robust security measures and compliance with regulations like GDPR. Moreover, biased algorithms can lead to unfair financial recommendations, disproportionately affecting certain user demographics.[9]

2.4.8 Emerging Trends and Future Directions

Recent advancements, such as conversational AI and generative models, are expected to further revolutionize personal finance apps. Voice-activated financial assistants powered by technologies like GPT (Generative Pre-trained Transformers) are increasingly popular. Furthermore, the integration of blockchain for enhanced security and decentralized finance (DeFi) capabilities is an emerging area of interest. Future research should focus on combining these technologies with AI to create more robust, secure, and user-centric financial solutions.

2.4 Additional Features in Our Application

Our personal finance application differentiates itself by incorporating innovative features that enhance user experience and deliver auto data entry and extraction. This feature is designed to provide added value and cater to specific user needs beyond the common functionalities found in most personal finance applications:

2.4.1 AI Chatbot

Our Conversational AI Assistant streamlines financial management by handling user transactions efficiently. Key features include:

2.4.1.1 Answering User Queries

Provides instant responses to questions like, “What did I spend on food last month?”

2.4.1.2 Automatic Transaction Categorization

Automatically classifies expenses into the correct category based on user input. For example:

- User: “I ate lunch today for 200.”
- Chatbot: “Recorded 200 in the Food category.”

This functionality ensures accurate expense tracking with minimal effort, helping users stay organized and focused on managing their finances.

2.5 Feature Matrix

Figure 11: Feature Matrix

Feature	Savvy	Mint	YNAB	PocketGuard	EveryDollar	GoodBudget
Expense & Income Tracking	✅	✅	✅	✅	✅	✅
Budget Tracking	✅	✅	✅	✅	✅	✅
Customizable Categories	❌	✅	✅	✅	✅	✅
Goal Setting & Monitoring	✅	✅	✅	✅	✅	✅
Reports & Analytics	✅	✅	✅	✅	✅	✅
Spending Notifications	✅	✅	✅	✅	✅	✅
Security & User Management	✅	✅	✅	✅	✅	✅
Personalized Recommendations	✅	✅	✅	✅	❌	❌
Expense Predictions	✅	✅	❌	❌	❌	❌
Auto Categorization	✅	✅	✅	✅	❌	✅
Bank Sync	❌	✅	✅	✅	✅	✅
AI Chatbot	✅	❌	❌	❌	❌	❌

Chapter 3: Requirements Specification

This chapter outlines the essential requirements for the Personal Finance Tracking Application, a tool designed to empower users with effective financial management capabilities. It delves into the system's functional and non-functional requirements, providing a clear roadmap for implementation.

3.1 Functional Requirements

Functional requirements clearly outline the specific behaviors and functionalities that the system must exhibit. These core features are vital to our application's value and effectiveness.

3.1.1 User Registration and Authentication

- Users can register using an email address and a password.
- Users can log in securely and manage their accounts.

3.1.2 Expense Tracking

- Users can input daily expenses into predefined categories.

3.1.3 Income Tracking

- Users can log their sources of income and track earnings over time.

3.1.4 Budgeting

- Users can set monthly or annual budgets for specific categories.
- The system alerts users when they approach or exceed budget limits.

3.1.5 Spending Analysis

- The system provides visual reports (charts, graphs) of spending patterns.
- Users can view their spending aligns with their financial goals.

- Periodic Summaries: Summarize income, expenses, and savings to reflect financial health.

3.1.6 Predictive Analysis

- Machine learning models predict future expenses based on historical spending data.

3.1.7 Goal Setting

- Users can define financial goals (e.g., vacation savings).
- The system provides progress updates and goals.
- Allow users to set specific savings goals (e.g., save $1,000 in 6 months) and monitor progress.
- Provide visual indicators for goal completion percentages and remaining amounts.

3.1.8 Notifications and Alerts

- Notifications for overspending, upcoming bill payments, or budget limits.
- Notify users when category budgets are exceeded.
- Balance Warnings: Alert users when balances fall below a specific threshold to avoid overdrafts.

3.1.9 AI Chatbot

Conversational AI Assistant: A planned feature to enhance user interaction, providing capabilities such as:

- Answering queries like “What did I spend on food last month?”
- Automatic Transaction Categorization, e.g., “I ate lunch today for 200.”

3.2 Non-Functional Requirements

Non-functional requirements specify the quality attributes of the system.

3.2.1 Performance

The application should load data and provide insights within 2 seconds for a seamless user experience.

3.2.2 Scalability

The system should handle increasing numbers of users and data inputs without performance degradation.

3.2.3 Security

All data must be encrypted during storage and transmission.

3.2.4 Usability

The interface must be intuitive and user-friendly for people with varying levels of technical expertise.

3.2.5 Reliability

The system must maintain 99.9% uptime.

3.2.6 Maintainability

- The codebase should be modular to facilitate updates and bug fixes.
- The application should load data and provide insights within 2 seconds for a seamless user experience

3.2.7 Compliance

Ensure that the application complies with the General Data Protection Regulation (GDPR) to protect the personal data of users.

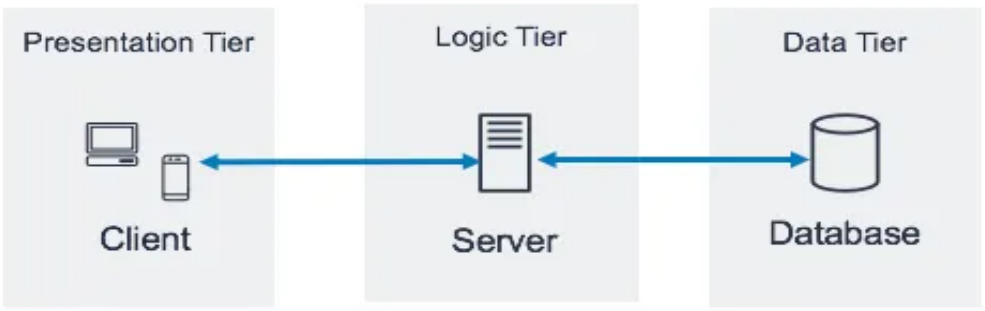
3.3 Software Diagrams

Software diagrams serve as visual representations of a system's architecture, components, and interactions. This helps to understand the structure and functionality of the software, ensuring clear communication and alignment during development. [10]

3.3.1 Three-Tier System Architecture

The Three-Tier Architecture consists of three distinct layers or tiers:

1. **Presentation Layer (Frontend)**: This is where the user interacts with the application.
2. **Application Layer (Backend/Logic)**: This layer contains the business logic and processes user requests.
3. **Data Layer**: This tier stores and retrieves data as needed by the application layer.



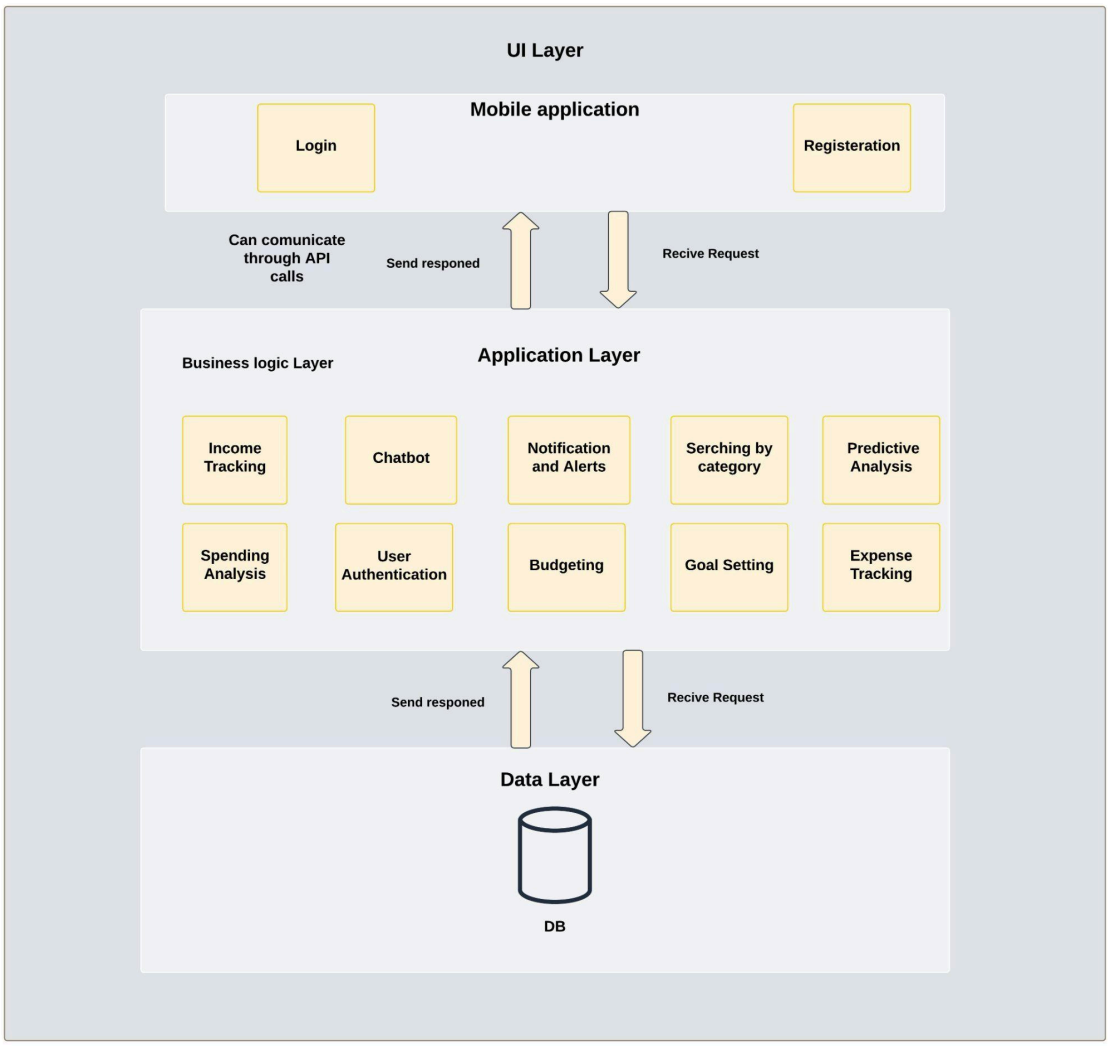


Figure 12: Three-Tier System Architecture

3.3.1.1 User Interface Layer (Presentation layer)

Mobile Application:

This is the front-end layer where users interact with the application through their mobile devices. It serves as the interface for users to log in, register, and access various features such as budgeting and tracking expenses.

Login & Registration:

These are the key features that allow users to Login and register if they don't have an account within the system. Once logged in, users gain access to their personal finance data.

3.3.1.2 Application Layer (Business Logic Layer)

This layer manages the application's core functionalities. It is responsible for processing data, implementing the business logic, and facilitating communication between the UI layer and the data layer.

- **Spending Analysis:**
  This feature analyzes the user's spending patterns to identify trends and provide insights into their financial habits.
- **Budgeting:**
  A budgeting feature allows users to create, track, and manage their budget goals, providing them with the ability to set limits on spending categories.
- **Chatbot:** Users can ask to add income or expense transactions automatically, and they can receive answers to queries like, "What did I spend on food last month?"
- **Notification and Alerts:**
  This functionality sends real-time alerts and reminders to users about important financial events, like bill due dates or budget thresholds being exceeded.
- **Income Tracking:**
  Users can track their income sources, helping them to get an overview of their financial inflows.
- **Goal Setting:**
  This feature allows users to define financial goals (such as saving for a trip or purchasing an item), and track their progress.
- **Expense Tracking:**
  Users can log and categorize their expenses, enabling better monitoring of their spending habits and they can track their expenses using visual graphs.

- **User Authentication:**
  This ensures secure login and access to user data, verifying identities through login credentials.
- **Search by category :**
  Users can search their expenses or income by category.
- **Predictive Analysis:**
  This feature uses machine learning or statistical methods to predict future financial trends based on past data.

3.3.1.3 Data Layer

This layer is responsible for storing, managing, and retrieving all the data used by the system.

SQL Database (SQL DB):
All sensitive user data and financial records are securely stored in this database. Encryption ensures that data is protected and complies with privacy standards.

1. **User Data:** Contains personal information such as the user's login credentials, preferences, and profile**.**
2. **Financial Records:** Includes transaction data, budget details, income, expenses, and other financial records.
3. **Budget and Goals:** Stores the users' budgets, financial goals, and their progress toward achieving them.
4. **Machine Learning (ML) Models:** This component houses the trained AI models used for predictive analysis, personalized recommendations, and spending analysis.

Data Flow and Interaction:

- **UI Layer to Application Layer:** The user interacts with the mobile application, triggering actions such as logging in, registering, or setting goals, which are processed by the application layer.
- **Application Layer to Data Layer:** The application layer communicates with the data layer to store and retrieve user data, financial records, and other information.

- **AI and Predictive Analysis:** The system utilizes machine learning models stored in the data layer for analysis and prediction, feeding insights back to the application layer for user interaction.

3.3.2 Entity-Relationship Diagram (ERD)

This ERD explains the structure and relationships of the database designed for a Personal Financial Management Application. The database consists of seven interconnected tables: User, Goal, Goal Type, Category, Transaction, Budget, and Notifications. Each table serves a specific purpose and is linked to others through well-defined relationships to ensure data integrity.

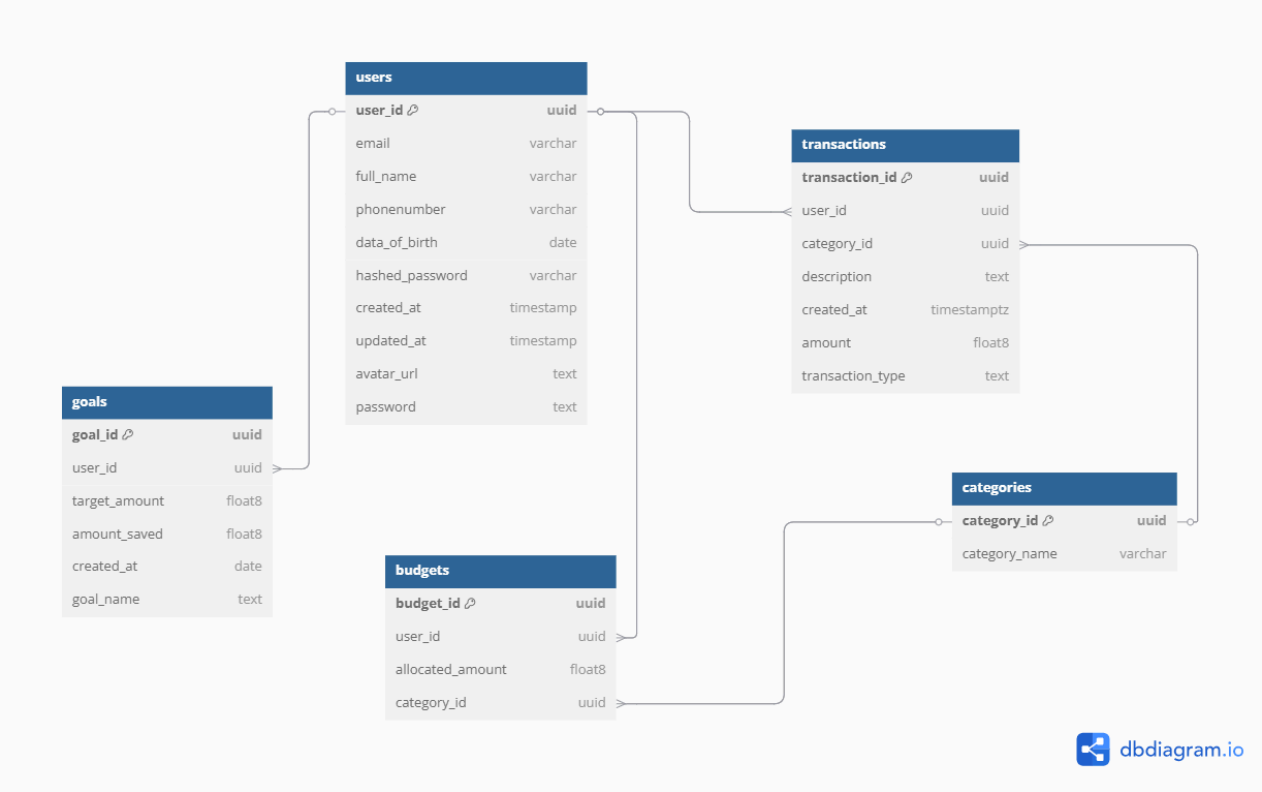


Figure 13: Entity relationship diagram

Tables and Attributes

1. User Table: Stores information about the users of the application.

Attributes:

- user_id (UUID, Primary Key): Unique identifier for each user.
- email (VARCHAR): User's email address for login and communication.
- full_name (VARCHAR): Full name of the user.
- hashed_password (VARCHAR): Securely stored hashed password for authentication.
- phonenumber (VARCHAR): User's phone number.
- data_of_birth (DATE): User's birthdate.
- created_at (TIMESTAMP): Timestamp when the account was created.
- updated_at (TIMESTAMP): Timestamp when the record was last updated
- avatar_url (TEXT): URL of the user's profile image

2. Goal Table: Stores details about the financial goals set by users.

Attributes:

- goal_id (UUID, Primary Key): Unique identifier for each goal.
- user_id (UUID, Foreign Key → users.user_id): References the user who owns the goal.
- target_amount (FLOAT8): The financial target for the goal.
- amount_saved (FLOAT8): The amount saved so far.
- created_at (DATE): Date when the goal was created.
- goal_name (TEXT): Name/title of the goal.

3. Category Table: Categorizes transactions, budgets, and goals for better organization.

Attributes:

- category_id (UUID, Primary Key): Unique identifier for each category.
- category_name (VARCHAR): The name of the category.

4. Transaction Table: Tracks financial transactions made by users.

Attributes:

- transaction_id (UUID, Primary Key): Unique identifier for each transaction.
- user_id (UUID, Foreign Key → users.user_id): References the user who made the transaction.
- category_id (UUID, Foreign Key → categories.category_id): References the category of the transaction.
- description (TEXT): Description of the transaction.
- created_at (TIMESTAMPTZ): Timestamp when the transaction occurred.
- amount (FLOAT8): Total amount involved in the transaction.
- transaction_type (TEXT): Indicates type (e.g., "income", "expense").

5. Budget Table: Manages the allocation of funds to different categories for users.

Attributes:

- budget_id (UUID, Primary Key): Unique identifier for each budget.
- user_id (UUID, Foreign Key → users.user_id): References the user who owns the budget.
- category_id (UUID, Foreign Key → categories.category_id): References the budget's category.
- allocated_amount (FLOAT8): The amount allocated to this category.

3.3.3 Data Flow Diagram (DFD)

In this graphical representation, the flow of data within a system is provided showing how data moves through different processes and how they are transformed.

3.3.3.1 Context level/ level 0 (Overview)

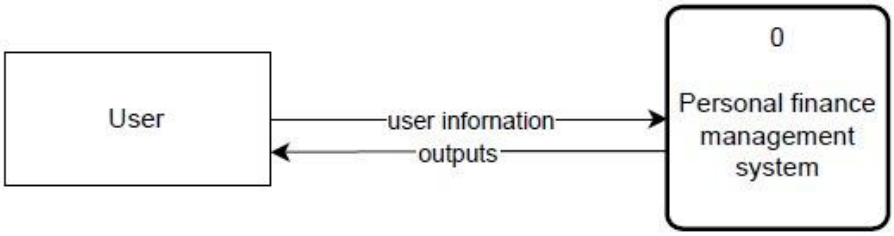


Figure 14: DFD Level 0

User Information is data provided by the user, such as personal details, income, expenses, and financial goals.

Outputs: Information the system generates, such as financial reports, budgets, and charts.

3.3.3.2 DFD Level 1

Overview: This DFD level provides a more detailed system view, breaking down the Level 0 process into several sub-processes.

External Entity: user interacting with the system

Key Processes

- **1.0 Register:** Handles user authentication and registration.
- **2.0 Adding transactions and categorizing:** Transactions(expense/income) details entered by the user and categorizes them.
- **3.0 Set budget:** Creates, updates, and manages budgets.
- **4.0 Analyze Spending and Visualize Summaries:** Analyzes user spending patterns and visualizes summaries and goal progress to help users understand their financial behavior.
- **5.0 Predict Future Expenses:** Uses historical transaction data and spending trends to predict future expenses.
- **6.0 Set and track goals:** Enables users to define financial goals and track progress toward achieving them.
- **7.0 Interacting with chatbot:** provides an interactive interface for users to ask questions, get insights, and receive financial advice based on real-time, historical data.
- **8.0 Send Notification:** Generates alerts and notifications such as goal reminders, budget warnings, or spending updates.

Data Stores

User Data: Stores user credentials, preferences, and profile information.

- **Inputs**: Data from the registration process and user preferences.
- **Outputs**: Used for authentication and personalization of features like notifications.

Transaction Data: Stores all income and expense transactions.

- **Inputs**: Transaction details from users.
- **Outputs**: Used for budgeting, analysis, and visualizations.

Budget Data: Stores user-defined budgets (spending limits).

- **Inputs**: Budget details entered by users.
- **Outputs**: Provides alerts and indication of budget limit per category.

Category: Maintains categories' names.

- **Outputs**: Provides categories' names.

Goals Data: Stores user choice of financial goals and their progress.

- **Inputs**: Goal details and updates from the goal-tracking process.
- **Outputs**: Provides updates and insights for tracking goals.

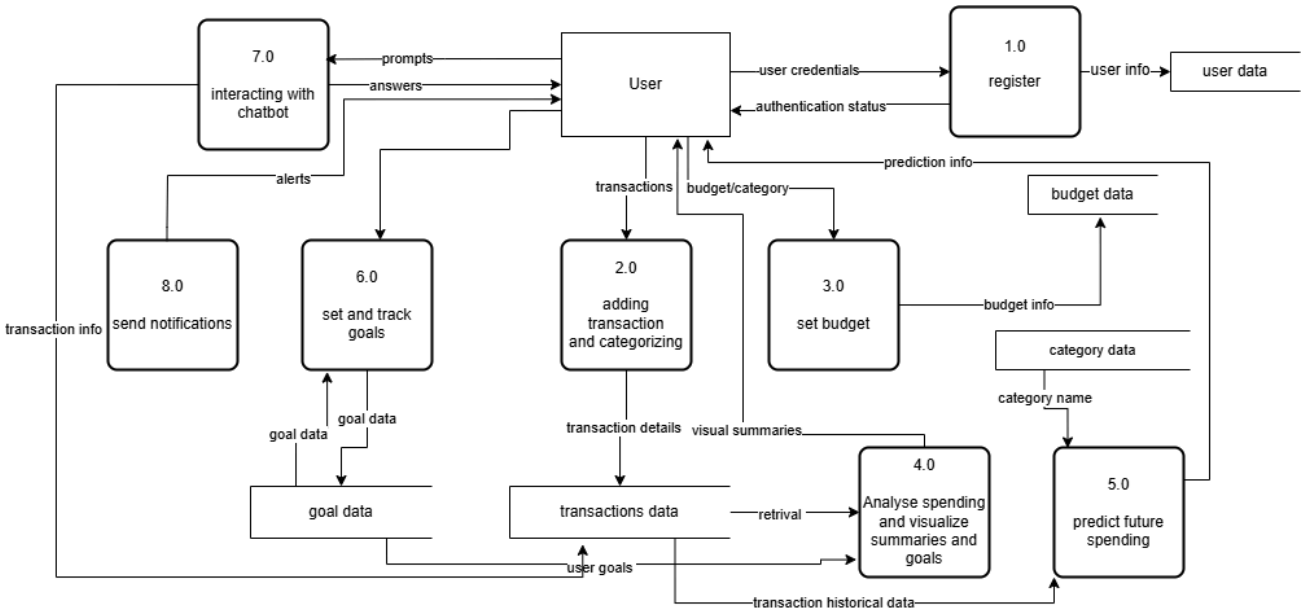


Figure 15: DFD Level 1

3.3.4 Sequence Diagrams

Sequence Diagrams are used to illustrate the main features of an application by showing how different pages, components, and users interact with each other over time. They help visualize the flow of actions, data exchange, and the order of operations, making it easier to understand how the application functions as a whole.

