Savetify

Software Design Specification

12.05.2024

Mustafa Emir Uyar - 150120007

Muhammed Hayta - 150121068

Efe Özgen - 150121077

Necati Koçak - 150120053

Prepared for

CSE3044 Software Engineering Term Project

Table of Contents

1. Introduction

1.1. Purpose

Savetify aims to empower individuals to take charge of their finances by providing a comprehensive platform to track income, expenses, investments, and more. Our goal is to promote financial awareness and informed decision-making.

1.2. Statement of scope

1.2.1 Project Description

Savetify is a cross-platform mobile and web application designed to simplify personal finance management. It provides a secure environment for users to centralize and track their financial information, receive insights into their spending habits, and set achievable financial goals.

1.2.2 Major Inputs

1.2.2.1 User-Entered Data: Expenses (amount, category, date), Income (amount, source, date), Investments (type, purchase details, current value, date), Financial Goals.
External Data Sources (Optional): Real-time financial data from third-party APIs for stocks, cryptocurrency, currency exchange rates.

1.2.3 Major Outputs

1.2.3.1 Financial Summaries: Dashboard with key financial metrics (net worth, income/expense trends, portfolio overview).

1.2.3.2 Detailed Reports: Breakdowns of spending by category, income by source, investment performance.

1.2.3.3 Spending Insights: Personalized recommendations and observations based on user data.

1.2.3.4 Budget Alerts: Notifications when spending approaches or exceeds limits.

1.2.4 Essential

1.2.4.1 Expense Tracking: Record, categorize, and store expense transactions.

1.2.4.2 Income Tracking: Record and categorize income transactions.

1.2.4.3 Basic Investment Tracking: Manually enter investment details (type, quantity, purchase price). Calculate basic returns (ROI).

1.2.4.4 Account Management: Secure user registration and login (through Firebase).

1.2.4.5 Data Visualization: Generate charts/graphs of expenses, income, and investment performance over time.

1.2.5 Desirable

1.2.5.1 **Insight Generation:** Analyze spending patterns and provide basic recommendations to the user.

1.2.5.2 **Budgeting Tools:** Allow users to set budget limits and receive alerts.

1.2.5.3 **Personalized Spending Profiles:** Categorize users into spending profiles and offer tailored insights (requires further research into suitable algorithms).

1.2.6 Future Requirements

1.2.6.1 **Advanced Investment Features:** Portfolio analysis, risk assessment, rebalancing suggestions.

1.2.6.2 **Gamification Elements:** Progress tracking, rewards systems for building positive financial habits.

1.2.6.3 **Automated Data Retrieval:** Explore secure connections with financial institutions (open banking integration).

1.2.6.4 **Integration with Financial APIs:** Retrieve live stock/cryptocurrency prices (if reliable sources are found).

1.3.Software context

Savetify aims to be an easy-to-use personal finance app, competing with similar tools already out there. It aims to stand out by being very user-friendly, tracking not just spending but also income and investments, and working on multiple devices. Success depends on making the experience smooth and later exploring ways to make money, like offering extra features for a fee, using collected data responsibly, or connecting users with financial services. Savetify hopes to attract people who want a simpler way to manage their money without feeling overwhelmed.

1.4.Major constraints

1.4.1 Resource Constraints: The project has a limited time frame and only 4 team members. Also, we don't have any budget for paid assets and resources.

1.4.2 Technical Constraints: Flutter framework has some limitations about its cross platform compatibility. Not every plugin is compatible with all platforms.

1.4.3 Offline Functionality: Implementing offline capabilities with data synchronization introduces technical complexity and could increase development time.

1.4.4 Scope-Related Constraints: The app must clearly state that it provides tracking tools, not financial advice. This limits the types of recommendations it can offer users.

1.5. Definitions

1.5.1. **Asset:** Any item of economic value owned by the user (e.g., cash, stocks, bonds, real estate, precious metals, cryptocurrency).

1.5.2. **Investment:** Allocating money with the expectation of generating income or profit.

1.5.3. **Portfolio:** A collection of assets held by the user.

1.5.4. **Diversification:** The strategy of spreading investments across different asset classes to reduce risk.