3.3.4.1 Authentication

The sequence diagram for Authentication illustrates the interaction between the user, frontend interface (login page, signup page), backend server, and database. It shows how a user enters their credentials, how the frontend sends this data to the backend, and how the backend validates the credentials by checking the database. If the information is correct, the server responds with a success message and may issue a session token or redirect the user to the dashboard. This diagram highlights the key steps and components involved in securely logging into the application.

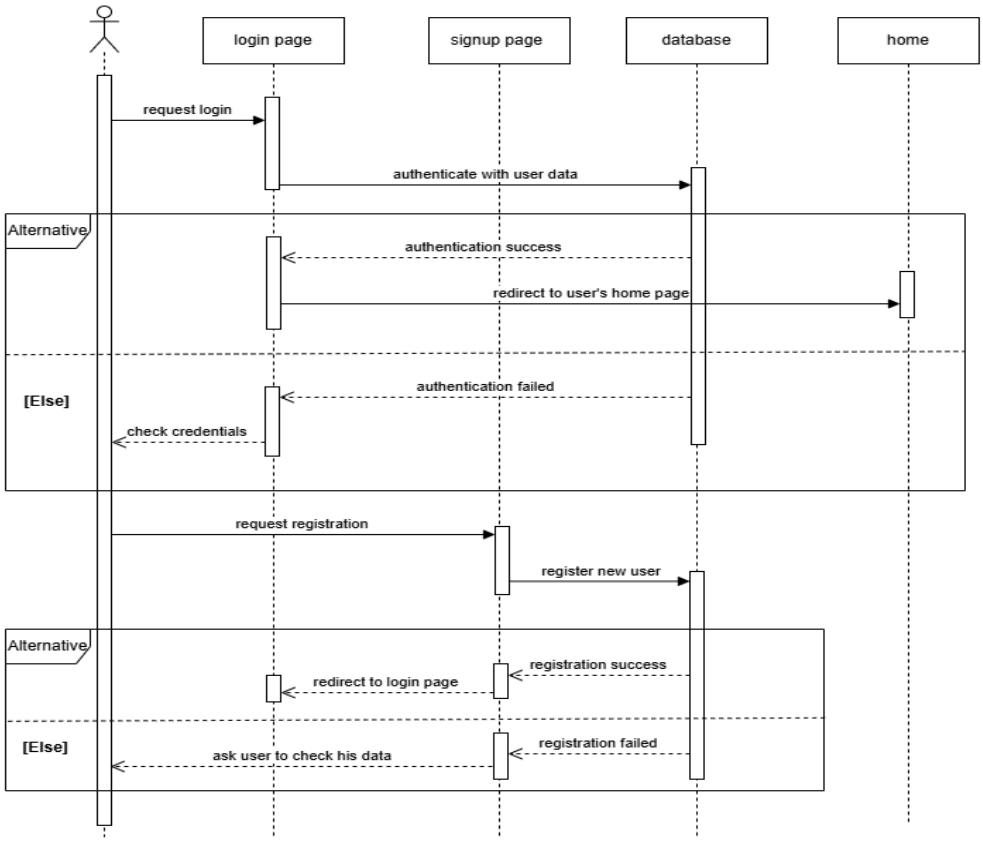


Figure 16: Sequence diagram authentication

3.3.4.2 Add Transactions

The sequence diagram for adding transactions demonstrates how a user interacts with the application to add either an Income or Expense entry. The user selects the transaction type, chooses a category (Salary, Food, Transportation,...etc), and enters the amount and details. The frontend sends this data to the backend, which validates and stores the transaction in the database. Once saved, the updated transaction list can be retrieved and displayed on the Transactions screen, where all income and expense entries are viewable in one place.

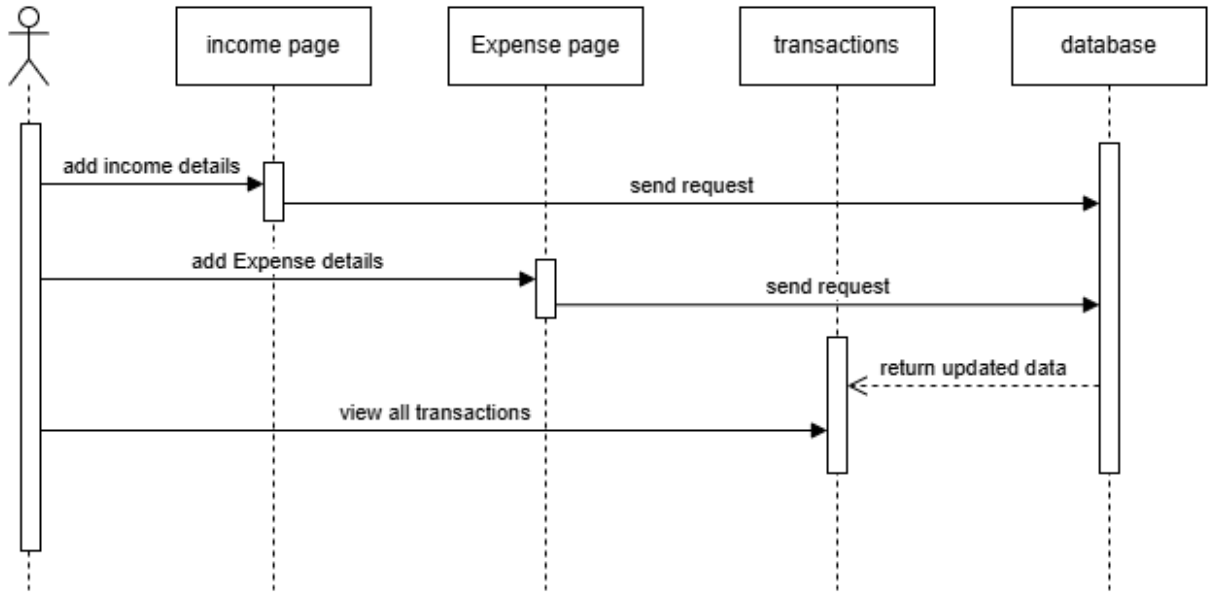


Figure 17: Sequence diagram transactions

3.3.4.2 Chatbot

The sequence diagram for the Chatbot feature includes two main interaction types:

- **Insight Chatbot** gives users summaries and analysis of their transactions (e.g., spending trends).

- **Action Chatbot** helps users add new transactions by guiding them through selecting type, category, and amount, then saving the data.
  Both improve user experience through smart, conversational interaction.

Both chatbot types enhance user interaction by making financial management more conversational and intuitive.

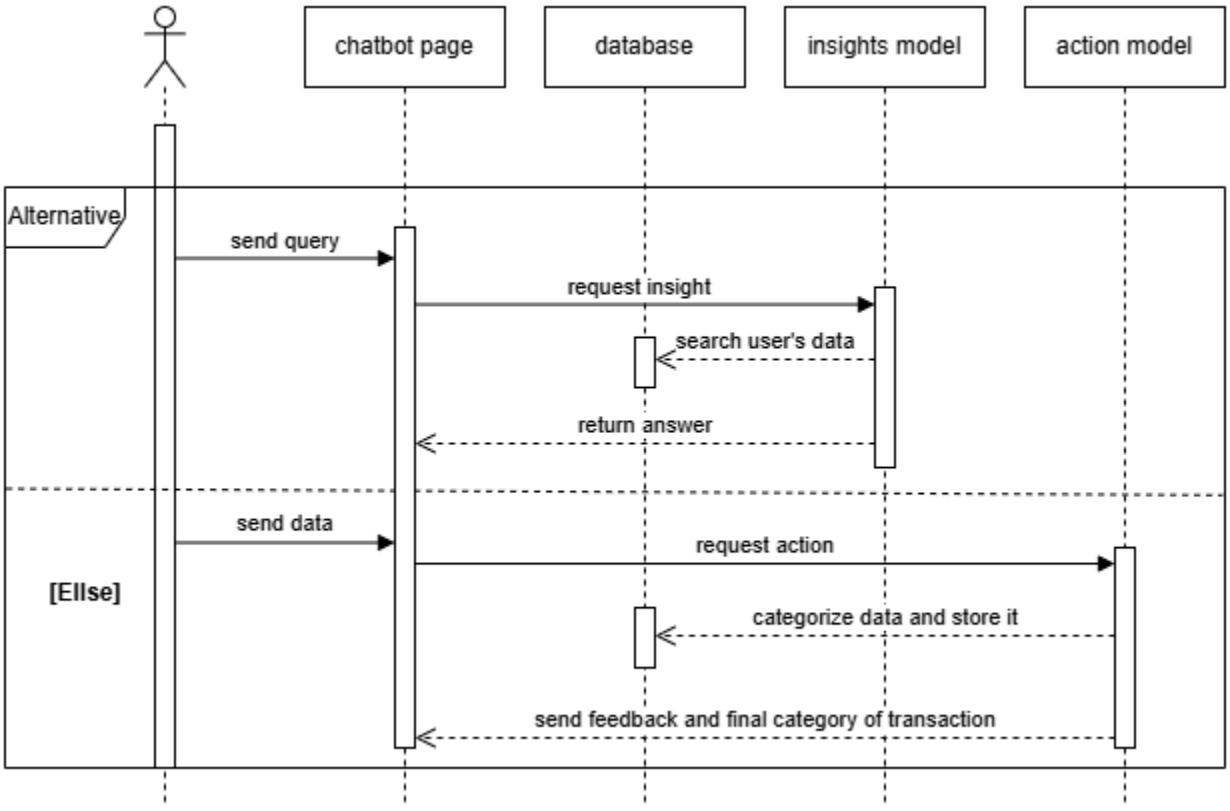


Figure 18: Sequence diagram chatbot

3.3.5 Use Case diagram

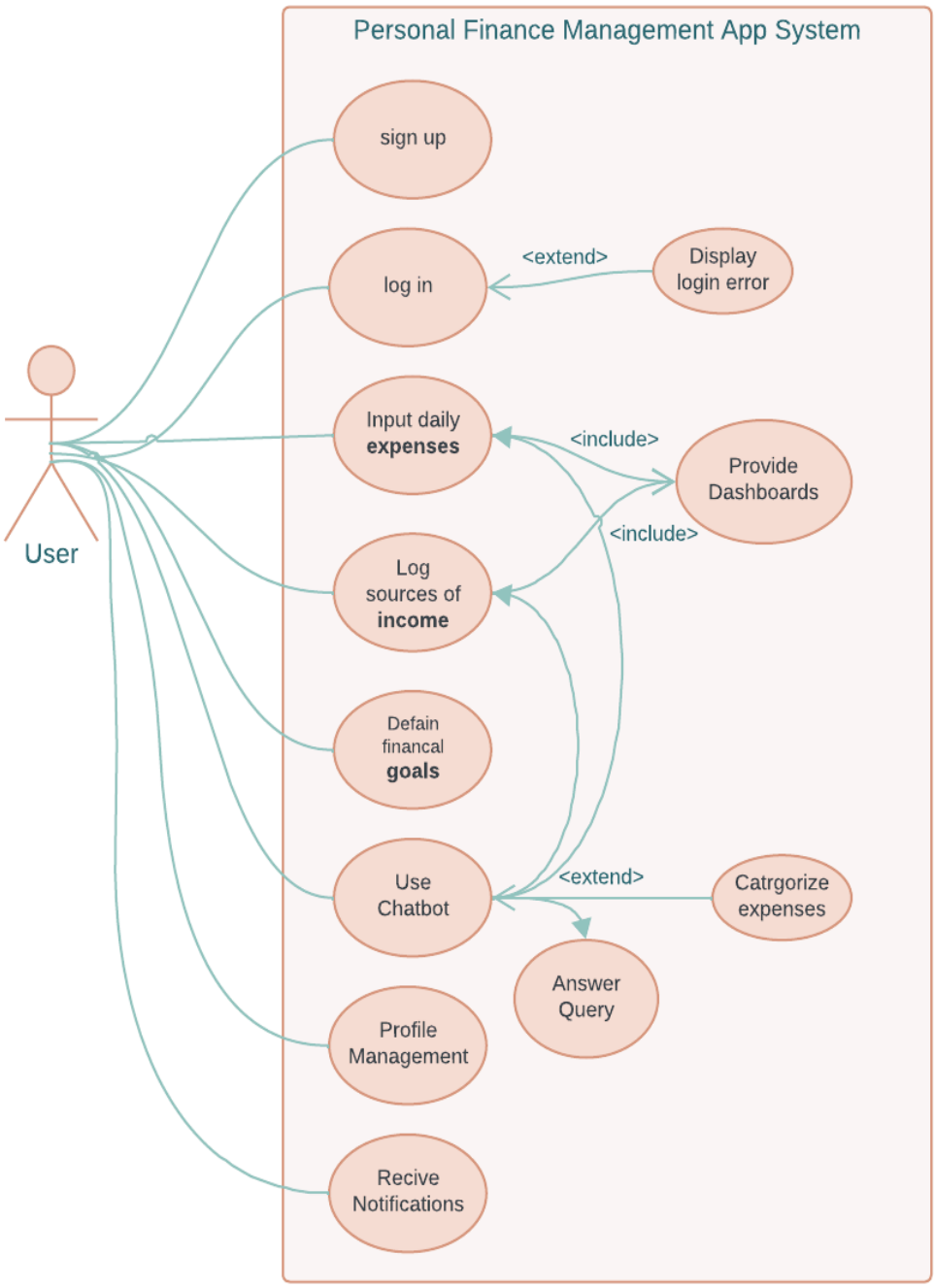


Figure 19: Use case

3.3.5.1 Main Components

1. **Actor**
    - **User**: The primary user of the system interacts with various functionalities provided by the app.
2. **System Boundary**
    - The rectangle labeled **Personal Finance Management App System** encloses all the use cases (functionalities) available in the system.

3.3.5.2 Use Cases

Each oval represents a specific use case, or feature, that the system offers. The diagram also specifies relationships between use cases, such as include and extend, which provide additional context.

3.3.5.3 Use Cases and Descriptions

1. **Sign Up**:
    - Allows the user to register for the app by providing the necessary credentials or information to create an account.
2. **Log In**:
    - Enables the user to access the app by entering their credentials (e.g., username and password).
    - Extend Relationship: If login credentials are incorrect, the system triggers the Display Login Error use case to notify the user.
3. **Input Daily Expenses**:
    - Provides functionality for the user to record their daily expenses.
    - Include Relationship: Automatically triggers the Provide Dashboards use case to visually represent the entered expenses for better tracking and analysis.
4. **Log Sources of Income**:
    - Allows the user to record their income sources, such as salary or freelance payments.
    - Include Relationship: Similar to expenses, logging income also triggers Provide Dashboards to visually show the income trends.
5. **Define Financial Goals**:

- Lets the user set specific financial goals, such as saving a target amount or reducing expenses.

6. **Use Chatbot**:
    - Provides an AI-powered chatbot for answering user queries and assisting with financial management.
    - Extend Relationship: The chatbot can trigger the Categorize Expenses use case when the user asks to add expenses without specifying a category.
    - Answer queries, input expenses, and income are core parts of the chatbot's functionality.
7. **Profile Management**:
    - Offers functionality for users to update their personal information, account settings, and preferences.
8. **Receive Notifications**:
    - Sends timely notifications to users about their financial activities, reminders for goals, or unusual spending patterns.

3.4 SWOT Analysis

3.4.1 Strengths

- AI Chatbot: Unique selling point compared to basic budgeting apps.
- Predictive Analysis: Uses machine learning models to provide accurate predictions, such as spending trends or goal achievement probabilities.
- Auto Categorization: Automatically categorizes transactions and budgets based on user data and past patterns, saving time and improving accuracy.
- User-Friendly Interface: Designed to simplify financial management for non-experts.
- Customization: Users can define their goals and budgets, offering flexibility

3.4.2 Weaknesses

- Limited Resources: Small team with restricted budget and time.

- Technical Challenges: Implementing AI and ML models accurately may require advanced expertise.
- User Adoption: Convincing users to trust a new app for financial management.
- Data Privacy Concerns: Ensuring robust security measures to protect sensitive user data.

3.4.3 Opportunities

- Growing Demand: Increasing interest in personal finance tools, Between students, freelancers, and all individuals
- Partnerships: Potential collaboration with universities, banks, or financial organizations.
- Growing Awareness: Increasing interest in financial literacy among younger generations could boost app demand.
- Technological Advancements: Leveraging new AI/ML tools to improve predictions and insights.

3.4.4 Threats

- Competition: Established apps like Mint or YNAB may overshadow the project.
- Regulatory Issues: Handling user data responsibly and complying with privacy laws like GDPR is essential.
- Technical Failures: Bugs or inaccuracies in predictions could harm user trust.
- User Retention: If the app doesn't provide unique value, users may switch to competitors.[11]
- Highlight unique selling points, such as a focus on AI Chatbot, in marketing efforts.[12]

3.5 Risk Analysis

3.5.1 Data Privacy and Security Risks

Storing and processing sensitive financial data exposes the project to risks of data breaches.

3.5.2 Model Accuracy and Reliability

Inaccurate predictions due to flawed algorithms or insufficient data could harm user trust.

3.5.3 User Engagement Risks

The application may fail to engage users if the interface is not intuitive or if the recommendations are not actionable.

3.5.4 Competition Risks

Competing products with similar or better features may overshadow the project

3.5.5 Solutions

- Implement end-to-end encryption, secure authentication methods, and regular security audits.
- Use diverse datasets for training, validate models rigorously, and provide disclaimers about prediction limitations.
- Conduct user testing and feedback sessions to refine the UI/UX and recommendation engine.

3.6 Software Development Methodology

3.6.1 Introduction

The personal finance application will be conducted based on the Incremental methodology. This choice was driven by the evolving nature of our project and the need for flexibility and adaptability throughout its development lifecycle. [13]

3.6.2 Project increments

Increment 1: Core Features

Objective: Establish the foundational functionalities, allowing users to register, log in, and input basic financial data (expenses and income).

Features to Develop:

- **User Registration and Authentication**
    - Email/password registration and secure login.
    - Profile management (update email/password).
- **Expense Tracking us**
    - Predefined categories for expense classification.
- **Income Tracking**
    - Input sources of income.
    - Track earnings over time.
- **Goal Setting:**
    - Users can define financial goals (e.g., saving $500 in 6 months).
    - Provide visual progress indicators for goals.

Expected Deliverables:

- Fully functional registration, login, and profile management.
- Functional expense and income tracking.
- Data is saved to the database for retrieval.
- Users can set financial goals.

Increment 2: Budgeting and chatbot
Objective: Add features for setting budgeting limits and chatbot for interaction.

Features to Develop:

- **Budgeting**
    - Users can set budgets for specific categories (Food, Rent, Entertainment, and others).
- **Chatbot**
    - Users can interact with the chatbot.[14]
- **Notifications and Alert**
    - Notify users about:
        1. Approaching budget limits.
        2. Exceeding budgets.

3. Predicted overspending in specific categories.

Expected Deliverables

- Functional budgeting system with category-specific settings.
- Chatbot integration.
- Notifications and Alerts.

Increment 3: Spending Analysis

Objective: Implement insights for spending patterns and create charts for visualization.

Features to Develop:

- **Time Series Model**:
  - Develop a financial forecasting system using historical expense data.
- **Spending Analysis Charts:**
  - Create visual reports (bar charts, pie charts, etc.) showing spending by category.

Expected Deliverables:

1. Users can view visual spending trends.
2. Future Expenses prediction.

3.7 Mobile application development

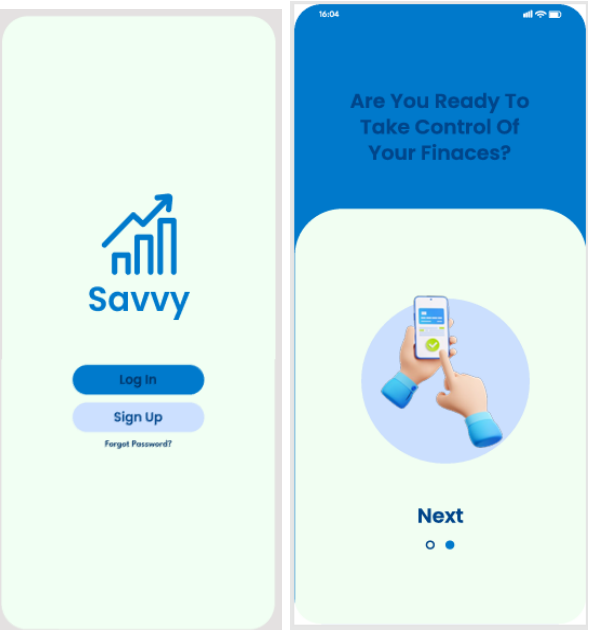
3.7.1 Framework and development tools

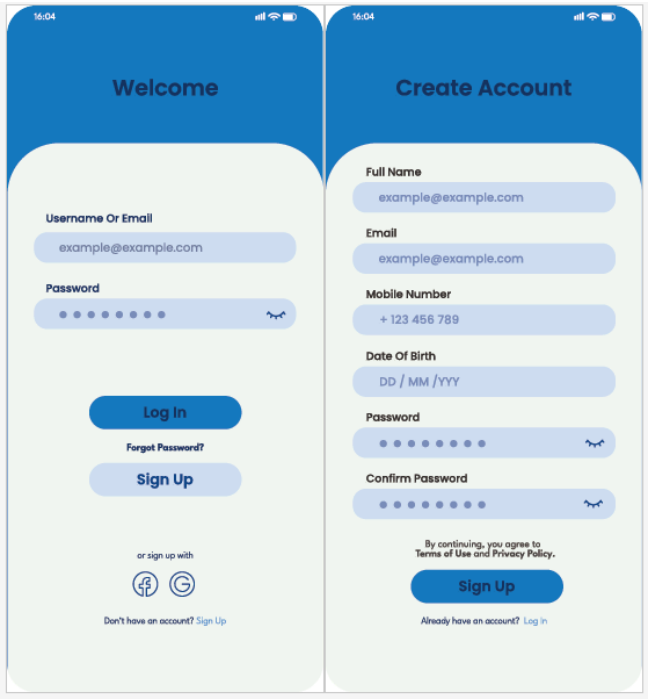
1. Flutter SDK (Version 3.0 or later): For cross-platform mobile app development.
2. Dart language (Version 2.12 or later): For building the app's logic and UI components.
3. Visual Studio code with flutter extensions.
4. Flutter DevTools for debugging and performance profiling.