1.5.5. **Financial Goal:** A specific target the user aims to achieve financially (e.g., saving for a down payment, building an emergency fund, retiring early).

1.5.6. **Insight:** A data-driven observation about the user's financial behavior that provides helpful information (e.g., identifying a recurring expense that could be reduced).

1.5.7. **ROI (Return on Investment):** A percentage measuring the profitability of an investment over time, taking into account gains and losses.

1.5.8. **Cloud:** The way to store your data (like files, photos, and software) and access powerful computing resources over the internet.

1.5.9. **Transaction:** An instance of buying or selling something; a business deal.

1.5.10. **Category:** A group of items sharing common characteristics, used for organizing transactions.

1.5.11. **Balance:** The remaining amount of money in an account after all credits and debits have been accounted for.

1.5.12. **Dividend:** The portion of a company's profits that is distributed to its shareholders.

1.5.13. **Stock Exchange:** A marketplace where shares of stock and other assets are bought and sold.

1.6. *Acronyms and Abbreviations*

1.6.1. **ROI:** Return on Investment

1.6.2. **UI:** User Interface

1.6.3. **UX:** User Experience

1.6.4. **API:** Application Programming Interface

1.7. *References*

1.7.1. **Flutter Development Guidelines:** https://docs.flutter.dev

1.7.2. **Firebase Best Practices:** https://firebase.google.com/docs

2. *Design Consideration*

2.1. Design Assumptions and Dependencies

2.1.1 Related Software or Hardware:

2.1.1.1. It is assumed that users have internet access on their devices for Savetify to function properly.

2.1.1.2. Stable operation of backend services such as Firebase is required.

2.1.1.3. Dependency on cross-platform development tools is necessary to ensure users can use the application on various devices and operating systems (Web, Android, iOS, Windows).

2.1.2 Operating Systems:

2.1.2.1. Supported operating systems for Savetify: Windows 10 and above, Android 6.0 and above, iOS 12 and above, Latest versions of Chrome web browser.

2.1.3 End-user Characteristics:

2.1.3.1. Basic internet access is assumed for users.

2.1.3.2. Users are generally expected to have a basic understanding of financial management.

2.1.3.3. Considering users may come from various age groups and professions, the user interface design and user experience should accommodate this diversity.

2.1.4 Possible and/or Probable Changes in Functionality:

2.1.4.1. Local Operation and Data Retrieval: Adding the ability for Savetify to operate locally and retrieve some data from the local device can allow users to use the application without internet access. This could be particularly useful in scenarios where users have limited or no internet connectivity. Mechanisms for local data storage and synchronization can serve this purpose.

2.1.4.2. Advanced Data Analytics and Recommendations: More in-depth analysis of users' financial habits and preferences can be conducted, leading to personalized recommendations.

2.2. General Constraints

2.2.1 Hardware or software environment:

2.2.1.1. Savetify should be able to operate on various mobile devices (Android and iOS), desktop computers (Windows) and Chrome web browser.

2.2.1.2. Software-wise, Flutter will be utilized for development, and access to backend services like Firebase will be required.

2.2.2 End-user environment:

2.2.2.1. Users should be able to use Savetify in both online and offline environments.

2.2.3. Availability or volatility of resources:

2.2.3.1. Savetify should function stably even on devices with limited resources, ensuring availability and performance.

2.2.3.2. Resources like database access and network connectivity are crucial for availability.

2.2.4. Standards compliance:

2.2.4.1. Adherence to coding standards and guidelines, such as those provided by Flutter and Firebase, should be maintained throughout the development process.

2.2.4.2. Compliance with usability and accessibility standards is essential.

2.2.5. Interoperability requirements:

2.2.5.1. Savetify should be able to interact with financial services (e.g., payment gateways, financial APIs).

2.2.5.2. Integration with external data sources (e.g., stock prices, exchange rates) should be provided.

2.2.6. Interface/protocol requirements:

2.2.6.1. The user interface should be user-friendly and intuitive.

2.2.6.2. Suitable APIs should be used for backend and database access.

2.2.7. Data repository and distribution requirements:

2.2.7.1. User data should be securely stored and distributed in Firebase Firestore.

2.2.8. Security requirements (or other such regulations):

2.2.8.1. User data should be protected with strong encryption and authentication methods.

2.2.8.2. Authorization and session management should be implemented.

2.2.9. Memory and other capacity limitations:

2.2.9.1. The application may have limited memory and processing power on various devices and environments.

2.2.9.2. Efficient resource utilization and performance optimization are important.

2.2.10. Performance requirements:

2.2.10.1. The application should meet specific performance targets for fast response and smooth user experience.

2.2.10.2. Ensuring fast and seamless user interactions is crucial.

2.2.11. Network communications:

2.2.11.1. The application should be able to exchange data over reliable and fast network connections.

2.2.12. Verification and validation requirements (testing):

2.2.12.1. The application should undergo comprehensive testing, including unit tests, integration tests, and user acceptance tests.

2.2.12.2. Reliability, performance, and security of the software should be tested.

2.2.13. Other means of addressing quality goals:

2.2.13.1. User feedback and continuous improvement processes should be utilized to enhance the application's quality.

2.2.14. Other requirements described in the requirements specification:

2.2.14.1. All other requirements specified in the project requirements document should be considered and met.

2.3. System Environment

Savetify will be developed and deployed in the following system environment:

2.3.1. Hardware:

2.3.1.1. Development Machines: Standard desktop or laptop computers with sufficient processing power and memory to run development tools such as Flutter and Firebase emulators.

2.3.1.2. Target Platforms:

2.3.1.2.1. Mobile Devices: Android smartphones and tablets with various screen sizes.

2.3.1.2.2. Web Browsers: Modern web browsers with support for HTML5, CSS3, and JavaScript (e.g. Chrome).

2.3.2. Software:

2.3.2.1. Development Tools:

2.3.2.1.1. Flutter SDK: For cross-platform mobile and web app development.

2.3.2.1.2. Firebase: For backend services like user authentication, database management, and cloud functions.

2.3.2.1.3. Version control system (Git): for code management and collaboration.

2.3.2.1.4. Integrated Development Environment (IDE) or code editor: for development activities.

2.3.2.2. Target Platform Software:

2.3.2.2.1. Android: Android operating system for mobile devices.

2.3.2.2.2. Web Browsers: No additional software required beyond the user's web browser.

2.3.3. Network:

2.3.3.1 A reliable internet connection is required for initial application download, user authentication, data synchronization with the backend server, and potentially for retrieving real-time financial data through external APIs.

2.3.3.2. The application will be designed to function partially offline, allowing users to enter and store data locally with synchronization occurring when an internet connection is re-established.

2.4. Development Methods

We are planning to use the Waterfall approach in the development of this project mainly because of strict timing constraints. Because we already had a detailed requirement specification the Waterfall approach was also more suitable for our use.

For a detailed description of the waterfall approach checkout reference[1]

We are planning to use the MVVM and Repository design patterns mainly to modularize and simplify the program, thereby increasing maintainability. We chose the Repository design pattern because it enables us to easily use local and cloud storage. We chose the MVVM design pattern because it gives us the opportunity to easily modify the program for future needs.

For a detailed description of the MVVM design pattern checkout reference[2]

For a detailed description of the Repository design pattern checkout reference[3]

We are planning to use test driven development with automated unit tests to ensure code quality and reliable operation of the program.

For a detailed description of the Test Driven Development(TDD) checkout reference[4]

We considered the agile development method but mainly because of timing limitations we could not use this approach. Another reason is that we don’t have changing requirements thereby rendering the Agile process useless for our approach.

For a detailed description of the Agile approach checkout reference[5]