3.7.2 Backend and Database

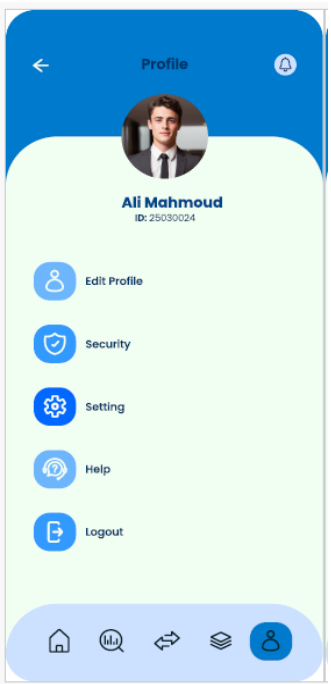
1. PostgreSQL: A relational database for storing structured financial data.
2. FastAPI /RestAPI or Flask.
3. SQLAlchemy: An ORM (Object Relational Mapper) for interacting with the database efficiently.
4. Supabase Authentication: for secure user authentication and identity management.
5. Git/Github: for Version Control and Collaboration

3.8 Prototype

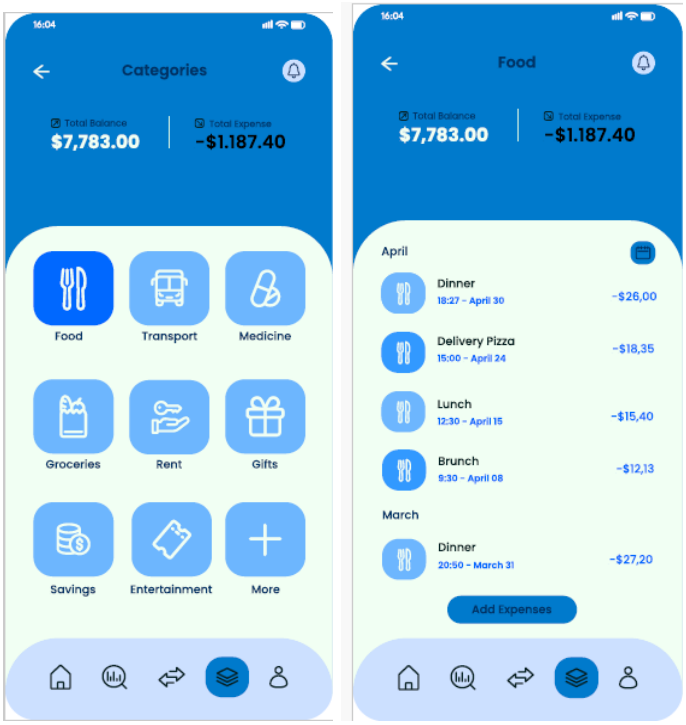




Sign up/sign in



User profile home page quick analysis


16:04
Categories
Total Balance
$7,783.00
Total Expense
-$1.187.40
Food
Transport
Medicine
Groceries
Rent
Gifts
Savings
Entertainment
More
16:04
Food
Total Balance
$7,783.00
Total Expense
-$1.187.40
April
Dinner
18:27 - April 30
-$26,00
Delivery Pizza
15:00 - April 24
-$18,35
Lunch
12:30 - April 15
-$15,40
Brunch
9:30 - April 08
-$12,13
March
Dinner
20:50 - March 31
-$27,20
Add Expenses

Categories

Chapter 4: Machine learning and AI

4.1 Chatbot Implementation Using LangChain

4.1.1 Brief overview of LangChain and its role

LangChain is an open-source framework designed to simplify the development of applications powered by large language models (LLMs). Rather than treating an LLM as a standalone black box, LangChain provides modular building blocks such as prompt templates, chains, agents, retrievers, and memory components that let you compose complex workflows clearly and maintainable.

4.1.2 Langchain Components

1- Prompt Templates: These are blueprints for crafting prompts to guide language models. They allow dynamic input insertion, ensuring consistent and structured interactions. For example, a template might define a question format where variables like user input are swapped.

2- Memory: This component enables applications to retain context across interactions. It stores conversation history or relevant data, allowing the model to reference past exchanges. Types include conversation buffers, summary memory, or vector store-backed memory for long-term retention.

3- Indexes: Indexes (now often referred to as retrievers in LangChain) organize and retrieve external data efficiently. They connect language models to documents, databases, or knowledge bases, enabling retrieval-augmented generation (RAG). Common implementations include vector stores like FAISS or Chroma for semantic search.

4- Chains: Chains are sequences of operations combining prompts, model calls, and data processing. They structure workflows, such as retrieving data, feeding it into a prompt, and parsing the output. Examples include the LLMChain for simple prompt-model interactions or RetrievalQA for question-answering over documents.

5- Agents: Agents are decision-making components that use language models to choose actions dynamically. They combine tools (e.g., search APIs, calculators) with reasoning to solve complex tasks. Agents iterate through action-observation loops, guided by strategies like React or tool-calling.

6- Tools: These are external utilities or functions that agents or chains can call to perform tasks, such as querying a database, running a web search, or executing code. LangChain provides integrations for tools like Google Search, Wikipedia, or custom APIs.
7- Output Parsers: This structure model outputs into usable formats, such as JSON, lists, or custom objects. They ensure the model's responses are consistent and actionable, especially for programmatic use.

8- Callbacks: Callbacks allow developers to hook into the lifecycle of LangChain operations for logging, monitoring, or custom behavior. They're useful for debugging or tracking performance metrics.

9- Document Loaders and Text Splitters: Document loaders ingest data from various sources (e.g., PDFs, web pages, CSVs), while text splitters break large documents into manageable chunks for processing or indexing, crucial for RAG workflows.

10- Vector Stores: These are specialized databases for storing and querying high-dimensional vectors (embeddings) generated from text. They power semantic search and retrieval.

11- Embeddings: Embeddings convert text into numerical vectors for semantic understanding. LangChain supports providers like OpenAI, Hugging Face, or Cohere to generate embeddings for documents or queries.

12- Language Models (LLMs and Chat Models): These are the core AI components, including large language models (e.g., GPT-4, Llama, DeepSeek) for text generation and chat models optimized for dialogue. LangChain abstracts its APIs for seamless integration.[15]

4.1.3 Why do we use Langchain

LangChain is used to simplify and enhance the development of applications powered by language models, particularly for tasks requiring context, external data, or complex workflows. Here's why developers choose LangChain.

1- Modularity & Reusability:

- Breaks complex logic into discrete "chains," "agents," and "tools," so we can develop, test, and maintain Q&A and categorization workflows independently.

2- Simplified Integration with External Data:

- Through document loaders, vector stores, and retrievers, LangChain enables seamless integration of language models with custom datasets or knowledge bases, facilitating retrieval-augmented generation (RAG) for accurate, data-driven outputs.

3- Advanced Natural Language Understanding Requirements:

- Modern users expect financial applications to understand complex queries expressed in natural language, ranging from simple balance inquiries to sophisticated investment strategy discussions. Traditional rule-based systems prove inadequate for handling the nuanced language patterns and contextual dependencies inherent in financial conversations. LangChain provides sophisticated natural language processing capabilities that enable applications to interpret user intent accurately, even when queries involve complex financial concepts or require multi-step reasoning processes.The framework's integration with advanced language models enables applications to understand financial terminology, recognize relationships between different financial concepts, and maintain conversational context across extended interactions. This capability proves essential for personal finance applications where users may need to explore various scenarios, compare options, and receive explanations tailored to their specific financial situations.

4- Streamlined Prompt Management:

- Prompt templates ensure consistent, reusable prompts with dynamic inputs, reducing errors and improving model performance across diverse use cases.
- Prompt templates and output parsers standardize and simplify experimentation, crucial when tuning prompts for financial insights versus expense classification.

5- Conversation Memory:

- Memory modules help maintain user context (e.g., past queries or preferences) across turns, enhancing the chatbot’s continuity in longer dialogues.

6- Asynchronous & Batched I/O:

- LangChain’s LLM wrappers and retrievers support async calls out of the box. You can fire off embedding lookups, vector-store queries, and LLM generations in parallel, whereas a basic Hugging Face pipeline tends to run everything sequentially.

7- Pluggable Vectorstores:

- You can swap in high-performance vector databases (FAISS with GPU support, Pinecone, etc.) with minimal code changes. Those are typically faster at nearest-neighbor search than writing your own in pure Python.

8- Retrieval-Augmented Generation for Financial Advisory Services:

- Personal finance management requires access to current, accurate, and comprehensive financial information that extends far beyond what can be embedded within a single language model. LangChain's retrieval-augmented generation capabilities enable applications to combine the reasoning capabilities of large language models with access to extensive financial knowledge bases, regulatory documents, and real-time market information.[16]

4.1.4 Key Tools and Libraries

1- DeepSeek API:

- Provides specialized financial-insight embeddings and inference endpoints.
- We call DeepSeek to generate contextual vectors for transaction descriptions, which power both the Q&A insights and auto-categorization steps.
- The dual functionality of DeepSeek for both embedding generation and inference endpoints creates architectural efficiency by reducing integration complexity and maintaining consistency in financial domain understanding across different application functions.
- The decision to utilize DeepSeek for contextual vector generation specifically for transaction descriptions demonstrates understanding of the critical importance of accurate transaction categorization in personal finance management

2- Fireworks Embeddings:

- Our primary embedding model for the RAG pipeline.
- Integrated via LangChain’s vectorstore interface to index PDFs, CSV exports, and live PostgreSQL transactions.
- Supports incremental updates, so new transactions are available for retrieval immediately.

- Fireworks' integration through LangChain's vectorstore interface provides standardized access patterns that simplify development while maintaining compatibility with the broader LangChain ecosystem.

3- FAISS (Facebook AI Similarity Search):

- High-performance, in-memory vector database used as our default vector store.
- Enables sub-second nearest-neighbor searches over thousands of document chunks or transaction embeddings.
- Tuned with HNSW indexing for fast insertions and lookups
- The selection of FAISS as the vector database solution reflects a performance-oriented approach that prioritizes query speed and memory efficiency over distributed scalability features.
- The implementation of HNSW indexing demonstrates sophisticated understanding of the trade-offs between insertion speed and query performance. This configuration choice indicates optimization for applications with frequent data updates, which aligns well with financial applications requiring real-time transaction processing.

4- Pandas & CSV Loaders:

- For initial bulk ingestion of historical transaction exports.
- Used in our one-time preprocessing step to split, clean, and load CSVs into both the database and the vectorstore.

5- PyPDF2 Loader:

- Extracts text from bank statements and financial reports.
- Feeds into our chunking pipeline, allowing insights to reference official report language when answering questions.

6- Conversational Retrieval Chain:

- ConversationalRetrievalChain is a LangChain pipeline that allows your chatbot to:
    1. Understand context from previous messages (i.e., carry on a real conversation).
    2. Retrieve relevant documents (from PDFs or user transactions) using vector search.
    3. Generate personalized answers using a large language model (LLM) like DeepSeek.
- It remembers the chat, pulls useful info, and responds smartly.

- The implementation of ConversationalRetrievalChain as the core dialogue management component demonstrates sophisticated understanding of the requirements for financial advisory conversations. The integration of retrieval capabilities within the conversational flow enables contextually appropriate responses that can reference specific financial documents and transaction histories.

7- Conversation Buffer Memory:

- Stores the chat history between the user and the assistant.
- Maintains a running memory of past questions and answers so the assistant can understand context in a multi-turn conversation.
- Without memory, your chatbot would treat every question in isolation. This makes follow-ups (like "What should I do next month?" or "Can you explain more about last week's spending?") **feel natural** and **intelligent**.

8- Prompt Template:

- Defines how the LLM should behave and respond. It's the instruction you give to the model on how to use the context, history, and current question.
- Guides the model to use both **external documents** and **user data**.
- Provide actionable advice and personalize the user's situation.

9- Ensemble Retriever:

- EnsembleRetriever is a meta-retriever in LangChain that combines the results from multiple different retrievers into one final set of documents.
- Instead of relying on a single source of truth (like just PDFs or just PostgreSQL), it blends the strengths of multiple retrievers, giving you more comprehensive and relevant results.

4.2 RAG pipeline details

Our system architecture is centered around a robust RAG pipeline, designed to support personalized financial guidance through the integration of external documents and user-specific data. The RAG pipeline enhances the capabilities of the language model by augmenting its context with relevant retrieved documents before generating a response. The core components and flow are detailed below.

4.2.1 Data sources

- **General Financial Knowledge:**

  PDF documents serve as a foundational source of **trusted, professional financial knowledge** in the system. These PDFs include expert guides, handbooks, and tips on personal finance management, budgeting, saving strategies, and investment basics. By integrating this curated content, the chatbot can provide **authoritative, evidence-backed advice** that complements personalized transaction insights.

- **User Transaction History:**

  Fetched in real-time from a PostgreSQL database (via Supabase) using the user ID. Each transaction record is formatted into a readable document and embedded for semantic search. This enables personalized responses grounded in a user's financial behavior.

4.2.2 Document loading and chunking

1- PDFs are processed with PyPDFLoader to extract and clean raw text.

2-Transactions are converted into natural language summaries, formatted as mini-documents.

3-Both sources are chunked using RecursiveCharacterTextSplitter to ensure contextually coherent input units.

4.2.3 Embedding generation

All documents and transaction summaries are converted into vector embeddings (**Embedding** are numerical representation of the text) using FireworksEmbeddings, which utilizes DeepSeek models under the hood for financial domain alignment; text with similar content will have similar vectors.

4.2.4 Vectorstore (FIASS)

- Embeddings are stored and indexed in **FAISS**, configured with HNSW for optimized similarity search.
- Enables fast nearest-neighbor queries to retrieve the most relevant content in real time.

4.2.5 Retrieval System

- Two separate retrievers are instantiated:
  - **General Retriever** for PDF advice.
  - **Personal Retriever** for transaction history.

- These are combined using EnsembleRetriever, allowing weighted aggregation of both sources (e.g., 50% general + 50% personal), ensuring balanced relevance in answers.
- We used MMR(Maximum Marginal relevance) that always take the document that are most similar to the query in the embedding space, we send a query then we initially get back set of responses with “fetch_k” parameter that can control in order to determine how many responses we get this based on semantic similarity we can work with smaller set of docs and optimize for only most relevant ones.

4.2.6 Conversation Memory

- Implemented with ConversationBufferMemory, which keeps track of ongoing dialogue for context continuity.
- This enables follow-up questions and multi-turn interactions without losing context.

4.2.7 Prompt Engineering

A custom PromptTemplate guides the language model to:

- Distinguish between personal and general finance context.
- Provide actionable, empathetic advice tailored to the user.
- Maintain professionalism and user-centered tone.

4.2.8 Conversational Retrieval Chain

- The orchestrator of the pipeline.
- Combines the retriever, LLM (DeepSeek via Fireworks), memory, and prompt into one callable chain.
- Automatically retrieves relevant documents based on user input, feeds them to the LLM, and returns a generated answer.
- We used the stuff method, which is the default approach, to consolidate all the documents into the final prompt. This is beneficial because it requires only one call to the language model.[17]

4.2.9 User session lifecycle management

The session lifecycle implements a dual-mode initialization strategy that optimizes performance through intelligent resource management. The system distinguishes between established and new user sessions, applying appropriate initialization procedures to ensure optimal response times.

4.2.10 Warm start protocol

When receiving user queries, the system immediately evaluates whether an active session exists for the provided user identifier. For established sessions, the warm start protocol leverages pre-configured computational resources including semantically embedded vector representations, configured retrieval mechanisms, and preserved conversational context. This approach eliminates computational overhead associated with data processing, enabling immediate query processing with minimal latency while maintaining contextual awareness throughout extended advisory sessions.

4.2.11 Cold Start Initialization

New users without existing sessions undergo comprehensive cold start initialization that establishes complete technological infrastructure for personalized financial advisory services. The process begins with data ingestion from persistent storage systems, retrieving transaction histories and financial documents. Raw financial data undergoes preprocessing and semantic embedding generation to create high-dimensional vector representations capturing contextual relationships within user financial information.The system then instantiates retrieval components providing access to both general financial knowledge and user-specific information. Ensemble retrievers strategically balance authoritative financial guidance with personalized insights derived from individual transaction patterns. The initialization concludes with conversation management infrastructure establishment, including memory buffers for dialogue history and complete Retrieval-Augmented Generation chain construction.

4.2.12 Session Persistence and Performance Optimization

Successfully initialized sessions are preserved in memory structures organized by user identifier, transforming subsequent interactions from cold start scenarios to warm start protocols. This persistence strategy balances immediate accessibility with resource conservation, maintaining comprehensive session state including embedded vectors, configured retrieval mechanisms, and conversation history.The dual-mode architecture delivers substantial performance improvements by eliminating computational delays for established sessions while optimizing resource utilization through reduced redundant operations. The system maintains prepared sessions for active users while avoiding unnecessary resource allocation for inactive accounts, achieving optimal balance between responsiveness and resource conservation under varying load conditions.

4.3 Transactions chatbot

4.3.1 Overview

The goal of the Transactions Chat module is to allow users to log, categorize, and receive feedback on financial transactions in natural language as part of a seamless conversational experience. Users

are empowered to log spending or income naturally (“Bought lunch for 50 yesterday”, “Salary 8000 today”) and receive instant categorization feedback, along with personalized financial advice, in a single chat turn. This module leverages an advanced Large Language Model (LLM) prompt for robust entity extraction and classification, dramatically improving accuracy and user engagement over traditional approaches.

4.3.2 Evolution and Approach

4.3.2.1 Early Attempts and Limitations

Our initial attempts at transaction categorization and extraction leveraged a BERT-based classifier for intent detection and entity extraction. While BERT showed promise for clean, predictable inputs, it struggled significantly with the nuanced requirements of a dynamic financial chatbot.[18]

- Limited Natural Language Understanding: BERT proved insufficient for handling the full spectrum of user input, including informal phrasing, relative date references (“last Friday”, “this morning”), and varied ways of expressing amounts. This led to frequent misinterpretations and parsing failures.
- Inadequate Conversational Interactivity: BERT’s strength lies in classification, not generative dialogue. Providing rich, empathetic, and personalized feedback was difficult without extensive rule-based systems or additional generative models, resulting in stiff, uninteresting responses.
- Brittle Categorization: Achieving high accuracy in categorizing transactions required a rigid set of rules and extensive data for fine-tuning BERT, making the system inflexible to new transaction types or user phrasing. Many edge cases for amounts, descriptions, and categories remained unhandled.
- Intent Detection Challenges: When we attempted to integrate BERT with the Retrieval-Augmented Generation (RAG) chatbot (designed for answering general finance questions) in a single chatbot, the system’s overall accuracy suffered drastically. The BERT component frequently failed to reliably detect the user’s true intent, whether they wanted to add a transaction or ask a question. This intent ambiguity led to misdirected responses and a frustrating user experience.

We also experimented with merging “add transaction” and “ask question” flows in a single prompt. The model often failed to correctly interpret user intent, leading to almost unusable results.

4.3.2.2 Final Solution: LLM-Driven Prompt Engineering

To overcome these significant limitations, we transitioned to a prompt-driven approach using a powerful, domain-aligned LLM (DeepSeek via Fireworks). This strategic shift delivered immediate

and substantial improvements in both the model's extraction accuracy and the overall conversational experience[18]:

- Superior Natural Language Understanding: Modern LLMs, such as DeepSeek, excel at understanding and parsing loosely structured, colloquial messages. They can infer context, handle relative dates, and extract information even from compositionally complex or out-of-distribution phrasings, effectively performing advanced intent detection and sequence labeling within a single prompt.
- Enhanced Interactivity and Personalization: The prompt empowers the LLM to not only extract structured data but also to generate natural, empathetic, and personalized feedback. This capability creates a much more interactive and helpful assistant, going beyond just data logging to provide value and engagement.
- Robust, Flexible Categorization: By defining the category labels directly within the prompt, the LLM intelligently classifies transactions into predefined types, adapting to various input styles without the need for extensive retraining or brittle rule sets. This allows for a more flexible and accurate categorization system.
- Streamlined Workflow: This LLM-based approach allows a single model to perform multiple functions (extraction, classification, response generation) in one step, simplifying the architecture compared to chaining multiple specialized models. By handling the transaction logging as a dedicated prompt interaction, it avoids the intent collision issues encountered when trying to mix it with RAG queries in a single, complex prompt.

4.3.3 Prompt Design: Rules, Structuring, and Categorization

At the heart of the Smart Finance Assistant's transaction module is a precisely engineered prompt template. This template acts as a sophisticated instruction set for the LLM, guiding its behavior, output format, and conversational tone[19][20].

Prompt Engineering Guidelines:

- Structured Output Enforcement: The prompt explicitly defines a strict output schema using labeled fields (CREATED_AT, AMOUNT, TYPE, CATEGORY, DESCRIPTION, FEEDBACK). This forces the LLM to organize its response in a predictable, machine-readable format, maximizing the reliability of subsequent parsing and database insertion.
- Dynamic Date Understanding: Critical for user convenience, the prompt explicitly instructs the LLM to interpret relative date references (e.g., "yesterday," "last week," "this morning") and to assume the current date if no specific date is provided. This natural language processing capability significantly lowers friction for users.

- Controlled Category Labels: To ensure consistent and accurate data for analytics, the prompt explicitly list the allowed categories for both income and expense transactions. This method constrains the LLM's output to a predefined set, preventing the generation of unclassified or inconsistent labels.
- Conversational Feedback Generation: A key feature enhancing user engagement is the instruction for the LLM to generate natural, empathetic, and actionable feedback (FEEDBACK field). This turns a simple logging operation into an interactive experience, providing value beyond basic data entry (e.g., suggesting savings tips, complimenting good decisions, or expressing concern for high spending).
- Few-Shot Examples: The inclusion of specific input-output examples within the prompt (e.g., "I got today 2,000 EGP from upwork") serves as "few-shot prompting." These examples are crucial for guiding the LLM, clarifying the desired output format, and demonstrating how to handle various input nuances, thereby improving its generalization capabilities[20][21].

Connecting to LangChain Concepts:

- Prompt Templates: The transaction_prompt directly embodies the concept of a PromptTemplate. It is a blueprint that allows for dynamic input insertion ({input_text}, {current_date}) while ensuring a consistent structure for interacting with the DeepSeek LLM[21].
- Parametric Knowledge: The LLM's inherent understanding of financial concepts, date parsing, and general conversation styles (acquired during its pre-training) constitutes its parametric knowledge. The prompt leverages this deep-seated understanding.
- Source Knowledge: The specific user query (input_text) and the current date (current_date) passed into the prompt are examples of source knowledge. This information is provided at inference time, allowing the LLM to tailor its response to the immediate context of the user's transaction.

4.3.4 NLP Strategy & Why This Approach Excels

The core NLP strategy for the Transactions Chatbot relies on the advanced capabilities of the DeepSeek LLM, guided by meticulous prompt engineering. This contrasts sharply with our earlier, less successful attempts using BERT[18].

- Robust Entity Extraction: Unlike BERT, which often requires explicit named entity recognition (NER) models or complex rule sets for each entity (amount, date, type, category), the LLM, driven by this prompt, performs integrated entity extraction. It intelligently identifies and pulls out the AMOUNT, CREATED_AT, TYPE, CATEGORY, and DESCRIPTION from diverse, natural language inputs in a single inferential step.

- Contextual Categorization: The LLM's ability to understand the semantic meaning of the user's transaction description, combined with the explicit category list in the prompt, enables highly accurate categorization (CATEGORY). It moves beyond keyword matching to infer the most appropriate category based on the overall context, leading to fewer misclassifications than rule-based or simple classifier systems.
- Intent Understanding via Prompting: Instead of a separate intent detection model, the design of the prompt itself, combined with the broader architecture, manages intent. For the "add transaction" intent, a dedicated prompt is used, focusing the LLM entirely on extraction and categorization. This dedicated approach ensures high accuracy for this specific task, overcoming the critical issue of intent detection failure observed when trying to combine transaction logging and RAG in a single, complex LLM prompt.
- Generative Natural Language Generation (NLG): The FEEDBACK component showcases the LLM's NLG capabilities. It generates personalized, contextually relevant, and varied conversational responses. This interactive feedback loop significantly enhances the user experience, making the chatbot feel more like a helpful financial advisor than a mere data entry tool. This was a significant limitation with BERT, which lacks the generative capacity for dynamic, human-like responses.

In essence, by switching from a BERT-based approach to a sophisticated LLM prompt, we transformed a brittle, limited system into a highly accurate, interactive, and user-friendly Smart Finance Assistant. The LLM's inherent understanding and generative capabilities, when precisely guided by the prompt, eliminate the need for complex, multi-component NLP pipelines and deliver a superior user experience.

4.3.5 Output Parsing

While LangChain offers dedicated Output Parsers classes (4.1.2.7), the Transactions Chatbot module implements a custom parse_llm_response function. This function serves as the critical bridge between the LLM's free-form, structured text output and the structured data required for database insertion.

- Purpose: Its primary role is to reliably extract each piece of information (e.g., CREATED_AT, AMOUNT, TYPE, CATEGORY, DESCRIPTION, FEEDBACK) from the LLM's response, converting it into a programmatic dictionary.
- Robustness: The custom parser utilizes regular expressions (re.search) to robustly match patterns defined in the prompt's output format. It also includes fallback mechanisms (e.g., defaulting CREATED_AT to datetime.now() if parsing fails) to ensure resilience against minor LLM variations.

- Integration: This custom parser ensures that the LLM's human-readable, structured output is consistently transformed into a format that can be seamlessly validated and inserted into the Supabase database.

4.4 Time Series

4.4.1 Introduction

In this section, we explore time series analysis as a means of forecasting future monthly expenses. Time series forecasting is particularly suitable for financial data, as it captures patterns and trends over time, enabling more accurate budgeting and planning. To achieve this, we employed two distinct approaches: a traditional statistical model and a deep learning-based method.

The first approach utilizes Seasonal Autoregressive Integrated Moving Average (SARIMA), a well-established technique in time series forecasting that models seasonality, trends, and autocorrelations in data. The second approach leverages the power of deep learning through a Long Short-Term Memory (LSTM) network, which is capable of capturing complex temporal dependencies and non-linear patterns in multivariate financial records.

By comparing these two models, we aim to evaluate their effectiveness in predicting users' future monthly expenses and determine which method provides more reliable and insightful forecasts for our budgeting application.[22]

4.4.2 Data Selection and Preparation

For accurate time series forecasting, the quality and structure of the input data are crucial. In this study, we selected a dataset that includes users' monthly spending across various predefined categories, with a focus on capturing long-term patterns and trends.

The dataset contains expense records organized by month and year, extending up to 2025, which provides a sufficiently long time frame to identify seasonal behaviors and trends in spending. Each entry reflects the total amount spent by the user in specific categories, including Food, Transportation, Health, Entertainment, Education, Fashion, and Lifestyle.[23]

These categories were chosen to reflect the most common areas of personal expenditure and to support a multivariate analysis that considers how spending in one category may influence or relate to another over time. Ensuring the presence of timestamps and maintaining consistent frequency (monthly records) were essential steps before applying both SARIMA and LSTM models.

By preparing the data in this way, we enable the forecasting models to learn from historical patterns and generate meaningful predictions that can assist users in better managing their future expenses.

4.4.2.1 Data Exploration

Before building predictive models, it is essential to explore the structure and behavior of the expense data to understand underlying patterns and potential challenges.

Spending Distribution by Category

To understand how users allocate their spending, we visualized the distribution of total expenses across different categories using a pie chart. The analysis revealed that Food is the dominant category, accounting for approximately 49% of total expenses. This indicates that food-related spending is a major component of users' budgets and will likely have a significant impact on overall expense trends. The remaining categories Transportation, Health, Entertainment, Education, Fashion, and Lifestyle make up smaller proportions, but still contribute meaningfully to the spending profile

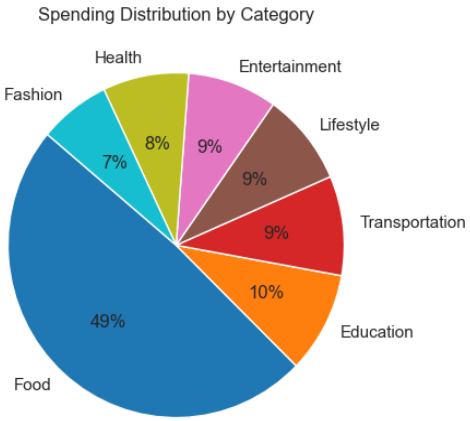


Figure 20: Category distribution

Daily Expense Trends

We also examined daily expense trends to identify the temporal behavior of spending. The plotted time series shows noticeable spikes, which are likely outliers caused by one-time large purchases or irregular spending events (e.g., travel, medical emergencies, or shopping sprees). These irregularities highlight the need for models that are robust to noise and capable of distinguishing regular patterns from anomalies.

This exploration guided our preprocessing steps and model choices, ensuring that the forecasting process considers both the dominant categories and the presence of outlier behavior in the data.

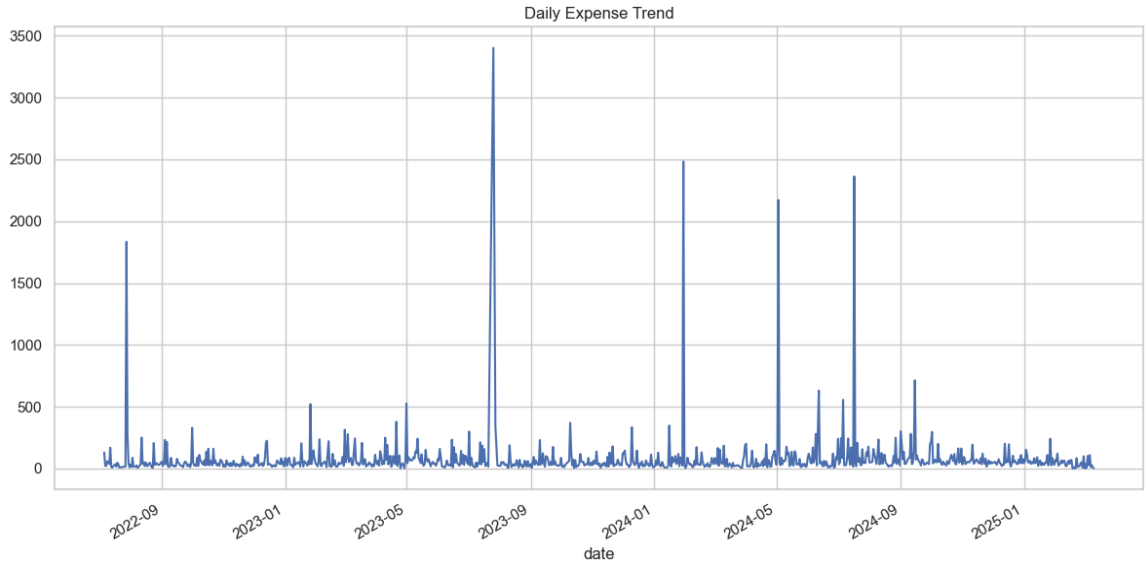


Figure 22: Daily Expenses

4.4.2.2 Anomaly Detection

To improve the accuracy of our time series forecasting, we addressed the presence of outliers unusually large or irregular expense values that can distort model predictions. These anomalies, observed as sharp spikes in the daily expense trend, often represent atypical spending behavior such as major purchases, emergencies, or rare events.

To detect and handle these outliers, we employed an ensemble-based anomaly detection approach, combining the strengths of multiple algorithms. Specifically, we used three complementary techniques:

- k-Nearest Neighbors (KNN): Identifies points that differ significantly from their closest neighbors in feature space.
- Local Outlier Factor (LOF): Measures local deviation of density, flagging points that are less dense than their surroundings.
- Isolation Forest (IForest): An unsupervised learning algorithm that isolates anomalies by randomly partitioning data.

Each of these models assigns an outlier probability score to each data point. We averaged the scores across all models to produce a final anomaly score, offering a more robust and reliable detection

method than any single algorithm. Data points with scores exceeding a certain threshold were classified as outliers.[24]

By removing or adjusting these detected anomalies, we ensured that the forecasting models were trained on data that more accurately reflects regular spending behavior, leading to improved prediction stability and performance.

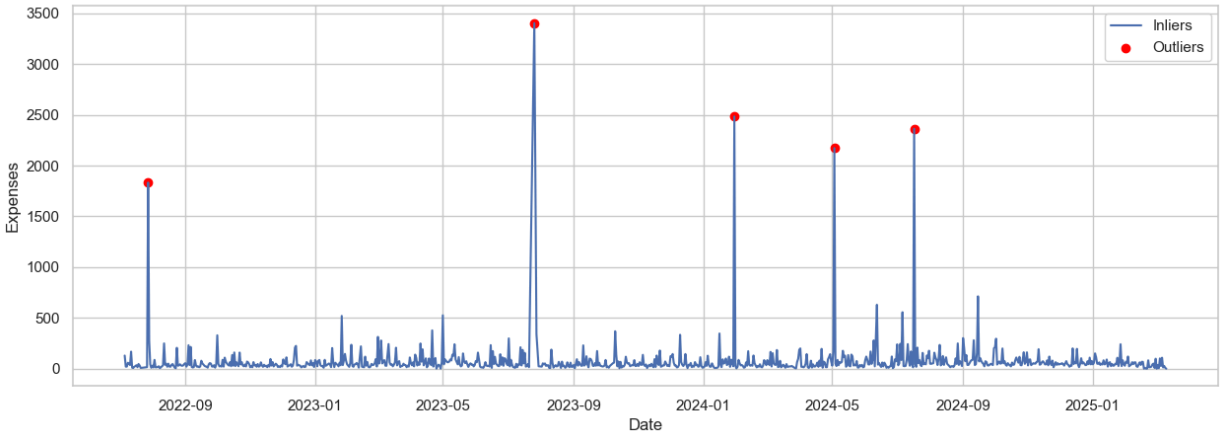


Figure 23: Anomaly detection

4.4.2.3 Time Series Components

To gain a deeper understanding of the structure within our monthly expense data, we applied STL decomposition (Seasonal-Trend decomposition using Loess). Since our goal is to predict monthly expenses, the daily data was aggregated to a monthly level before applying the decomposition. This allowed us to break down the time series into three main components: trend, seasonality, and residuals.[25]

We used a robust version of STL with a periodicity of 3 months to account for short-term seasonal cycles. Below are the interpretations of each component:

1- Seasonal Component

This component captures repeating patterns that occur at regular intervals. In our case, there is a clear seasonal effect every few months, reflecting cyclical changes in user spending potentially due to recurring events such as salary cycles, holidays, or school terms. The amplitude of the seasonal swings suggests that some months consistently involve higher or lower spending.

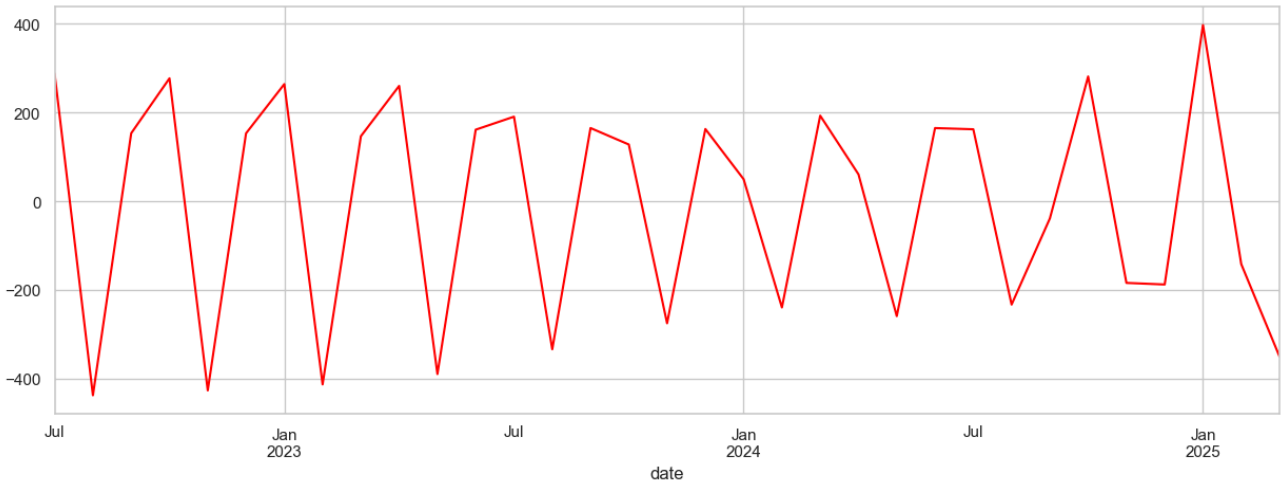


Figure 24: Seasonal component

2. Trend Component

The trend line reveals the long-term progression of monthly expenses. We observe a general increase in spending throughout 2023, followed by a slight decline across 2024 and another rise in late 2024 before dropping again in early 2025. This insight is useful for understanding how user behavior evolves over time and may reflect financial or lifestyle changes.

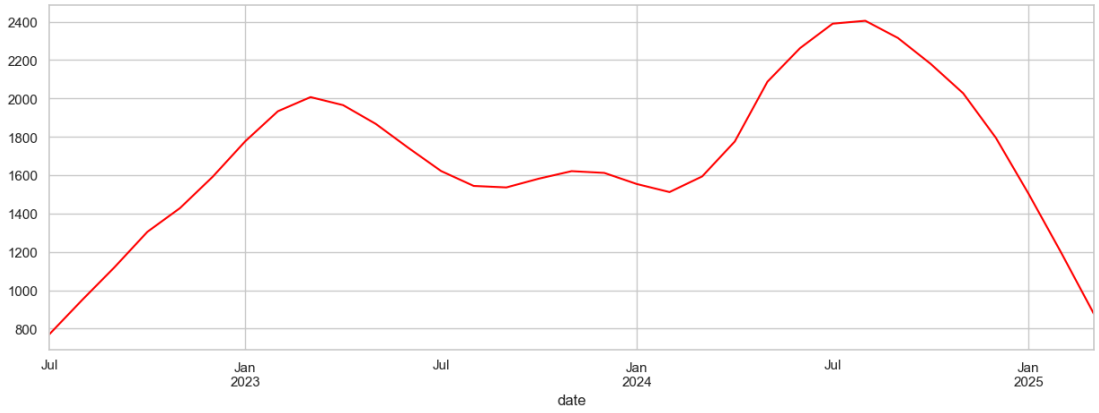


Figure 25: Trend component

3. Residuals

These residuals represent the remaining variation after removing the trend and seasonal effects and after outliers have already been removed. As expected, most values now fluctuate closely around zero, with much smaller spikes compared to the raw data. This confirms that our anomaly detection and cleaning process was effective, leaving only the normal, low-variance noise behind. Such clean residuals are essential for accurate forecasting and model evaluation.

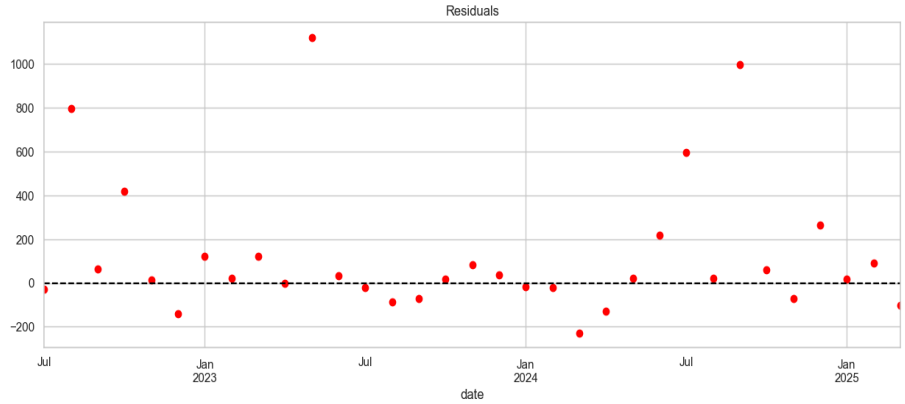


Figure 26: Residuals

Residuals Autocorrelation

After removing outliers and decomposing the time series into trend and seasonal components, we analyzed the residuals to check whether any underlying patterns still remained.

One way to do this is by examining how the residual values relate to their past values over time. This is done through an autocorrelation analysis, which helps identify if the residuals follow any repetitive or dependent structure that was not captured during decomposition.

In our analysis, the results showed that all autocorrelation values fell within the 95% confidence interval. This means:

- The residuals do not exhibit significant relationships with past values.
- There is no remaining structure or pattern in the residuals.
- The decomposition process was effective in isolating the predictable components (trend and seasonality), leaving behind what appears to be pure noise.

This confirms that the residuals are well-behaved and suitable for use in forecasting models, as they no longer carry predictable or systematic information.

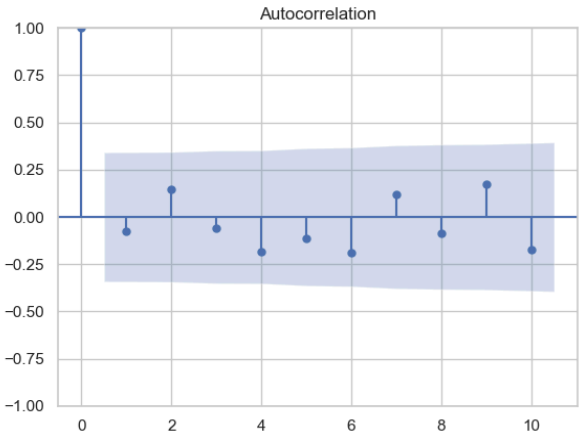


Figure 27: Residuals Autocorrelation

4.4.3 Seasonal AutoRegressive Integrated Moving Average (SARIMA)

To forecast future monthly expenses, we selected SARIMA as our traditional time series modeling approach. SARIMA extends the classical ARIMA model by incorporating seasonal components, making it well-suited for data that shows repeating patterns over time, which was evident in our expense trends.[26]

SARIMA was chosen for several key reasons:

1. **Seasonality Handling**: Unlike standard ARIMA, SARIMA explicitly models both **seasonal and non-seasonal** behaviors, which align well with the clear monthly cycles observed in our expense data.
2. **Flexibility**: SARIMA provides a flexible framework that allows fine-tuning of autoregressive, differencing, and moving average components for both seasonal and non-seasonal patterns.
3. **Proven Accuracy**: It is a well-established model in time series analysis with a strong track record of delivering reliable forecasts for structured and seasonal datasets.

By using SARIMA, we aimed to capture the regularities in user spending behavior across months while also accounting for short-term fluctuations.

4.3.3.1 Stationary test

Before applying SARIMA, it is important to check whether the time series is stationary, a key assumption of the model. A stationary time series has a constant mean and variance over time, which allows the model to better capture and forecast patterns.

Rolling statistics

To evaluate stationarity, we first visually inspected the behavior of the series using the rolling mean and rolling standard deviation with a window size of 3 months: [27]

- The rolling standard deviation remained relatively constant, indicating stable variance over time.
- However, the rolling mean showed fluctuations, with noticeable upward or downward trends in certain periods.

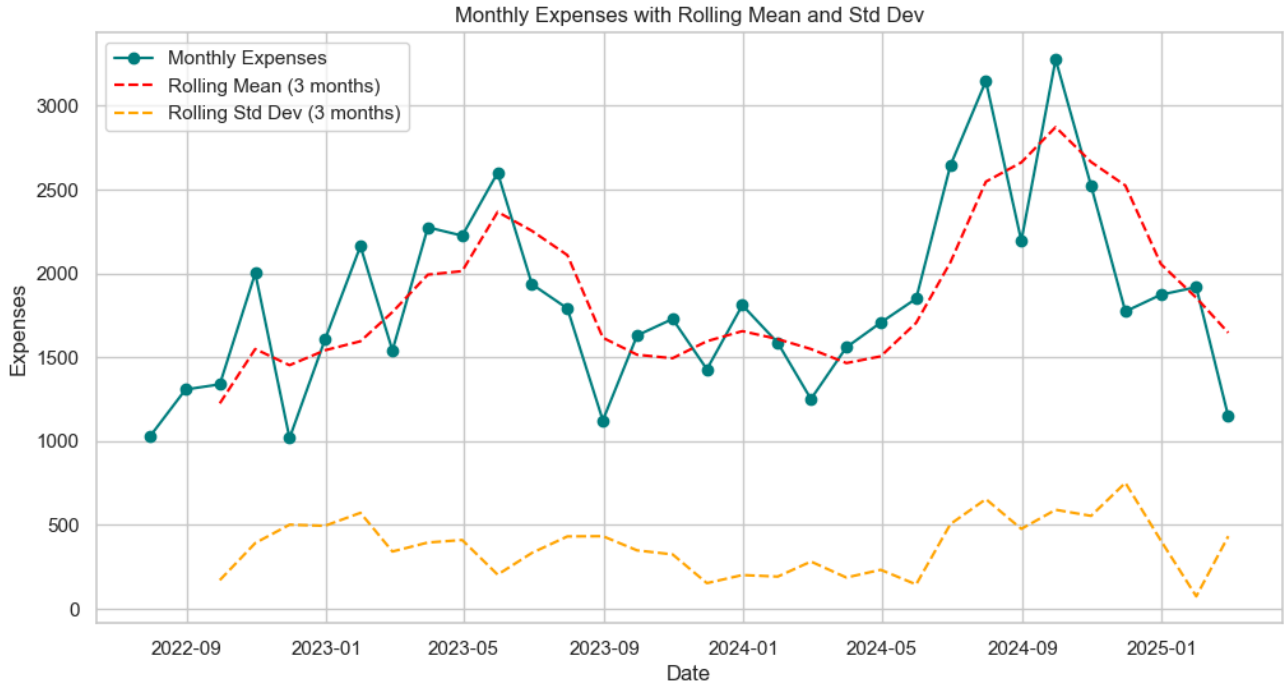


Figure 28: Rolling statistics

This suggests that the series is **not fully stationary**, especially in terms of its mean. To confirm this, we proceeded with a more formal test.