3. Architectural and component-level design

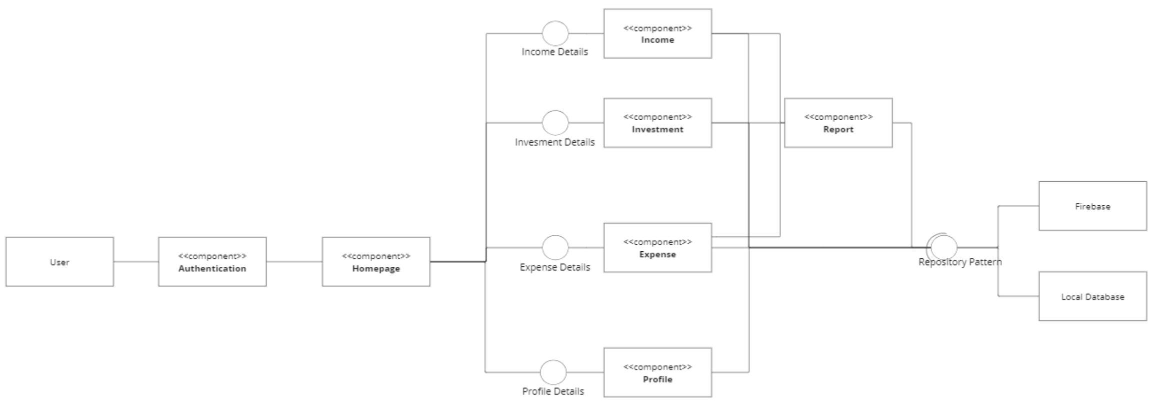
3.1. System Structure

Savetify has 7 components: Income, Expense, Report, Investment, Profile, Homepage, Authentication

Savetify's client-side, built with Flutter, has 5 core components. The Income component captures, displays, and visualizes user income data. Similarly, the Expense component handles expense tracking, categorization, and analysis. The Report component generates summaries and visualizations of financial data, while the Investment component enables users to track various types of investments and calculate returns. The Profile component keeps the users personal information and preferences of investment choices.

Other components like the homepage serve as the central hub for users, displaying key metrics and providing access to other components. Another component is User authentication and secure account management are handled through Firebase Authentication, ensuring data privacy and protection. Firebase Firestore, a NoSQL database, stores user data, enabling efficient retrieval and updates.

3.1.1. **Architecture diagram**



3.2. Description of Components

1. Income:

1.1. Processing Narrative (PSPEC):

1.1.1. Accepts user input for income amount, date, and source.

1.1.2. Validates entered data to ensure accuracy and consistency.

1.1.3. Stores income data securely within the Firebase database.

1.1.4. Provides functionalities to retrieve and display income history, categorized by source and date.

1.1.5. Calculates and presents income trends over time to aid in financial awareness.

1.2. Interface Description:

1.2.1. Inputs:

1.2.1.1. Income amount (numerical)

1.2.1.2. Income date (date/time)

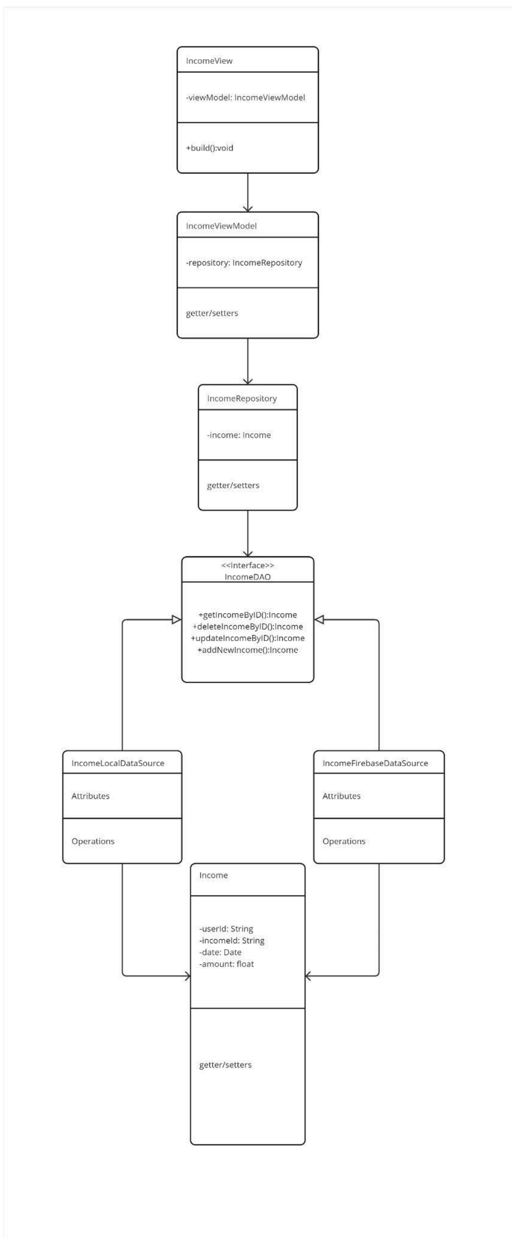
1.2.1.3. Income source (text)