Augmented Dickey-Fuller (ADF) Test

The ADF test is a statistical method used to determine whether a time series is stationary. It specifically tests for the presence of a unit root, which would indicate non-stationarity.

Given the visual evidence of a changing mean, we applied the ADF test to validate our observation. The results confirmed that differencing was needed to stabilize the series before fitting it into the SARIMA model.[28]

ADF Test Results:

- **Test Statistic**: –3.258
- **p-value**: 0.0168
- **Critical Values**: [1%: –3.661 ,5%: –2.961 , 10%: –2.619]

Since the test statistic is lower than the 5% critical value and the p-value is below 0.05, we reject the null hypothesis of the presence of a unit root. This indicates that the series is stationary at the 5% significance level after preprocessing.

With this confirmation, the data is deemed suitable for modeling with SARIMA.

4.3.3.1 Identifying SARIMA Parameters

Once the time series was confirmed to be stationary, the next step was to identify the appropriate parameters for the SARIMA model. SARIMA, short for Seasonal AutoRegressive Integrated Moving Average, extends the classical ARIMA model by adding seasonal terms. It is defined by the following set of parameters: SARIMA(p, d, q)(P, D, Q, s)

- p,d,q for non seasonal part
- P,D,Q for seasonal part

Autoregressive Terms (p and P)
p: Represents the number of past observations (lags) the model uses to predict the current value.
P: Represents the number of seasonal past observations used for prediction.
These values are typically estimated using the **PACF** (Partial Autocorrelation Function) plot[29]

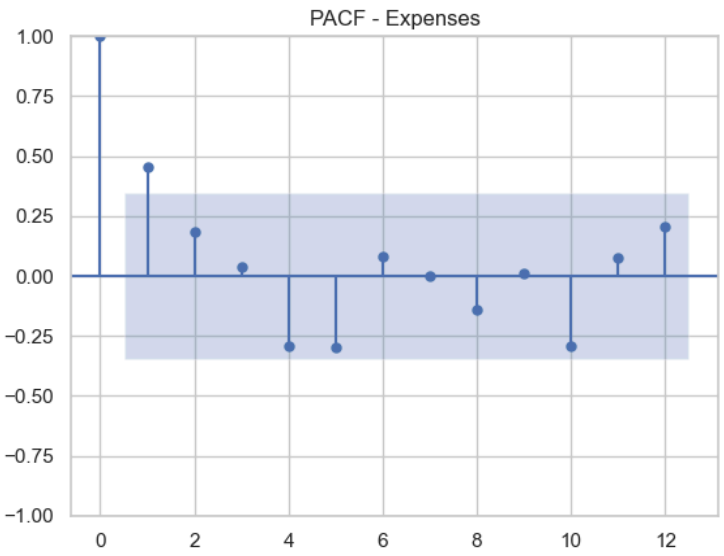


Figure 29: PACF graph

In the Partial Autocorrelation Function (PACF) plot, we noticed a significant spike at lag 1, which is outside the confidence interval. There is a strong correlation between the current month's expenses and the immediately preceding month. This justifies setting the non-seasonal autoregressive term p = 1. No significant spikes were observed at lag 12 (seasonal lag), indicating that P = 0.

Moving Average Terms (q and Q)

q: Number of moving average (MA) terms, captures the influence of past forecast errors.
Q: Number of seasonal moving average (SMA) terms , similar to q, but across seasonal lags.
These values are typically estimated using the **ACF** (Autocorrelation Function) plot [29]

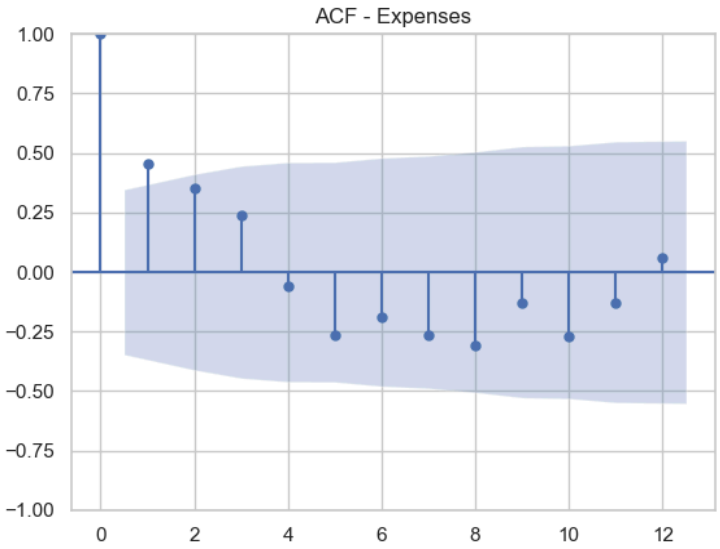


Figure 30: ACF graph

The ACF plot shows only lag 1 as a significant spike, and all other lags (up to lag 12) fall within the confidence interval. This means a strong correlation between current value and the immediate previous value's error and no significant correlation with earlier lags' errors.

- Suggests a Moving Average process of order 1, so: q = 1 (non-seasonal MA term).
- Since no seasonal lags (like lag 12) show spikes: Q = 0 (seasonal MA term).

Differencing Parameters

Stationarity was verified using the Augmented Dickey-Fuller (ADF) test. The test statistic was –3.26 with a p-value of 0.0168, which is below the 5% significance level. This indicates that the null hypothesis of non-stationarity can be rejected, confirming that the series is stationary. As a result, no differencing was required, and we set the non-seasonal and seasonal differencing parameters to d = 0 and D = 0, respectively.

Seasonal Period

The seasonal period parameter s defines the length of the seasonal cycle. Given that the dataset contains monthly expense records, a full seasonal cycle corresponds to 12 months, representing one calendar year. Therefore, the seasonal period was set to s = 12 to appropriately capture any potential yearly seasonality in the time series.

4.3.3.2 Model Implementation

To develop the SARIMA model, we incorporated additional influencing factors as exogenous variables. To prevent data leakage, these variables were lagged by one time step, ensuring that only past information was used for prediction.

The dataset was then divided into training and testing sets, with the final few months held out for evaluating model performance. The SARIMAX model was trained using the identified optimal parameters and fitted to the training portion of the data. The model's performance was then assessed on the test set using the withheld observations and corresponding lagged exogenous variables.

4.3.3.2 Model Evaluation

To evaluate the performance of the SARIMA model, we tested it on the final four months of data that were excluded from training. The model's accuracy was assessed using the **Root Mean Squared Error (RMSE)**, a standard metric that measures the average magnitude of prediction errors.

- **Root Mean Squared Error (RMSE):** 840.15

This value indicates the typical deviation between the model's forecasts and actual monthly expenses. The RMSE provides a clear understanding of the error scale in the context of the original data.

4.4.4 Long-Short Term Memory (LSTM)

LSTM (Long Short-Term Memory) is a type of Recurrent Neural Network (RNN) specifically designed to model sequential and time-dependent data. Unlike traditional feedforward neural networks, LSTM networks are capable of retaining information from previous time steps, which makes them particularly suitable for time series forecasting.[30]

We chose to use an LSTM model to predict future monthly expenses based on past values. The key reason for using LSTM is its ability to capture temporal dependencies and patterns over time, something that classical models like ARIMA or SARIMA may struggle with, especially when the data contains non-linear trends or irregular fluctuations.

Moreover, LSTM helps to overcome the vanishing gradient problem common in standard RNNs, thanks to its special memory cell architecture that selectively retains or forgets information across time steps.

Although our current implementation is univariate relying solely on past expense values it serves as a strong baseline for understanding and forecasting future expenses. Future improvements include adding more input features (category-level spending) to create a multivariate LSTM model, which can further enhance predictive accuracy.

4.4.4.1 Data Preparation

To train the LSTM model effectively, the time series data must first be structured in a way that captures its temporal dependencies. The process begins by isolating the target variable in this case, the monthly expenses and transforming it into a format suitable for sequential modeling.

Since neural networks are sensitive to the scale of the data, we normalize the values using **standardization**, which adjusts the distribution to have a mean of 0 and a standard deviation of 1. This ensures that the model converges more efficiently during training and that each input contributes equally to the learning process.

The dataset is then split into training and testing subsets, with 85% of the data reserved for training. To prepare the input-output pairs for the LSTM, a **sliding window approach** is applied. This technique creates sequences where each input consists of the expense values from the previous three time steps (the last three months), and the corresponding output is the expense for the following month. These sequences are then used to train the model to learn patterns and dependencies over time.

Finally, the structured sequences are converted into a format compatible with the LSTM architecture, enabling it to process time-dependent information effectively.

4.4.4.2 Model Architecture

The LSTM model is designed using a sequential architecture, consisting of multiple layers that progressively extract temporal patterns and non-linear relationships from the time series data.

The first layer is an **LSTM layer** with 64 memory units and configured to return sequences. This allows the model to pass the entire sequence of outputs to the next LSTM layer, enabling deeper temporal abstraction.

The second layer is another **LSTM layer** with 64 units, but this one outputs only the final state. This design helps the network condense the learned sequence information into a compact representation that summarizes the previous time steps.

Following the LSTM layers, a **fully connected (dense) layer** with 64 neurons and ReLU activation is included to introduce non-linearity and enhance the model's ability to capture complex relationships.

To reduce the risk of overfitting, a **dropout layer** is applied with a dropout rate of 20%. This randomly disables a portion of the neurons during training, which encourages the model to generalize better to unseen data.

Finally, the model ends with a **single-node output layer** that predicts the next value in the time series, the forecasted expense for the following month.

The model is compiled using the Adam optimizer for efficient gradient-based learning, and the Mean Absolute Error (MAE) is selected as the loss function. Additionally, Root Mean Squared Error (RMSE) is used as a performance metric to evaluate the average magnitude of prediction errors in the same unit as the original data.

4.4.4.3 Model Training and Early Stopping

To train the LSTM model, the prepared data sequences are fed into the network over multiple iterations (epochs). The model uses a small batch size to update its weights gradually, which often helps capture subtle patterns in time series data.

An **Early Stopping** mechanism is incorporated to prevent overfitting and reduce unnecessary training time. This technique monitors the model's performance on the training loss and automatically stops the training process if the loss does not improve over a defined number of consecutive epochs (in this case, 10). When early stopping is triggered, the model restores the weights from the epoch with the best performance, ensuring optimal generalization.

This setup allows the model to train up to 1000 epochs but typically stops earlier once convergence is detected, leading to a more efficient and robust training process.

4.4.4.4 Model Evaluation

After training the LSTM model, it is essential to evaluate its performance on unseen data to assess its generalization capability. The test set consists of the remaining 15% of the original dataset, with the last three data points from the training set included to provide sufficient context for the sliding window input.

Using the same sliding window approach as in training, sequences of three consecutive months are created from the test data. These are reshaped to match the input format expected by the LSTM network, which requires data in three dimensions: samples, time steps, and features.

The trained model then generates predictions for the test sequences. Since the data was initially scaled for training, the predictions are transformed back to the original scale using the same scaler to ensure a meaningful evaluation.

To quantify the model's accuracy, the **Root Mean Squared Error (RMSE)** is calculated between the predicted and actual expense values. RMSE is a widely used metric for regression tasks, as it expresses the average prediction error in the same units as the target variable. In this case, the LSTM model achieved an RMSE of **510.90**, indicating the average deviation of the model's forecasts from the actual monthly expenses.

4.4.4.5 Multivariate LSTM

The multivariate LSTM model extends the forecasting capabilities of the univariate approach by incorporating multiple relevant features rather than relying solely on past expense values. This model uses historical data from several spending categories such as Education, Entertainment, Fashion, Food, Lifestyle, Transportation, and Health as predictors to estimate future total expenses.

The idea behind using a multivariate approach is to capture the influence of different types of expenditures on overall monthly spending. By providing the model with a broader set of inputs, it gains a more comprehensive view of the underlying financial behavior, enabling it to detect correlations and complex temporal patterns across categories.[31]

The input data is structured using a fixed-size sliding window that looks back over **three consecutive months**. For each time window, the model observes the values of all selected features and learns to predict the total expense in the following month. This design allows the LSTM to process sequential data with multiple variables over time, enhancing its ability to generalize.

The architecture remains based on stacked LSTM layers followed by dense and dropout layers to extract temporal dependencies and avoid overfitting. The model is trained with early stopping to ensure optimal performance without unnecessary training iterations.

After training, the model is evaluated on unseen data using the same windowing structure, and predictions are made for upcoming expenses. The final output reflects the model’s ability to estimate future costs using patterns learned from multiple related financial indicators. The model achieved a **Root Mean Squared Error (RMSE) of 312.49**, indicating a relatively low average prediction error and demonstrating the effectiveness of the multivariate approach in capturing complex dependencies across spending categories.

4.4.5 Models Comparison

To assess the forecasting performance of different modeling approaches, three distinct models were developed and evaluated using Root Mean Squared Error (RMSE) as a common metric:

Model	Input Type	RMSE
SARIMA	Univariate (past Expenses)	**840.15**
Univariate LSTM	Past 3 months of Expenses	**510.90**
Multivariate LSTM	Past 3 months Expenses of 7 Categories	**312.49**

- **SARIMA** captures seasonal and trend patterns in a purely statistical way, but it does not leverage relationships between other spending categories. As a result, it showed the **highest error** among the models.

- **Univariate LSTM** improves performance significantly by learning non-linear temporal dependencies in expense data using deep learning. With RMSE = **510.90**, it outperforms SARIMA by a considerable margin.

- **Multivariate LSTM** incorporates additional features such as **Education, Entertainment, Fashion, Food, Lifestyle, Transportation, and Health**, enabling it to learn richer patterns from **related variables**. This model achieved the **lowest error (RMSE = 312.49)**, highlighting the value of multivariate deep learning for expense forecasting.

4.4.5 Integration of Predictive Modeling in the application

In the context of Savvy application, the **Multivariate LSTM model** is integrated as the primary forecasting engine for users with sufficient historical data. Specifically, the model is applied to users who have been active on the app for **more than three months**, ensuring that enough past data is available to build a reliable prediction window.

How It Works:

- For these users, the model uses the past three months of financial activity including expenses in categories like Food, Transportation, Health, etc. to predict upcoming monthly expenses.
- The forecasted value is then compared directly with the user's actual monthly spending, offering personalized insights into overspending or under-budgeting.
- This comparison is displayed on the home screen, helping users make informed financial decisions.

Why Multivariate LSTM?

By leveraging patterns across multiple spending categories (not just total expenses), the model provides more **context-aware and accurate forecasts**. This allows for a **more meaningful comparison** between predicted and actual behavior enhancing user trust and promoting smarter budgeting habits.

Chapter 5: Frontend & Backend Implementation

5.1 Frontend Implementation

The development of the frontend of our application using Flutter, a modern open-source UI framework by Google. Flutter enabled us to create a high-performance, and cross-platform user interface from a single codebase. Our application incorporates intuitive navigation, reactive state management, and seamless integration with backend services, ensuring a smooth and consistent user experience.[32]

5.1.1 Widget Tree Design

The user interface of the application is structured as a hierarchical widget tree, with the **MaterialApp** widget serving as the entry point. This root widget configures global settings such as the app theme, navigation logic, and initial route. Beneath MaterialApp, the app branches into two primary flows: authenticated and unauthenticated. An intermediate wrapper widget determines the user's authentication status and directs them accordingly. Unauthenticated users are routed to the login or registration interface, while authenticated users are directed to the main interface of the application.[33]

Each screen is composed of a top-level **Scaffold** that provides the basic visual layout structure. This includes built-in support for components such as the **FloatingActionButton** and **SnackBars**, which are integrated directly into the layout system and follow Material Design principles.

Beneath the **Scaffold**, the screen's actual content is arranged using various **layout widgets** such as **Column**, **Row**, **Expanded**, and **ListView**. The choice of layout depends on the structure and behavior of the specific screen whether it's scrollable, responsive, or organized in rows or vertical stacks.

5.1.2 Modular Structure

To promote reusability and maintain a clean, scalable codebase, the application follows a modular development approach at both the UI and project structure levels.

Reusable UI Components

Frequently used interface elements such as buttons, input fields, summary cards, and custom navigation items have been extracted into standalone widgets. This design pattern minimizes code duplication, enforces a consistent visual identity, and improves maintainability across the application.

- **AppBarClipper**: Custom app headers built using **CustomClipper** to create visually distinct, curved top bars.
- **CustomNavBar:** provides a stylized, curved bottom navigation bar using a clipped container and interactive icons, dynamically highlighting the selected tab and handling navigation through a user-defined callback.
- **buildProfileOptions**: Displays user account options such as profile editing, logout, settings and help.
- **buildSettingsOption**: Renders individual settings entries with icons and tap actions.
- **buildIncomeExpenseCard**: Shows summarized income and expense data in a compact, card-like UI.
- **buildSummaryCards**: Dynamically renders the user's total balance, income, and expenses. offering interactive filter-based selection.
- **buildTransactionDetail**: Provides a detailed view of individual transactions, including metadata like description, date and category.
- **buildTextField**: Custom form input field used throughout the app, supporting theming, validation, and icons.
- **showCustomDatePicker:** is a utility function that wraps Flutter's native showDatePicker() with a consistent theme that matches the app's design system. It ensures all date selection interfaces throughout the app use the same style, colors, and behavior. It is used in various parts of the app (e.g., Add Expense, Add Savings).

Modular Folder Structure

All feature-specific screens are organized under (lib/screens), where each subdirectory corresponds to a distinct feature of the app (e.g., authentication, categories, notifications). This ensures that each feature remains isolated and cohesive in terms of UI implementation.

Complementing the screen-based modularity, the project also includes several top-level folders in lib that serve the application globally:

- **widgets:** Contains custom reusable UI components that are shared across features. Examples include buttons, cards, headers, and navigation elements. This promotes a consistent user interface and reduces duplication.
- **models:** Holds application-wide data classes**.**
- **notifiers:** Manages state logic using Riverpod providers. This folder contains both global and feature-specific state managers.
- **services:** Encapsulates logic for communicating with external systems, such as Supabase queries, API calls, and local storage
- **utils:** Provides utility functions, constants, styling helpers, and formatters that assist with various app-wide functionalities like date formatting, amount formatting, and color themes.

5.1.3 Screen Functional Overview

This section provides a detailed walkthrough of the key screens that make up the application's user interface, focusing on their individual functionalities and how they contribute to the overall user experience. Each screen is designed with a specific purpose in mind-ranging from budget tracking and transaction management to user profile settings and analytics.

5.1.3.1 Onboarding

Overview

The **Onboarding screen** is the first point of interaction for new users, designed to introduce them to the app's purpose and benefits in a visually appealing and concise manner. It uses a horizontally swipeable interface with informative slides that guide users through key features of the application, helping establish context and value before transitioning to authentication.

Once users complete the onboarding flow, they are directed to the **Launch Screen**, where they can choose to either log in, sign up, or recover their password. This smooth transition from introduction to authentication ensures a user-friendly first-time experience.

Features

- **Multi-page Introduction:** Uses a PageView.builder to allow users to swipe between onboarding screens. Each page includes a motivational title and a relevant illustration to visually introduce the app's features.
- **Smooth Page Indicator:** An interactive indicator shows the current page and allows users to tap on dots to navigate between pages. It enhances user experience and orientation within the onboarding flow.[34]
- **Launch Screen Options**: Log In, Sign Up.

5.1.3.2 Authentication (Signup-Login)

Overview

The Login/Sign-up screens serve as the initial point of interaction for users, allowing them to authenticate themselves if they already have an account or to register for a new one.

Authentication Methods: Users can log in or sign up using their
Email accounts, providing a seamless and quick access method.

5.1.2.3 Home

Overview

The Home Screen acts as the central hub of the budgeting application, providing users with a personalized welcome, a dynamic financial summary, quick insights, and shortcuts to essential features like analytics, transactions, and chatbot assistance. It is designed for clarity, utility, and engagement, showing only the most relevant and actionable information to users as soon as they log in. The screen is implemented using a PageView , making it easy to swipe between primary app sections (Home, Analytics, Transactions, Categories, Profile), with a custom bottom navigation bar for quick tab switching.

Features

- **Dynamic PageView Navigation:** Powered by PageController for swiping between tabs. Smooth animated transitions or jump-based on distance between pages.
- **Financial Summary:** current month's Income, Expenses, and Balance.
- **Expense Forecast Insight:** Comparison is made between actual expenses and forecasted ones to generate an insight message.
- **Quick Actions:** Three immediate action buttons at the center of the interface (Add Income - Add Expense - View Reports)
- **Chatbot Promotion and Entry:** Displays a short, visually guided explanation of the chatbot's capabilities. Button launches the Chatbot screen via navigation.

5.1.3.4 Chatbot

Overview

The Chatbot module integrates an AI-driven conversational assistant within the application. It supports two core functionalities:

- **"Ask Savvy" mode**: Allows users to query financial insights using natural language.
- **"Transactions" mode**: Enables users to manage and record transactions via text interaction.

This module provides a user-friendly, responsive chat interface, complete with automated scrolling, message history, and bot typing indicators. It enhances engagement by simplifying financial tracking and offering intelligent assistance.

Features

- **Automatic Categorization**: Your input is processed and categorized intelligently into the right expense/income types.

- **Personalized Financial Guidance**: Ask Savvy questions like "How can I save more on groceries?" or "Why did my expenses increase this month?" and receive tailored insights based on your spending behavior.
- **Mode Toggle**: Easily switch between "Ask Savvy" and "Transactions" modes using a toggle at the top of the screen.
- **Smart Typing Detection**: While the bot is responding, it prevents sending more messages to avoid confusion.
- **Auto-Scrolling**: The interface automatically scrolls to the most recent message unless you manually scroll up.
- **Clear Chat Option**: Allows users to reset the conversation and start fresh.

5.1.2.5 Categories

Overview

The Categories module serves as a central part for managing and visualizing spending by category within the personal finance management app. It offers a user-friendly, grid-based layout where users can quickly access detailed insights into different transaction types (expense and income), and savings goals.

Upon opening the Categories screen, users are presented with a color-coded grid of predefined categories (e.g., Food, Health, Transportation), each represented by an icon and label. These categories are managed using a centralized list (globalCategories) that is shared across the app.

Each category tile acts as a navigational entry point:

- **Expense Categories**: Tapping an expense category leads to the **Category Detail Screen**, where users can view the total amount spent in that category for the current month, track their budget progress through a visual progress bar, and add new expenses directly.

  A dynamic LinearProgressIndicator reflects the percentage of the budget used. When spending approaches or exceeds the budget ($\geq$ 90%), visual alerts inform the user to take action.

- **Savings Category**: If the category is marked as CategoryType.savings, the user is directed to the **Savings Goals Screen**, which allows them to create and manage savings objectives with deposit tracking.

- **Income Category**: The screen designed for allowing users to view and select from a list of **predefined income categories**. It serves as an entry point for users to add a source of income.

- The **Add Expense/Income Screen** offers a streamlined form for logging new expense entries:
    - Pre-selects the category if passed via navigation and auto-fills today's date.
    - Users can choose a custom date using a date picker, enter an amount, and optionally include a description.
    - Upon submission, the expense is validated, structured, and saved through the transactionProvider, which writes to Supabase.

5.1.3.6 Categories goals

Overview

The Goals module allows users to track savings progress toward predefined financial objectives (such as Travel, House, Vehicle…). Users can select a goal from a dropdown list of predefined goal templates, each associated with a recognizable icon and purpose.

Once selected, users can customize the target amount and incrementally add deposits toward their savings. Each goal tracks the amount saved, visualizes the progress, and keeps a history of contributions. This feature promotes disciplined saving by providing motivation through progress tracking and visual feedback.

All goal-related data is stored and synced via Supabase, ensuring persistence across devices and sessions.

Features

- **Saving Goals Screen:**
  Lists all savings goals defined by the user.
    - Displays each goal with its name, icon, and current saved amount.
    - Shows total savings across all goals.
    - Allows refreshing to fetch updated goal data from Supabase.
    - Enables deleting goals, which updates the database and UI state.
    - Dropdown selector to add a new savings goal from a list of predefined templates (e.g., Travel, House, Emergency).

- **Add Savings Screen:**
  Allows the user to contribute to a specific goal by:
    - Entering a deposit amount.
    - Optionally editing the goal's target amount (if not previously set).
    - Selecting the date of deposit via a custom calendar widget.
    - Validates input and updates both the local state and Supabase.

- Displays a progress bar showing how close the user is to reaching the goal.

- **Goal Detail Screen:**
  Shows detailed information about a selected goal:
  - Target amount.
  - Total amount saved.
  - Visual progress indicator (circular).
  - List of previous deposits, sorted by date.
  - Supports adding new deposits and editing the goal amount.
  - Automatically recalculates and displays progress after each change
    - Goal progress is computed as: progress = savedAmount / targetAmount
    - Where savedAmount = The total amount the user has already saved toward that goal.
    - targetAmount = The total amount the user wants to save.

5.1.3.7 Profile

Overview

The **Profile Screen** allows the user to view and interact with key account-related functionalities. The design features a clean layout with a profile avatar, user name, and interactive options.

Features

- **View Profile Info:** Full name and profile picture.[35]
- **Edit Profile:** Navigate to update name, photo, or other details.
- **Settings Access:** Notification Settings, Password Settings and Delete Account
- **Help Section:** Link to a help/FAQ or support interface.[36]
- **Logout Option:** Prompts confirmation before securely logging the user out.Clears session and all relevant cached providers. Navigates back to the onboarding screen.

5.1.3.7 Notifications

Overview
The NotificationsScreen provides users with a centralized place to view, manage, and delete their app notifications. These notifications may include budget reminders, low balance alerts, or custom daily reminders. The screen fetches saved notifications from **SharedPreferences** [37] via a custom **NotificationService**.[38][39]

Features

- **View Notifications:** Displays a list of saved notifications with Title, Body message and Timestamp
- **Delete All Notifications:** clears all saved notifications via NotificationService
- **Delete Individual Notifications:** Each notification has a red delete icon allowing users to remove it individually.

Notification functionality

The Notification is a core **feature module** in the app that manages how users receive and interact with notifications. It handles all logic for **sending, scheduling, saving, and removing notifications** related to budgeting such as daily expense reminders, budget limit warnings, and low balance alerts. It ensures that users stay informed and engaged with their finances while also giving them a way to view or clear past notifications.

Features

- **Initialize Notifications**: Prepares the notification system for both Android and iOS when the app starts.
- **Permission Handling**: Requests and checks system-level notification permissions from the user.[40]
- **Show Notifications**: Displays instant local notifications with custom titles and messages
- **Budget Limit Alerts**: Sends a notification when a spending category crosses its budget.
- **Low Balance Warnings**: Notifies users when their overall balance drops too low.
- **Daily Expense Reminders**: Schedules daily recurring notifications to encourage logging expenses.
- **Smart Notification IDs**: Uses custom ID ranges for budget and balance alerts to manage or cancel them later.
- **Save Notification History**: Stores delivered notification data (title, body, time, status) using SharedPreferences.
- **Load Notification History**: Loads saved notifications to show on the Notifications screen.
- **Delete Notifications**: Allows removal of all or specific saved notifications.
- **Cancel Notifications**: Cancels scheduled, budget-related, or all notifications as needed.

5.1.3.8 Transactions

Transactions Screen

Overview
provides users with a snapshot of their financial activity, showcasing the total balance, income, and expenses. It displays a scrollable list of recent transactions, grouped by type, and supports dynamic filtering and smart categorization. The screen is designed to be fast, readable, and visually clear.

Features

- Summary Cards
  - **Total Balance**: Shows overall balance across all transactions.
  - **Income Card**: Total income with upward arrow and green highlight.
  - **Expenses Card**: Total spending with downward arrow and red highlight.
  - Tap a card to filter transactions by type (all, income, or expense).
- Transaction List
  - Displays the **4 most recent** filtered transactions. Includes:
    - Description or category name
    - Transaction date (formatted YYYY-MM-DD)
    - Category label (auto-detected)
    - Colored amount (green for income, red for expense)
  - Auto-sorts transactions by created_at (most recent first).
- FAB navigates to the full **transaction history screen**

Transactions history screen

Overview
Transaction history screen watches the transactionProvider, listens for changes (e.g. new data, user input), and displays a **searchable, grouped-by-month list of transactions**.

Features

- **Search Feature:** allows users to search for transactions by typing keywords.
- **Month-Based Grouping:** Each group includes:
- Header month-name year
- **List of transactions** under that month:
  - category name
  - Date and type (Income/Expense).
  - Amount with color based on type.
  - Tap to navigate to the detail screen where the user can edit transaction details.

- Delete option for each transaction.

5.1.3.9 Analytics

Overview

The Analytics Module provides users with an interactive and visual breakdown of their financial activity across various time periods. This includes income, expenses, net savings, and spending distribution by category. By switching between daily, weekly, and monthly views, users gain insight into their financial habits, detect spending patterns, and evaluate their progress toward savings goals.

This module enhances financial awareness and supports informed decision-making through dynamic visuals and real-time data sourced from Supabase.

Features

- **Period Tabs**
  Users can toggle between **Daily**, **Weekly**, and **Monthly** tabs to analyze financial data for the respective time periods.

- **Summary Cards**
  Displays animated values for:

  - **Total Income**
  - **Total Expenses**
  - **Net Savings** (calculated as Income - Expense)

These cards quickly show how much users earned, spent, and saved over the selected period.

- **Bar Chart**
  - Dual bars (Income and Expense) per time unit (day, week, or month).
  - Helps users compare income vs expenses and spot trends, spikes, or irregularities.
  - Includes tap tooltips to display the exact EGP amount per bar.

- **Pie Chart (Spending Category Breakdown)**
  - Shows percentage breakdown of expenses by category on a daily basis.
  - Helps users understand which areas consume most of their daily spending.

- **Goal Progress Visualization**
  - A circular ring displays the percentage progress of each savings goal toward its target.
  - Goals are also listed with current savings and visual indicators.

Calculations & Logic

Net Savings = Total Income - Total Expense

Period Ranges:

- Daily: Last 7 days
- Weekly: Last 4 weeks
- Monthly: Last 6 months

Bar Scaling Logic: Bar chart height dynamically adjusts to data: maxY = highest(income or expense) * 1.2 → Adds 20% buffer above tallest bar for clarity.

Pie Chart Condition Rendered only when daily data includes expenses.

User-Specific Data: All transactions are fetched using the logged-in user's user_id via Supabase queries.

Charting Package: fl_chart

Purpose: To render interactive and animated financial visualizations such as bar charts and pie charts in the Analytics module.[41]

Why fl_chart?

- Flutter does **not have built-in chart widgets**
- fl_chart is:
    - Lightweight and efficient
    - Highly customizable
    - Supports touch gestures, animations, tooltips, and responsive layouts
    - Actively maintained and popular in the Flutter community

Key Features Utilized:

- **Dynamic Scaling**: Bar heights scale based on the highest income or expense (maxY * 1.2)
- **Tooltip Interaction**: Tapping a bar shows a tooltip with the exact amount
- **Pie Chart Breakdown**: Shows category-wise distribution of expenses for daily analytics
- **Custom Styling**: App color palette used for consistency

5.1.4 State Management

The app uses **Riverpod** [41] as its state management solution. Riverpod provides robust and scalable dependency injection and state management capabilities, making it ideal for managing asynchronous data such as user info, transactions, budgets, and forecasts.

- Authentication and User Info
  - authServiceProvider: A basic provider that instantiates the custom AuthService.
  - currentUserProvider: Returns the current authenticated Supabase user
  - userProvider: **StateNotifierProvider** that exposes the user's profile data asynchronously.
- Transactions
  - transactionProvider StateNotifierProvider that holds a list of all user transactions.
  - Managed by TransactionNotifier, which: Fetches initial transaction data, subscribes to real-time updates via Supabase.
- Financial Summary
  - financialSummaryProvider: StateNotifierProvider managing a FinancialSummary object. Handled by FinancialSummaryNotifier, which fetches aggregated financial data (e.g., total income, spending), Subscribes to updates for live changes.
- Budgets
  - budgetProvider: StateNotifierProvider for the list of budgets Uses BudgetNotifier, and automatically fetches the user's budget data if a user is logged in. Depends on currentUserProvider to get the user ID.
- Forecasting (ML Prediction)
  - forecastProvider: FutureProvider.autoDispose.family that fetches a forecast from an external API for a given userId, supports per-user forecasts (via .family modifier), automatically disposes itself when no longer used, but keeps alive during usage.

.

5.2 Backend Implementation

5.2.1. Introduction

The backend is the **central engine** of the application, handling the logic that powers the system's features and ensuring all data-related operations are carried out with precision and efficiency. It manages tasks such as handling financial data, performing real-time calculations, and responding to requests from the client side through structured and secure API endpoints, the backend supports real-time functionalities like calculating a user's total monthly spending, forecasting their future savings based on past transactions, and assessing whether their current spending habits align with their predefined budgets and goals.

A key responsibility of the backend is to uphold operational integrity across all financial modules. This includes enforcing constraints such as preventing users from exceeding budget limits, verifying goal milestones before updating progress, and validating transactions to ensure category alignment with user-defined budgets To ensure long-term scalability, clean code organization, and ease of maintenance, the backend is built using **FastAPI**, a modern Python web framework optimized for building RESTful APIs. FastAPI brings several advantages:

- **Asynchronous I/O** support allows the application to handle many requests simultaneously without blocking other operations. This results in faster response times and better performance, especially during high traffic.
- **Interactive documentation** via built-in Swagger UI and ReDoc interfaces, which improves developer productivity and eases integration with frontend components.
- **Automatic data validation** using Pydantic models ensures that only correctly structured data enters the system.

The project follows the principles of Clean Architecture, a software design approach that separates the system into distinct layers, each with a specific role:

- **Domain Layer**: Contains the core business entities and logic, completely independent of frameworks and libraries.
- **Service Layer**: Implements use cases, coordinating between repositories and external services to fulfill business needs.
- **Infrastructure Layer**: Handles interactions with external tools, such as the database or authentication providers.
- **API Layer**: Defines the HTTP endpoints exposed to the frontend, processes incoming requests, and returns structured responses.

Because each layer operates independently, the system can evolve naturally, as new features are added. This means enhancements such as AI-powered spending insights, machine learning-based transaction categorization.

5.2.2 FastAPI

5.2.2.1 Overview

FastAPI serves as the core framework for developing the backend of the system, delivering exceptional performance, clean code architecture, and developer efficiency. It is built atop two powerful libraries:[42]

- **Starlette** is a lightweight ASGI (Asynchronous Server Gateway Interface) framework that enables high-speed asynchronous web services, including WebSocket support and background task execution.
- **Pydantic** is a data validation and settings management library that enforces strict type constraints and automatic parsing of request and response data using Python type hints.

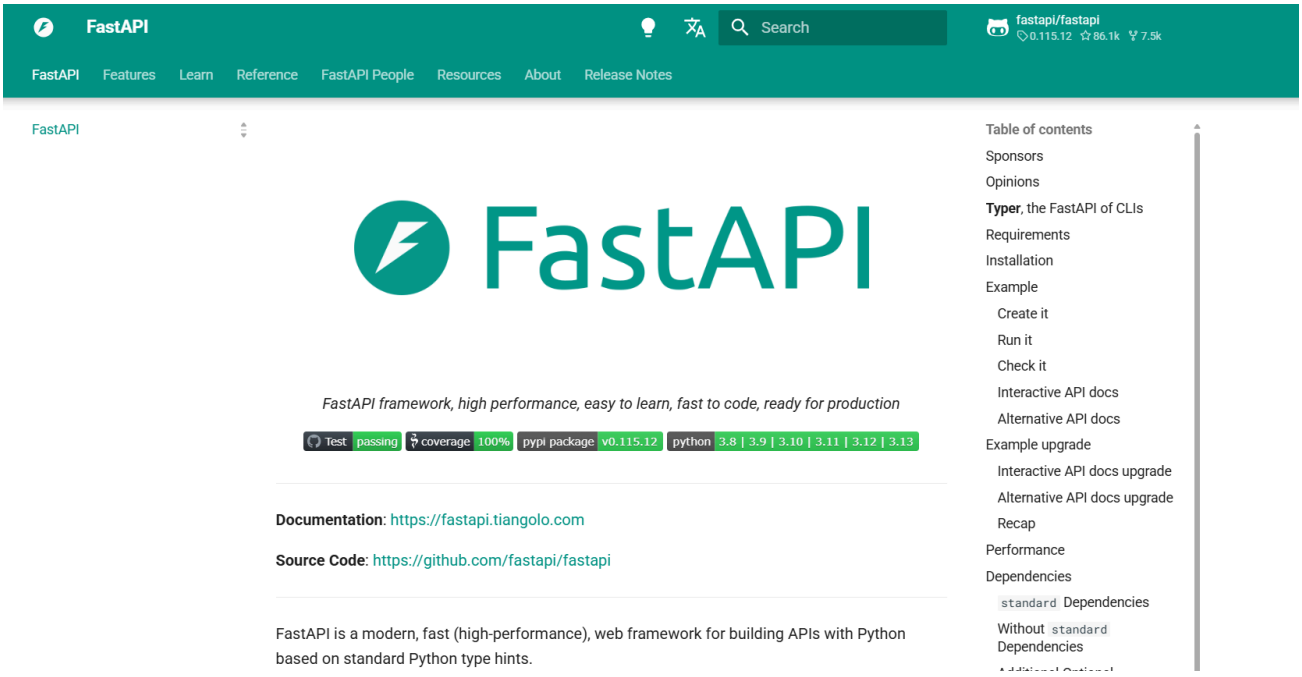
These foundations make FastAPI highly suitable for modern, API-driven systems where data accuracy, concurrency, and rapid development are critical, particularly in financial applications that demand both performance and precision. Unlike traditional synchronous frameworks, FastAPI fully supports asynchronous programming, allowing the backend to process multiple client requests in parallel without blocking the main thread.

FastAPI also integrates seamlessly with Pydantic to provide automatic request and response validation. By defining data models with clear type annotations, the system ensures that all inputs (e.g., transaction submissions or goal updates) are validated at the API layer before reaching business logic or storage. This minimizes bugs, enhances data integrity, and simplifies error handling during development. Furthermore, FastAPI enhances developer productivity and maintainability through:

- **Auto-generated interactive documentation**, available through Swagger UI and ReDoc, which provides live previews of all available endpoints, request schemas, and response formats.
- **Built-in dependency injection**, which simplifies configuration management and supports clean, modular code design.
- **Compatibility with modern tools** such as asynchronous ORMs, task queues, third-party APIs, and microservices, making it highly adaptable to both monolithic and distributed system designs.

FastAPI's lightweight and async-first nature also enables future extensibility. The backend is well-positioned to support additional modules such as:

- Real-time updates via WebSockets (e.g., budget alerts or goal progress).
- Machine learning endpoints for intelligent financial insights.
- NLP-driven transaction classification or anomaly detection.



5.2.2.2 Features

FastAPI offers a rich set of features that make it a powerful and efficient choice for developing modern backend systems. These features not only accelerate development but also enhance the reliability, scalability, and maintainability of the application. the key FastAPI features leveraged in our backend implementation :

- **High Performance**

  FastAPI is one of the fastest Python frameworks available, comparable to Node.js and Go in terms of raw performance. It is built on ASGI (Asynchronous Server Gateway Interface) using Starlette, enabling highly efficient asynchronous request handling. FastAPI's performance allows the backend to process thousands of concurrent requests, ideal for financial applications with real-time user interactions like budget updates, transaction processing, and analytics dashboards.

- **Asynchronous Support**

  FastAPI is designed with full support for asynchronous I/O using Python's async and await syntax. This enables the backend to perform non-blocking operations such as:

  - Concurrently handling multiple user sessions.
  - Triggering background tasks like periodic financial reports or goal progress updates.
  - Integrating with async-compatible services and APIs.

  Enables better scalability, improved response times, and efficient system resource utilization, especially during high-traffic periods or complex data processing tasks.

- **Data Validation with Pydantic**

  FastAPI uses Pydantic models for parsing, validating, and serializing request and response data. This enforces strict typing and automatically rejects malformed or incomplete data at the API layer. This is critical in financial systems where incorrect or missing values in a transaction or goal update could result in significant downstream errors. Pydantic ensures all payloads, such as income records or category updates, are verified before being processed.

- **Automatic Interactive API Documentation**

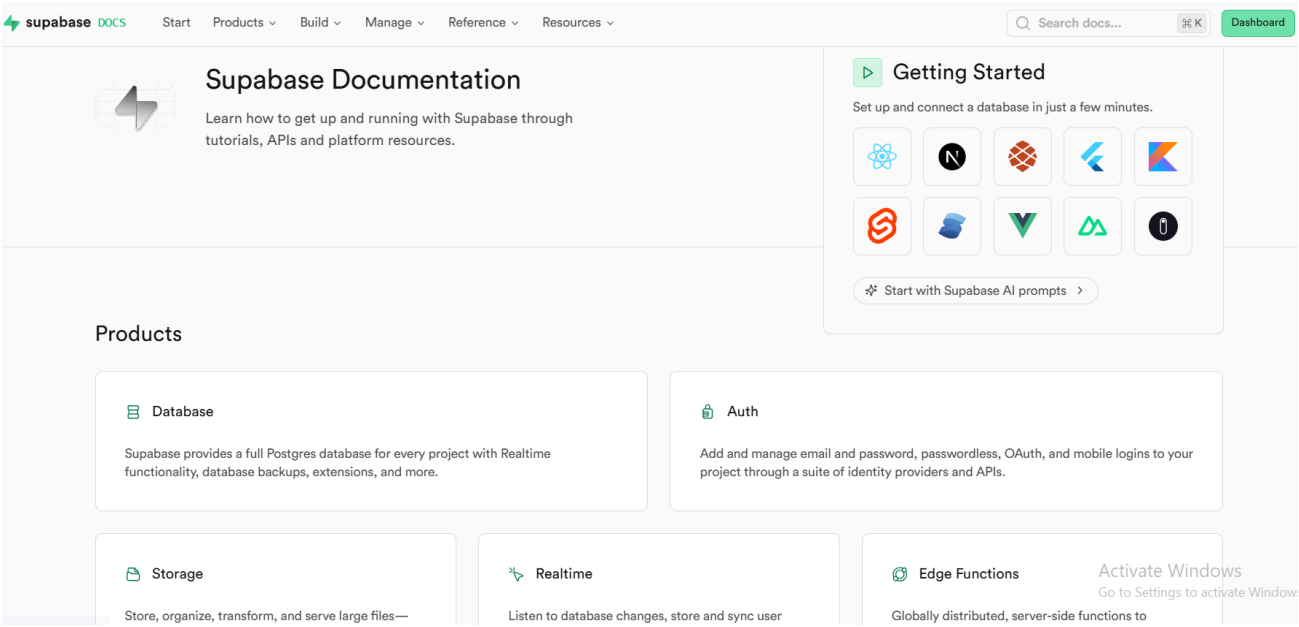
  FastAPI automatically generates **OpenAPI** documentation for every endpoint defined in the application. It provides two powerful, built-in tools:

  - **Swagger UI**: A dynamic, browser-based interface where developers and testers can explore and interact with the API in real-time.
  - **ReDoc**: A clean, structured documentation interface ideal for internal team references and third-party developers.
  - **Exportability**: Each API route displays the HTTP method, path, description, parameters, request/response schemas, and example payloads.
  - **Try-it-out mode**: Allows developers to send requests directly from the browser using live data, which is incredibly useful for manual testing and debugging.
  - **Onboarding new team members**: The self-explanatory docs help new developers or frontend engineers quickly understand how to interact with the backend without needing to dive into the source code.

5.2.3 Database Management with Supabase

5.2.3.1 Overview

Supabase is an open-source Backend-as-a-Service (BaaS) platform that sits on top of PostgreSQL, providing developers with scalable, production-ready infrastructure out of the box. It offers a rich suite of features, including a real-time database, authentication, storage, and edge functions, all accessible via RESTful and GraphQL APIs. For **SavvyAPI**, Supabase served as the core data management engine, enabling building and iterating quickly without worrying about complex backend setup or server maintenance. By leveraging Supabase, we gained full SQL flexibility and strong data integrity through PostgreSQL, while also benefiting from built-in user authentication, row-level security (RLS), and fine-grained access controls. This allowed us to maintain robust data security and personalized user experiences at scale. The real-time capabilities further enhanced user engagement by enabling dynamic updates to transaction history, goals, and budgets without page refreshes, Supabase empowered SavvyAPI to deliver a full-featured financial platform with modern backend capabilities, all while keeping the development process lean, efficient, and future-proof.[43]



5.2.3.2 Features

- PostgreSQL Database (Core Engine)

  Supabase offers a fully managed PostgreSQL instance, widely regarded as the most powerful open-source relational database system. It provides robust features such as relational data modeling with foreign keys and constraints, transactional integrity (ACID compliance) for reliability under concurrent operations, advanced querying capabilities including joins, subqueries, CTEs, full-text search, JSON fields, and time-series operations, as well as extensibility through PostgreSQL extensions like **pgcrypto and uuid-ossp**

  In **SavvyAPI**, we leverage Supabase PostgreSQL to model and store all core entities in our financial ecosystem: **users, transactions, budgets, goals, and predictions**. The relational structure ensures consistent linkage between entities; for example, each transaction is associated with a specific user and optionally linked to a goal. Timestamps are indexed to support efficient analytics and time-based operations, enabling advanced insights and forecasting.