1.2.2. Outputs:

1.2.2.1. User Income Data

1.3. Component Processing Detail:

1.3.1. Design Class Hierarchy:



1.4. Restrictions/limitations:

1.4.1. Income must have amount, date and source.

1.5. Performance issues:

1.5.1. Limited data should be shown in the initial upload, and the remaining income data should be uploaded if necessary.

1.6. Design constraints:

1.6.1. The interaction between the database should be secure with data encryption.

1.6.2. The income page should be responsive.

1.7. Processing detail for each operation:

1.7.1. Users select to add/edit income.

1.7.2. User selects the type and amount of income.

1.7.3. User selects to delete income.

1.7.4. All changes are stored in the local database and synced with the online database when internet connection is established.

2. Expense:

2.1. Processing Narrative (PSPEC):

2.1.1. Accepts user input for expense details (amount, date, source).

2.1.2. Stores expense data securely in the Firebase database.

2.1.3. Provides features to retrieve and display expense history, categorized by date, amount, and category.

2.2. Interface Description:

2.2.1. Inputs:

2.2.1.1. Amount

2.2.1.2. Date

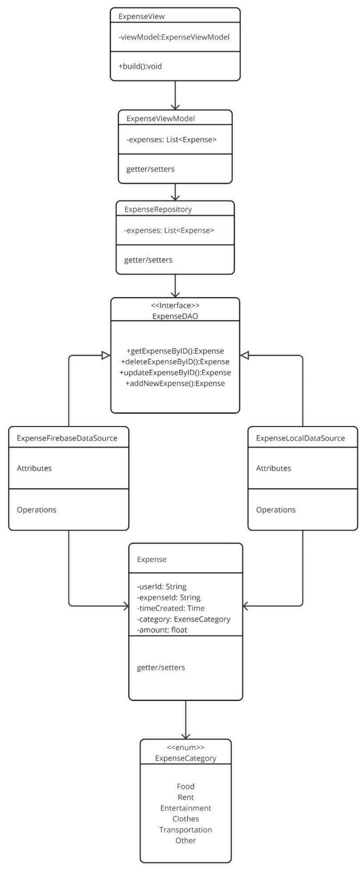
2.2.1.3. Category

2.2.2. Outputs:

2.2.2.1. User Expense Data

2.3. Component Processing Detail:

2.3.1. Design Class Hierarchy:



2.3.2. Restrictions/limitations:

2.3.2.1. Expenses must have an amount, date and source.

2.3.3. Performance issues:

2.3.3.1. Limited data should be shown in the initial upload, and the remaining expense data should be uploaded if necessary.

2.3.4. Design constraints:

2.3.4.1. The interaction between the database should be secure with data encryption.

2.3.4.2. The income page should be responsive.

2.3.5. Processing detail for each operation:

2.3.5.1. User selects to add/edit expenses.

2.3.5.2. User selects the type and amount of expense.

2.3.5.3. User selects to delete the expense.

2.3.5.4. All changes are stored in the local database and synced with the online database when internet connection is established.