- **Row-Level Security (RLS)**

  Row-Level Security is a PostgreSQL feature that Supabase enables by default to enforce fine-grained, row-level access control directly within the database, crucial for securing multi-user applications. With RLS, you can define SQL-based policies that control which rows a user can **read**, **insert**, **update**, or **delete**, based on attributes like their user ID. These policies are enforced automatically on **every request**, including those made through Supabase's REST and GraphQL APIs, ensuring secure access without requiring custom logic. RLS offers stronger protection than relying solely on application-level checks, as it guards against unauthorized access attempts even when querying the database directly.

Ex: Policy Allow users to view only their transactions:

```
CREATE POLICY "Users can only access their own transactions"
  ON transactions
  FOR SELECT
  USING (user_id = auth.uid());
```

In **SavvyAPI**, we use RLS to ensure that every financial record, transactions, budgets, goals—can only be accessed or modified by the authenticated user who owns it. This maintains strict data privacy and security across the entire application.

- **SQL Editor & Admin UI**

  The Supabase dashboard provides a powerful and intuitive interface for managing PostgreSQL database, featuring a built-in SQL editor and visual tools that streamline development and administration.

  - Easily create and modify tables, columns, constraints, indexes, and triggers through the UI.
  - Execute raw SQL queries directly within the built-in editor for advanced operations or data inspection.
  - Explore schema visually using the interactive **Entity-Relationship Diagram (ERD)** to understand relationships and dependencies.
  - Manage critical settings such as API keys, user roles, RLS policies, and authentication rules in a centralized interface.

  In **SavvyAPI**, the Admin UI significantly accelerated early-stage development. It allowed us to rapidly iterate on the schema, defining unique constraints, indexing **created_at** fields for time-series efficiency, and applying ON DELETE CASCADE rules to maintain relational consistency, all without writing custom migration scripts.

- **Supabase Storage (File Management)**

  Supabase Storage is a scalable, policy-controlled object storage solution tightly integrated with the Supabase ecosystem. It enables seamless uploading and serving of files such as images, receipts, invoices, and exported reports. Files are organized in logical "**buckets**" and can be accessed securely via public or signed URLs. Access to stored files is governed using the same **Row-Level Security (RLS)** policy system applied to tables, ensuring consistent and secure permissions across both data and assets.

  In **SavvyAPI**, Supabase Storage is uploading avatar images tied to user profiles (via the **avatar_url** field), attaching receipts to transactions for enhanced personal bookkeeping, and exporting financial reports as downloadable PDFs for organized recordkeeping.

5.2.3.3 Database Schema Design

We implemented a **normalized and relational database schema** in Supabase to support SavvyAPI's core financial functionalities, **user management, transaction tracking, budgeting, goal setting, and analytics**. The schema is designed for scalability, data consistency, and compatibility with Row-Level Security (RLS), ensuring secure multi-user data isolation.

- Tables & Structure
    1. Users: Manage authentication and profile details. Stores unique user accounts and personal information

Column	Type	Description
user_id	UUID	The primary key uniquely identifies the user
email	varchar	Unique user email for login
full_name	varchar	User's full name
phonenumber	varchar	Optional contact info
hashed_password	varchar	Encrypted password for authentication
avatar_url	varchar	URL to profile image
created_at	timestamp	Account creation time
updated_at	timestamp	Last profile update

    2. Transactions: Track income and expense. Captures individual financial entries with metadata for categorization and analysis**.**

Column	Type	Description
transaction_id	UUID	Primary key
user_id	UUID	Foreign key linking to users
category_id	UUID	Foreign key linking to categories

description	text	Optional note or label
created_at	timestamp	Date and time of transaction
amount	float8	Positive or negative monetary value
transaction type	text	Type: "income" or "expense"

3. Categories: Organize transactions and budgets Store both predefined and user-generated category labels for grouping financial data.

Column	Type	Description
category_id	UUID	Primary key
category_name	varchar	Descriptive label (e.g., "Rent")

4. Budgets: Define spending limits. Enables users to set budget allocations for specific categories.

Column	Type	Description
budget_id	UUID	Primary key
user_id	UUID	Foreign key linking to users
category_id	UUID	Foreign key linking to categories
allocated_amount	float8	Maximum allowable spending

5. Goals: Supports financial planning and saving. Allows users to define and track long-term financial objectives.

Column	Type	Description
goal_id	UUID	Primary key
user_id	UUID	Foreign key linking to users

goal_name	text	Goal title (e.g., “Vacation Fund”)
target_amount	float8	Total savings target
amount_saved	float8	Progress made toward the goal
created_at	date	Date the goal was created

The schema follows 3rd Normal Form (3NF) design principles, reducing data redundancy and ensuring data integrity:

- All financial entities (transactions, budgets, goals) reference the users table via user_id, providing a clear ownership structure for enforcing RLS policies and ensuring strict user-level data isolation.
- Both transactions and budgets are linked to the categories table through category_id, enabling consistent grouping and analysis across different modules.
- Time-based fields such as created_at are indexed to support efficient time-series queries and analytics, a key feature in forecasting and trend analysis.

5.2.4 Clean Architecture

5.2.4.1 Overview

Clean Architecture is a software design **approach** that emphasizes long-term modularity, and testability by enforcing a strict separation of **concerns**, particularly between core business logic and external systems such as frameworks, databases, and third-party services. In the context of **SavvyAPI**, adopting this architectural pattern has been instrumental in ensuring that the backend remains scalable, maintainable, and adaptable to future technological changes.

Clean Architecture asserts that the most critical part of any application, its **business rules**, should remain unaffected by **how data is stored, how APIs are delivered,** or which libraries or tools are used. It views external dependencies as interchangeable implementation details. This separation is achieved through a concentric, layered code structure, where each inner layer holds more abstract, self-contained logic, and outer layers depend only on the layers directly beneath them. Critically, the flow of dependencies moves inward; inner layers remain completely independent of outer ones, creating a highly decoupled system.

Most importantly, this architecture lays a strong foundation for **future growth**. As SavvyAPI evolves to support advanced features, such as analytics dashboards, AI-powered financial insights, shared budgeting among users, or PDF report exports, the modular, decoupled structure ensures that new capabilities can be added without disrupting existing functionality. Each feature can be introduced as a self-contained component that plugs into the system through contracts and interfaces, not brittle dependencies.

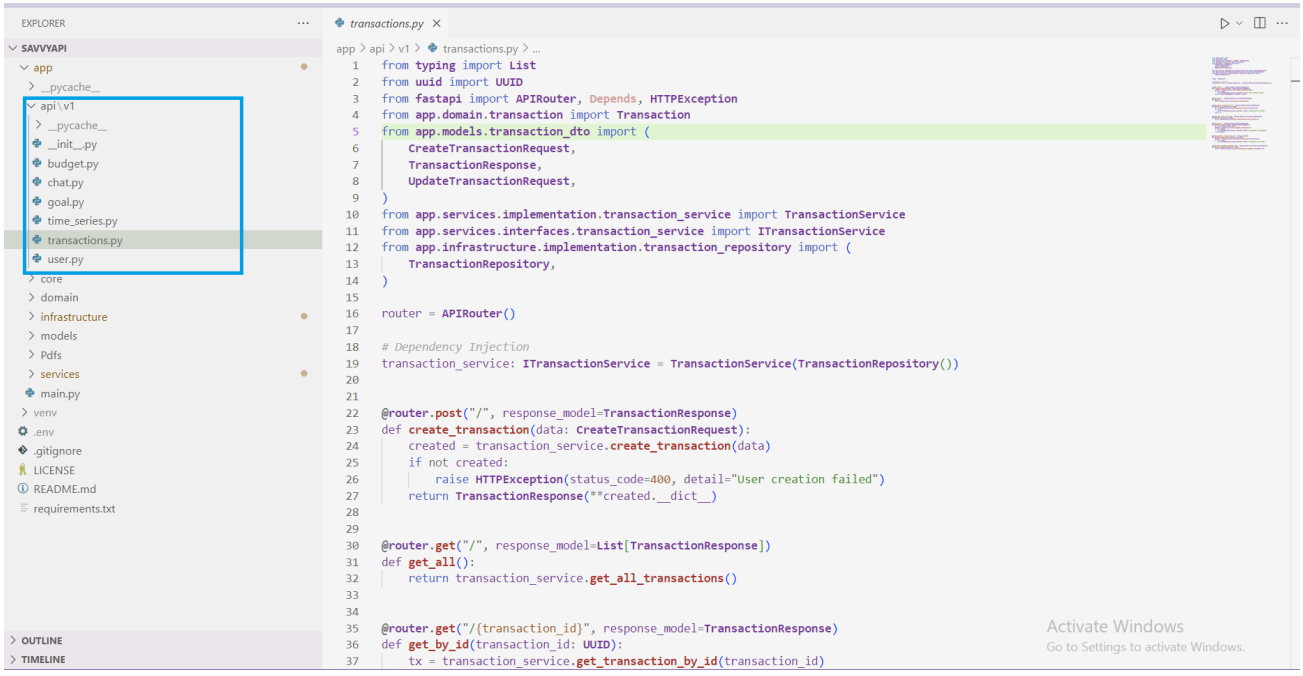
5.2.4.2 Clean Architecture Layers

- **Presentation Layer**

The **Presentation Layer** in SavvyAPI backend is the entry point to the system. It is responsible for handling all **external communication**, receiving HTTP requests, validating data, forwarding requests to the service layer, and returning appropriate responses. This layer is implemented using **FastAPI**, and it defines the routes for various modules like users, transactions, budgets, and goals. In Clean Architecture, the presentation layer is kept **lightweight** and free from business logic or database interactions. Its only job is to **translate client input into service calls** and **package the output for client delivery**.

The presentation layer uses **Dependency Injection** is a core principle that helps enforce the decoupling between layers in Clean Architecture. Instead of hardcoding dependencies inside a class or function, DI allows them to be passed in from the outside—usually via constructor parameters or function arguments.

In **SavvyAPI**, this allows each component to depend on **interfaces (abstractions)** rather than concrete implementations. This promotes testability, flexibility, and modularity.

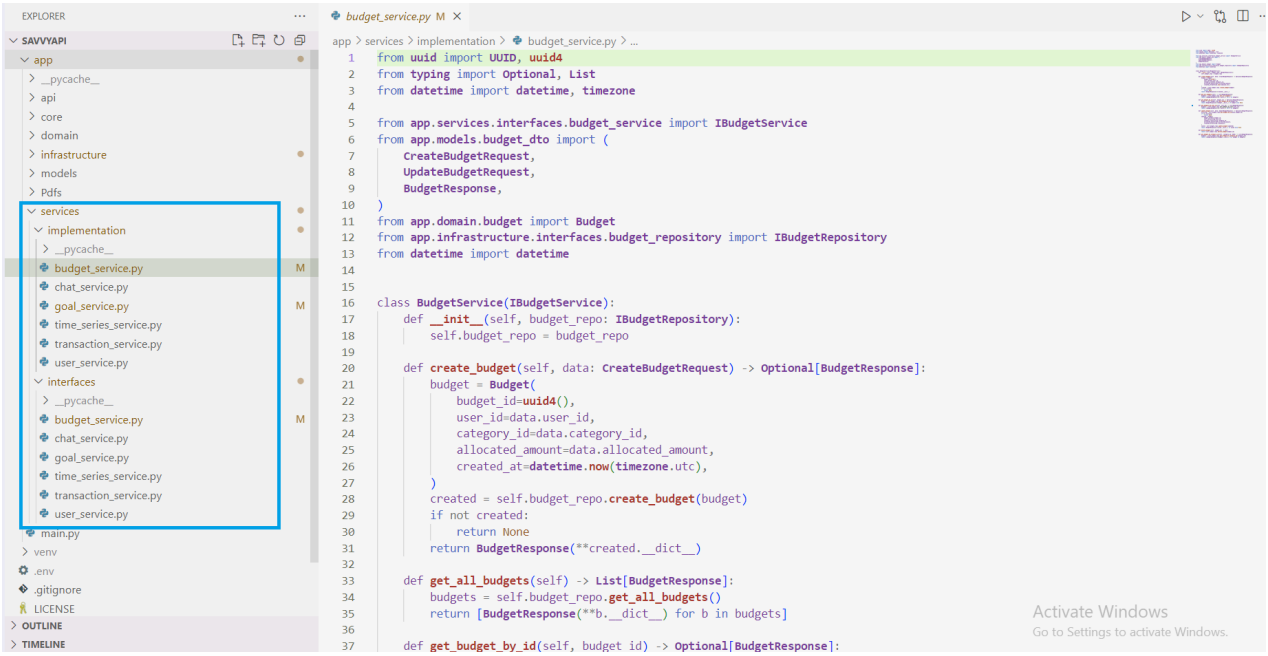


- **Service Layer**

The **Service Layer** acting as the primary engine for executing **business logic** and **application use cases**. It serves as an intermediary between the **Presentation Layer** (which handles HTTP requests and user interactions) and the **Infrastructure Layer** (which handles data persistence and external integrations). This layer enforces the rules and workflows defined in the **Domain Layer**, ensuring that the application's core logic remains consistent and protected.

In SavvyAPI architecture, the Service Layer ensures that every feature, from tracking transactions and managing budgets to evaluating goal progress, follows clearly defined logic and constraints. When a request arrives from the client, this layer validates the input against domain requirements, coordinates with the appropriate repositories or services, transforms or enriches the data as needed, and returns structured **DTOs** (Data Transfer Objects) to the API layer. It is also responsible for enforcing application-wide integrity rules, such as preventing users from exceeding budget limits, ensuring transaction categories align with user-defined constraints, or verifying goal milestones before progress is updated.

By cleanly separating business logic from both the user-facing routes and the database logic, the Service Layer promotes modularity, testability, and long-term maintainability. Developers can modify workflows, enforce new rules, or replace underlying technologies (like switching the database provider) without affecting the core behavior of the application. This abstraction also simplifies unit testing, as services can be tested in isolation using mocked dependencies.



- **Infrastructure Layer**

The **Infrastructure Layer** in the SavvyAPI backend architecture is responsible for **bridging the gap between the application's internal logic and the external world**. It sits at the lowest level of the Clean Architecture stack and **implements the technical details required to fulfill business operations**, as defined by the upper layers (especially the Service and Domain Layers).

In the SavvyAPI **project**, the Infrastructure Layer handles all the **concrete implementations of abstract contracts (interfaces)** that define how data is persisted, retrieved, or transmitted externally. It interacts directly with services like **Supabase**, **UUID libraries**, **datetime utilities**, or any third-party system the application may depend on.

This layer **does not contain any business logic or decision-making**; instead, it provides the raw capabilities, like inserting a record into the database, retrieving a user's transactions, or communicating with external storage APIs, **in a generic and reusable way**.

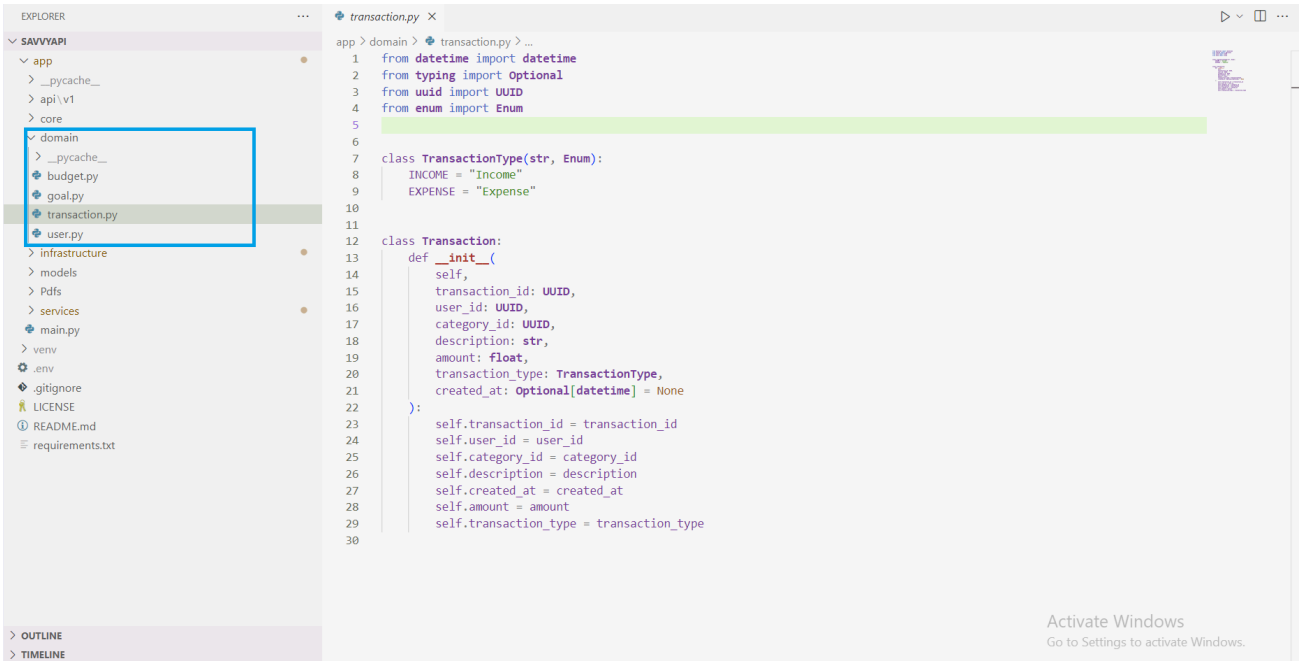
The **Infrastructure Layer** is a foundational element in the SavvyAPI architecture. It **translates domain requests into external actions**, such as storing goals in Supabase or retrieving user transactions. By separating this logic from business rules, the system becomes **modular, testable, and easier to evolve over time**.

- **Domain Layer**

The **Domain Layer** plays a **central and foundational role** in any system built using Clean Architecture. It represents the heart of business logic, where the system's core rules, concepts, and operations are defined in their purest form. This layer is completely independent of any external technology; it does not rely on databases, frameworks, user interfaces, or APIs. Instead, it focuses solely on modeling the real-world problems and behaviors your software is trying to solve.

These entities are implemented as plain Python classes that contain the essential data structures and business rules that govern them. They express what the system is and does, not how it's used or stored.

By isolating this logic in the Domain Layer, we ensure that our core functionality remains stable, reusable, and easy to test, regardless of changes in external systems like web frameworks or databases. This approach allows the system to grow and adapt while keeping its core intact and maintainable



5.2.5 Design Pattern

SavvyAPI depends on **Design Patterns** are proven, reusable solutions to commonly recurring problems in software design. They serve as templates or best practices that can be adopted to solve specific design issues in a consistent and scalable way. Rather than reinventing the wheel, developers use patterns to improve code maintainability, readability, testability, and flexibility.[44][45]

SavvyAPI adopts **Clean Architecture**, underpinned by key design patterns to ensure that the system is modular, decoupled, and testable. One of the central patterns implemented is the **Repository Design Pattern**, which abstracts and encapsulates data access logic..[46]

- **Repository Design Pattern**

The **Repository Design Pattern** is a structural pattern that provides a consistent API for data access, regardless of the underlying data source (e.g., a relational database, NoSQL store, or in-memory structure). It separates the business logic from the logic used to access the data source, promoting a clean separation of concerns.[47]

At its core, the repository pattern acts as an **in-memory collection interface for domain objects**, shielding the rest of the application from knowing how data is persisted or retrieved. It centralizes data logic, making it reusable and testable.

In the SavvyAPI project, the repository pattern is realized through two key concepts: **Interfaces** and **Implementations**, aligned with the Clean Architecture principles.

- Interfaces: app/infrastructure/interfaces/repositories/

Each repository interface defines the contract that must be fulfilled by any data access class.This interface tells the service layer what methods it can rely on, without revealing how the operations are carried out.

```
# IUserRepository
class IUserRepository(ABC):
    @abstractmethod
    def create_user(self, user: User) -> Optional[User]:
        pass
    @abstractmethod
    def get_user_by_id(self, user_id: str) -> Optional[User]:
        pass
```

- Implementations: app/infrastructure/implementation/repositories/

These are the actual classes that implement the interfaces and connect to the Supabase backend. This implementation may use httpx, Supabase Python client, or any data access logic, but the service layer is unaware of these internal details.

```
# UserRepository implements IUserRepository
class UserRepository(IUserRepository):
    def create_user(self, user: User) -> Optional[User]:
        data = {
            "email": user.email,
            "full_name": user.full_name,  }
        response = get_supabase().table("users").insert(data).execute()
        return user if response.status_code == 201 else None
```

- **Dependency Inversion Principle (DIP)**

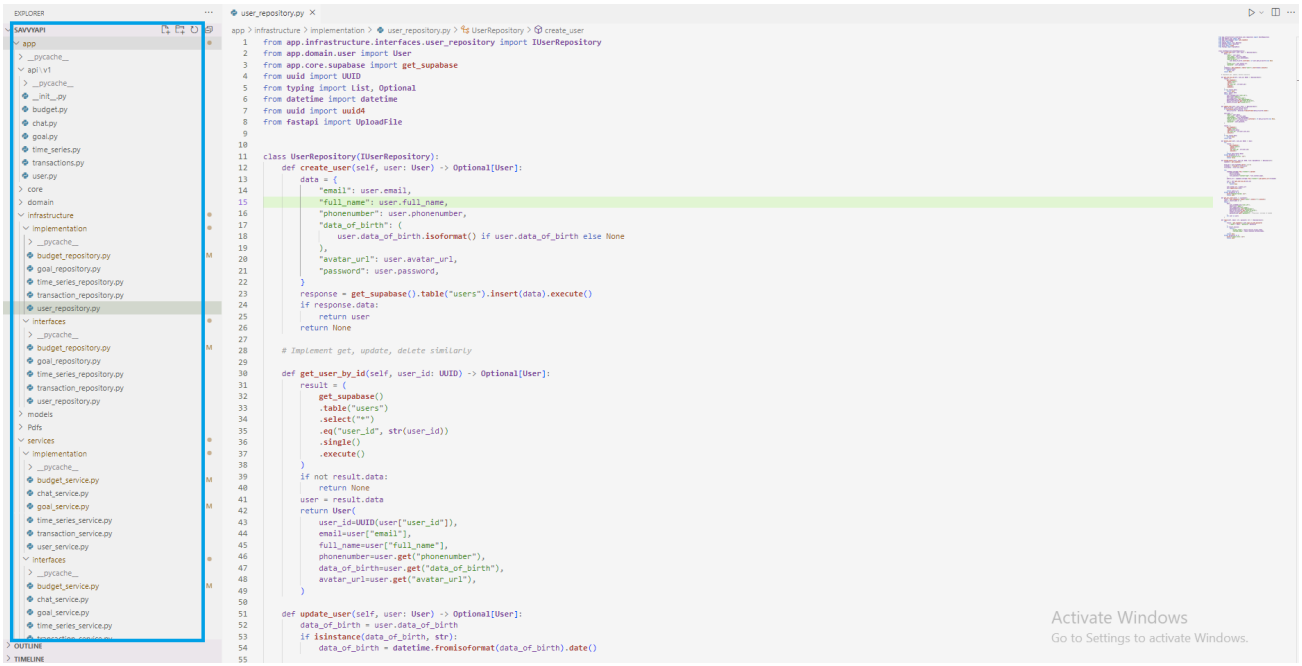
One of the foundational design principles followed in the SavvyAPI backend is the **Dependency Inversion Principle (DIP)** , the "D" in the **SOLID principles** This principle helps us maintain clean, scalable, and flexible architecture.

- Dependency Injection

We use Dependency **Injection** to invert control and provide flexibility in how components are wired together To **enforce DIP in practice** throughout the SavvyAPI project. This is the technique of **injecting dependencies into a class from the outside**, rather than hardcoding them internally.

5.2.6 Folder Structure

The SavvyAPI backend is built using **Clean Architecture** principles, which emphasize **decoupling**, **modularity**, and **maintainability**. It ensures that business rules are **isolated** from infrastructure concerns like databases, APIs, or authentication providers (in this case, Supabase). This approach makes the codebase easier to test, extend, and refactor [48]



5.2.7 API Documentation

A critical component of any modern backend system is comprehensive and easily accessible API documentation. In the **SavvyAPI** project, this need is addressed using **FastAPI's** built-in support for interactive documentation interfaces, most notably **Swagger UI** and **ReDoc**. These tools are automatically generated based on the API routes, request and response models, and type hints defined in the backend codebase, offering an up-to-date, self-explanatory overview of all available endpoints.

FastAPI leverages Python's type annotations and the Pydantic data validation system to generate precise and real-time OpenAPI-compliant documentation. This means that every time a new endpoint is added or an existing one is updated, the documentation is instantly refreshed without any additional effort. The auto-generated Swagger UI provides a user-friendly, web-based interface that lists all the API routes categorized by tags (e.g., /users, /transactions, /budgets, /goals), along

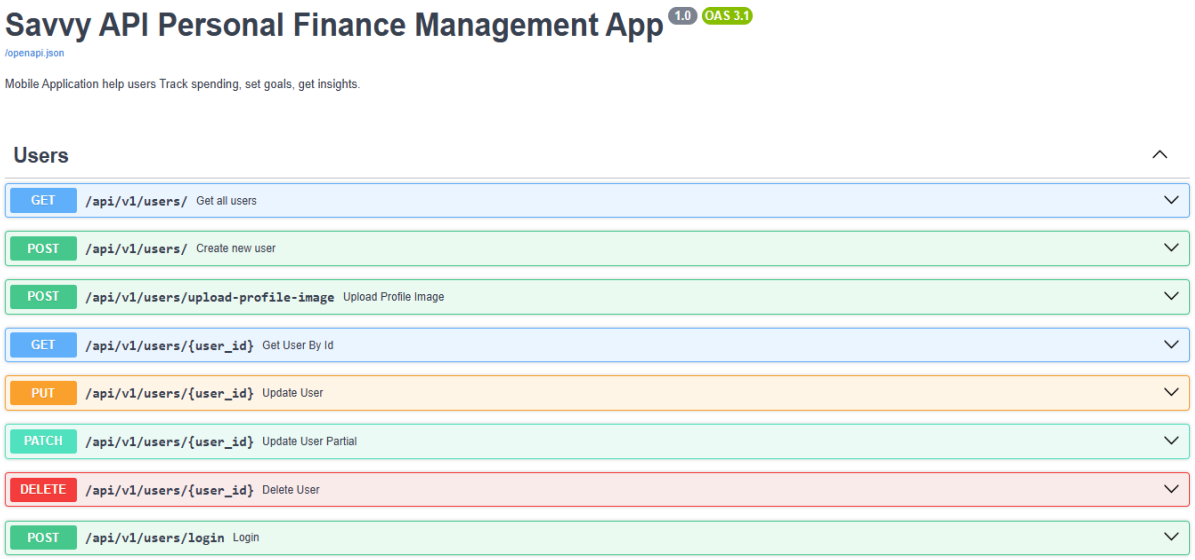
with their request parameters, expected request/response body schemas, and HTTP methods (GET, POST, PUT, DELETE, PUTCH).

Endpoint Overview

User Endpoints

The users endpoint group manages user-related operations. All routes are organized under the **"users"** tag in Swagger UI, which helps in navigating functionality clearly as example

- Get All Users
- Create New Use
- Upload Profile Image
- Get User by ID
- Update User
- Update User Partial
- Delete Use.



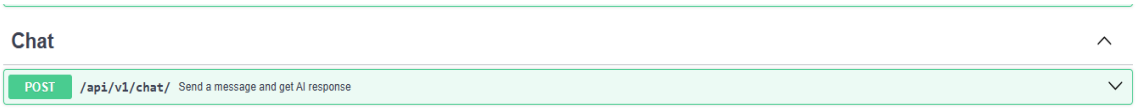
Chat Endpoints

Sends a message to the system's intelligent assistant, which uses RAG (Retrieval-Augmented Generation) to fetch personal financial data and respond meaningfully using a large language model (LLM).

- Send a Message and Get AI Response

  Request : "message": "How much did I spend on dining last month?"

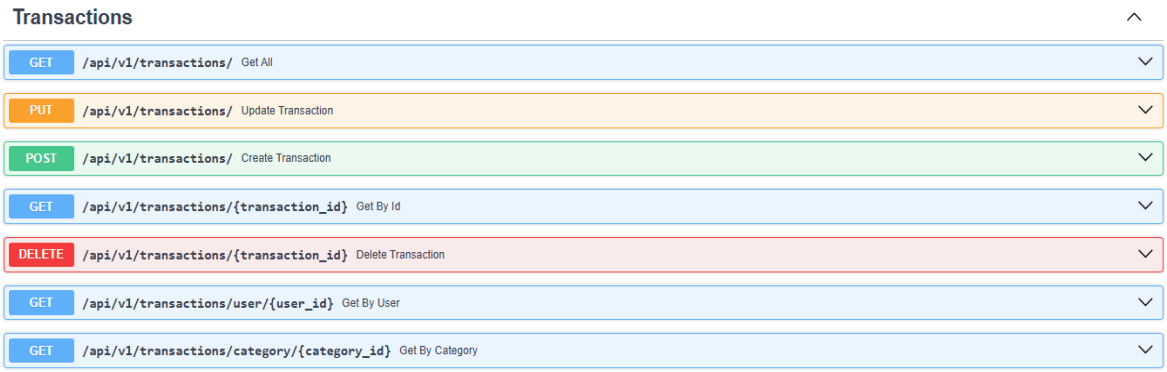
  Response: "response": "You spent approximately $220 on dining in May."



Transactions Endpoints

The transactions endpoint group handles all operations related to user transactions, such as adding, updating, deleting, or filtering transactions. These routes are essential for building financial histories, budget tracking, and analytics.

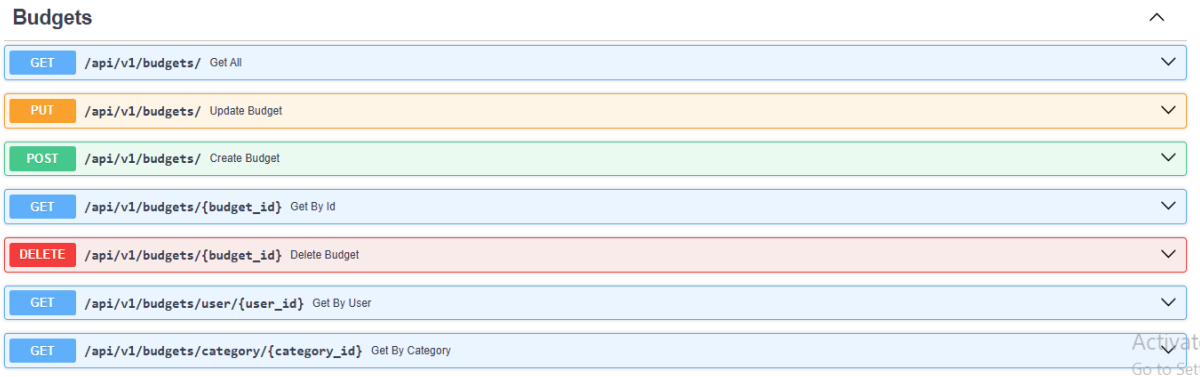
- Get All Transactions
- Create Transaction
- Update Transaction
- Get Transaction by ID
- Delete Transaction
- Get Transactions by User
- Get Transactions by Category



Budgets Endpoints

The budgets endpoint group allows users to set and manage their monthly or category-specific budgets. These endpoints are crucial for helping users stay within limits, plan expenses, and reach financial goals

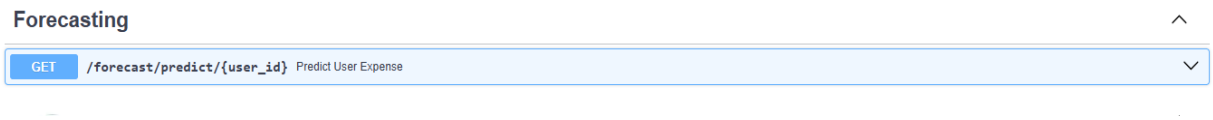
- Get All Budgets
- Create Budget
- Update Budget.
- Get Budget by ID
- Delete Budget
- Get Budgets by User
- Get Budgets by Category



Forcasting Endpoints

The forecasting endpoint enables predictive analytics using historical transaction data. It helps users plan ahead by estimating future expenses, giving them insights into potential spending patterns for the upcoming month. This feature is built using an LSTM model integrated into the backend.

- Predict user expense

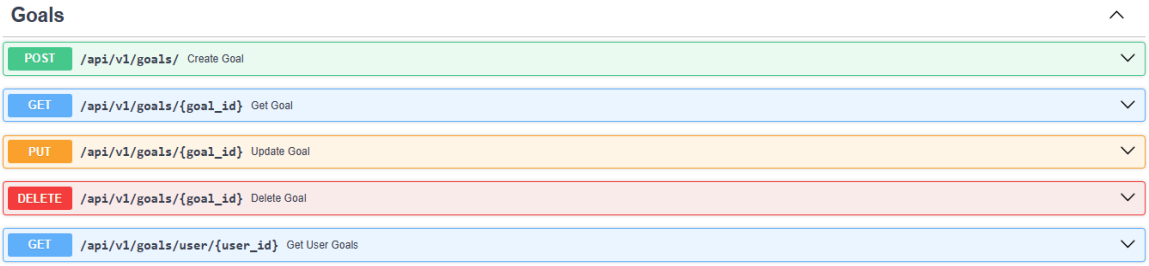


Goal Endpoints

The goal endpoint group allows users to set and track savings goals, such as "Save $1000 in 3 months" or "Save for a vacation." These endpoints provide full CRUD operations to help users define, monitor, and update their financial goals.

- Create Goal

- Update Goal.
- Get Goalby ID
- Delete Budget
- Get Budgets by User



Future work

Going forward, we plan to enhance our personal finance management app by:

1. In the future users can define their own custom categories and goals to suit their personal spending habits by setting a name and icon for the category and an optional deadline for a goal.
2. Enabling automatic tracking of income and expenses by securely connecting the app to users' credit cards, debit cards, and bank accounts, with advanced auto-categorization models for transactions to reduce manual data entry and improve accuracy.
3. Supporting multi-currency and international account management to better serve a global user base.
4. Implementing voice recognition in the chatbot to make financial management even more accessible and user-friendly.
5. Custom Date Range Filtering: currently, users can view analytics for fixed periods (Daily, Weekly, Monthly). Adding a custom date range filter would allow users to analyze financial activity over any arbitrary time span.
6. Personalized financial guardrail (Geofenced Notification), in future updates, the app will integrate geofenced location-based alerts. These notifications will be triggered when a user enters predefined high-risk spending areas (e.g., malls, fashion stores). Instead of generic alerts, the app will use humorous and motivational messages.
7. Shared Budget / Family Mode, users could optionally share budgets with family members or roommates to manage household spending together

These enhancements are designed to deliver a more personalized, and comprehensive financial management experience that adapts to the evolving needs of users.

Conclusion

In conclusion, **Savvy** represents a transformative step toward empowering individuals and organizations with smarter financial decision-making. This project brings together the power of data analytics, automation, and user-centric design to create a seamless and intelligent personal finance management system. By addressing the growing demand for accessible and insightful financial tools, **Savvy** not only simplifies budgeting, goal setting, and expense tracking, but also fosters long-term financial well-being.

Through the integration of advanced technologies and thoughtful user experience, **Savvy** positions itself as a reliable and scalable solution in the evolving landscape of financial technology. Its commitment to usability, affordability, and personalized insights sets it apart as a forward-looking platform designed to meet the needs of a diverse user base. As the culmination of rigorous planning, research, and development, **Savvy** stands ready to make a lasting impact on personal finance habits, promoting smarter savings, responsible spending, and greater financial confidence for all.

Resources

[1] PwC. 2023. *Employee financial wellness survey*. **PwC**. [accessed 2025 June 15]; https://www.pwc.com/us/en/services/consulting/business-transformation/library/employee-financial-wellness-survey.html.

[2] Leavy School of Business, Santa Clara University. 2024. *Financial modeling techniques and applications*. **Santa Clara University**. [accessed 2025 June 15]; https://onlinedegrees.scu.edu/media/blog/financial-modeling-techniques-and-applications.

[3] Stefanov, T., Stefanova, M., Varbanova, S., & Temelkov, S. 2024. *Personal Finance Management Application*. **TEM Journal, 13(3), 2066–2075**. [accessed 2025 June 15]; https://www.temjournal.com/content/133/TEMJournalAugust2024_2066_2075.pdf.

[4] Arthur William. 2024. *The impact of artificial intelligence on fintech innovation and financial inclusion: A global perspective*. **ResearchGate**. [accessed 2025 June 15]; https://www.researchgate.net/publication/385853158_The_Impact_of_Artificial_Intelligence_on_Fintech_Innovation_and_Financial_Inclusion_A_Global_Perspective.

[5] Mindful Budgets. 2024. *Mint vs YNAB: Which budgeting app is best for you?*. **Mindful Budgets**. [accessed 2025 June 15]; https://mindfulbudgets.com/mint-vs-ynab/.

[6] Ramsey Solutions. 2024. *EveryDollar: Budgeting made easy*. **Ramsey Solutions**. [accessed 2025 June 15]; https://www.ramseysolutions.com/ramseyplus/everydollar.

[7] Predictive Analytics in Financial Management. 2024. *Enhancing Decision-Making and Risk Management*. **ResearchGate**. [accessed 2025 June 15]; https://www.researchgate.net/publication/385025548_Predictive_Analytics_in_Financial_Management_Enhancing_Decision-Making_and_Risk_Management

[8] FinChat Blog. 2024. *Best AI Tools for Finance*. **FinChat Blog**. [accessed 2025 June 15]; https://finchat.io/blog/best-ai-tools-for-finance/

[9] Ethical Considerations of AI in Financial Decision. 2024. *CAI Journal*. **Academic Publishing Pte Ltd** [accessed 2025 June 15]; https://ojs.acad-pub.com/index.php/CAI/article/view/1290

[10] John Satzinger. 2015. *Systems Analysis and Design in a Changing World (7th Edition). Chapter 2: Investigating System Requirements*. **Cengage Learning** [accessed 2025 June 15]; https://books.google.com.eg/books/about/Systems_Analysis_and_Design_in_a_Changin.html?id=HW2ECwAAQBAJ&redir_esc=y

[11] Ovidijus Jurevicius. *n.d. SWOT Analysis: How to Do It*. **Strategic Management Insight** [accessed 2025 June 15]; https://strategicmanagementinsight.com/tools/swot-analysis-how-to-do-it/

[12]No author listed.2018. *Risk Assessment and Management*. **ASSP (American Society of Safety Professionals)** [accessed 2025 June 15]; https://www.assp.org/resources/risk-assessment-and-management-for-safety-professionals

[13] Yousuf M. 2023. *Incremental Process Model in Software Engineering*. **Medium** [accessed 2025 June 15]; https://medium.com/@m.yousuf1254/software-process-model-part-two-step-by-step-into-incremental-development-bdc3f6720a41

[14]Mah P.T. 2022. *Personal finance management mobile application with chatbot*. **Universiti Tunku Abdul Rahman** [Bachelor's degree final year project]. [accessed 2025 June 15]; http://eprints.utar.edu.my/4740/1/fyp_IB_2022_MPT.pdf

[15] Chase H. n.d. *LangChain Chat with Your Data*. **DeepLearning.AI** [accessed 2025 June 15]; https://learn.deeplearning.ai/courses/langchain-chat-with-your-data/lesson/snupv/introduction

[16] No author listed.n.d.*What is LangChain? Describe its purpose and benefits*. n.d. *Amazon Web Services (AWS)* [accessed 2025 June 15]; https://aws.amazon.com/what-is/langchain/

[17] Jonathan Bennion. n.d. *Developing LLM Applications with LangChain*. **DataCamp** [accessed 2025 June 15]; https://www.datacamp.com/users/sign_in?redirect=http%3A%2F%2Fapp.datacamp.com%2fLearn%2f Courses%2f Developing-llm-applications-with-long chain

[18] K C.S. 2024. *Bert vs. LLM: A business leader's understanding for competitive advantage*. **LinkedIn** [accessed 2025 June 15]; https://www.linkedin.com/pulse/bert-vs-llm-business-leaders-understanding-competitive-k-vucbc/

[19] No author listed. n.d. *Prompt Engineering Guide*. **Nextra** [accessed 2025 June 15]; https://www.promptingguide.ai/

[20] No author listed. n.d. *Prompting engineering for DeepSeek-R1*. **Together AI Docs** [accessed 2025 June 15]; https://docs.together.ai/docs/prompting-deepseek-r1

[21] Aryani A. 2023. *8 types of prompt engineering*. **Medium** [accessed 2025 June 15]; https://medium.com/@amiraryani/8-types-of-prompt-engineering-5322fff77bdf

[22] Shanthababu Pandian. 2025. *A comprehensive guide to time series analysis*. **Analytics Vidhya** [accessed 2025 June 15]; https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-to-time-series-analysis/

[23] Ismat Samadov. 2025 . *Personal Budget Transactions Dataset*. **Kaggle** [accessed 2025 June 15]; https://www.kaggle.com/datasets/ismetsemedov/personal-budget-transactions-dataset?select=budget_data.csv

[24] Bex Tuychiyev. n.d. *Anomaly Detection in Python*. **DataCamp** [accessed 2025 June 15]; https://app.datacamp.com/learn/courses/anomaly-detection-in-python

[25] No author listed. n.d. *Components of Time Series Data*. **GeeksforGeeks** [accessed 2025 June 15]; https://www.geeksforgeeks.org/data-science/components-of-time-series-data/

[26] Howell E. 2023. *How To Forecast With SARIMA*. **Towards AI** [accessed 2025 June 15]; https://pub.towardsai.net/how-to-forecast-with-sarima-d4b4c98fca7b

[27] Iván Palomares Carrascosa. 2025. *How to Use Rolling Statistics for Time Series Analysis in Python*. **Statology** [accessed 2025 June 15]; https://www.statology.org/how-to-use-rolling-statistics-for-time-series-analysis-in-python/

[28] Monigatti L. 2y. *Stationarity in Time Series — A Comprehensive Guide*. **Towards Data Science** [accessed 2025 June 15]; https://towardsdatascience.com/stationarity-in-time-series-a-comprehensive-guide-8beabe20d68/

[29] Monigatti L. 2022. *Interpreting ACF and PACF Plots for Time Series Forecasting*. **Towards Data Science** [accessed 2025 June 15]; https://towardsdatascience.com/interpreting-acf-and-pacf-plots-for-time-series-forecasting-af0d6db4061c/

[30] No author listed. n.d. *Long Short-Term Memory (LSTM)*. **NVIDIA Developer** [accessed 2025 June 15]; https://developer.nvidia.com/discover/lstm

[31] Brownlee J. 2017. *Multivariate Time Series Forecasting with LSTMs in Keras*. **MachineLearningMastery** [accessed 2025 June 15]; https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

[32] 2025. *Flutter – Build apps for any screen*. **Flutter** (by Google) [accessed 2025 June 15]; https://flutter.dev/

[33] 2025. *Flutter UI Widgets* — widget catalog and documentation. **Flutter Documentation** [accessed 2025 June 15]; https://docs.flutter.dev/ui/widgets

[34] 2025. *smooth_page_indicator* — Flutter package for customizable page indicators. **pub.dev** [accessed 2025 June 15]; https://pub.dev/packages/smooth_page_indicator

[35] 2024. *image_picker* — Flutter package for picking images from the gallery or camera. **pub.dev** [accessed 2025 June 15]; https://pub.dev/packages/image_picker

[36].2025. *mailer* — easy-to-use Dart/Flutter library for composing and sending emails with attachments and HTML support. **pub.dev** [accessed 2025 June 15]; https://pub.dev/packages/mailer

[37] 2025. *shared_preferences* — Flutter plugin for reading and writing simple key-value pairs (wraps NSUserDefaults on iOS and SharedPreferences on Android). **pub.dev** [accessed 2025 June 15]; https://pub.dev/packages/shared_preferences

[38] 2025. *flutter_local_notifications* — cross-platform Flutter plugin for displaying and scheduling local notifications. **pub.dev** [accessed 2025 June 15]; https://pub.dev/packages/flutter_local_notifications

[39] 2025. *timezone* — IANA time zone database and time zone–aware DateTime for Dart/Flutter. **pub.dev** [accessed 2025 June 15]; https://pub.dev/packages/timezone

[40] 2025. *permission_handler* — cross-platform Flutter plugin for requesting and checking runtime permissions (Android, iOS, web, Windows). **pub.dev** [accessed 2025 June 15]; https://pub.dev/packages/permission_handler

[41] 2025. *fl_chart* — highly customizable Flutter chart library (supports Line, Bar, Pie, Scatter & Radar Charts). **pub.dev** [accessed 2025 June 15]; https://pub.dev/packages/fl_chart

[41] 2025. *flutter_riverpod* — a reactive caching and data-binding framework for Flutter, simplifying asynchronous handling. **pub.dev** [accessed 2025 June 15]; https://pub.dev/packages/flutter_riverpod

[42] 2025.*FastAPI – build APIs with modern Python type hints* — high-performance web framework with automatic OpenAPI docs. **FastAPI (by Sebastián Ramírez)** [accessed 2025 June 15]; https://fastapi.tiangolo.com/

[43] 2025. *Supabase Documentation* — detailed guides and API references covering Supabase services. **Supabase Docs (by Supabase Inc.)** [accessed 2025 June 15]; https://supabase.com/docs

[44] No author listed. 2024 . *Repository Design Pattern*. **GeeksforGeeks** [accessed 2025 June 15]; https://www.geeksforgeeks.org/repository-design-pattern/

[45] Bergman P.E. 2017. *Repository Design Pattern*. **Medium** [accessed 2025 June 15]; https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30

[46] Nanavaty R. n.d. *Clean Architecture*. **Medium** [accessed 2025 June 15]; https://medium.com/@rudrakshnanavaty/clean-architecture-7c1b3b4cb181

[47] No author listed. n.d. *Clean Architecture with FastAPI*. **Fueled** [accessed 2025 June 15]; https://fueled.com/blog/clean-architecture-with-fastapi/

[48] No author listed. n.d. *Clean Architecture*. **Bitloops** [accessed 2025 June 15]; https://bitloops.com/docs/bitloops-language/learning/software-architecture/clean-architecture