3. Report:

3.1. Processing Narrative (PSPEC):

3.1.1. Retrieves income and expense data from respective components (Income and Expense).

3.1.2. Calculates financial metrics (total income, expenses, savings, investment performance).

3.1.3. Analyzes data to generate reports based on user-specified timeframes (e.g., monthly, weekly).

3.1.4. Presents reports in user-friendly formats (tables, charts, graphs) for easy understanding.

3.2. Interface Description:

3.2.1.1. Inputs:

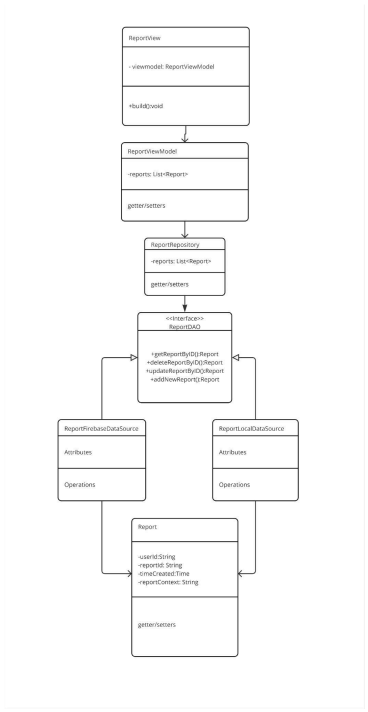
3.2.1.1.1. None

3.2.2. Outputs:

3.2.2.1. Required Report

3.3. Component Processing Detail:

3.3.1. Design Class Hierarchy:



3.3.2. Restrictions/limitations:

3.3.2.1. Report must be adjective according to screen size.

3.3.3. Performance issues:

3.3.3.1. Execution time of report generation cannot exceed 2 minutes.

3.3.4. Design constraints:

3.3.4.1. Users must specify a time period.

3.3.4.2. Time period of the report cannot exceed 1 year.

3.3.5. Processing detail for each operation:

3.3.5.1. User selects the type of report.

3.3.5.2. User selects the time period of the report.

3.3.5.3. All generated reports are stored in the local database and synced with the online database when internet connection is established.

4. Investment:

4.1. Processing Narrative (PSPEC):

4.1.1. Accepts user input for investment details (amount, date, source).

4.1.2. Stores investment data securely in the Firebase database.

4.1.3. Provides functionalities to retrieve and display investment history, categorized by type and performance.

4.1.4. Integrates with external APIs (if applicable) for real-time data or calculating returns.

4.2. Interface Description:

4.2.1. Inputs:

4.2.1.1. Type

4.2.1.2. Purchase details

4.2.1.3. Current value

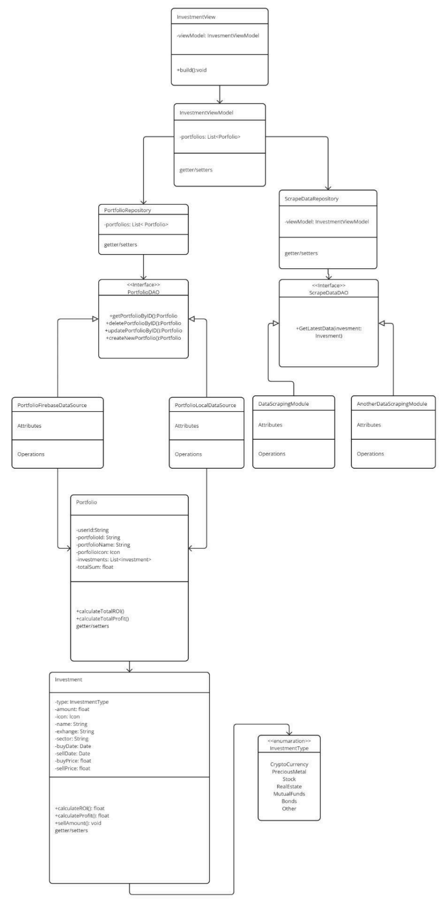
4.2.1.4. Date

4.2.2. Outputs:

4.2.2.1. User Investment Data

4.3. Component Processing Detail:

4.3.1. Design Class Hierarchy:



4.3.2. Restrictions/limitations:

4.3.2.1. Investment must have amount, date and source.

4.3.3. Performance issues:

4.3.3.1. Limited data should be shown in the first upload, and the remaining investment data should be uploaded if necessary.

4.3.4. Design constraints:

4.3.4.1. The interaction between the database should be secure with data encryption.

4.3.4.2. The income page should be responsive.

4.3.5. Processing detail for each operation:

4.3.5.1. User selects to add/edit investment.

4.3.5.2. User selects the type and amount of investment.

4.3.5.3. User selects to sell/delete investment.

4.3.5.4. All changes are stored in the local database and synced with the online database when internet connection is established.

5. Profile:

5.1. Processing Narrative (PSPEC):

5.1.1. This component is responsible for managing the information of the user. It records the personal information of the user and gives the option to edit their preferences.

5.2. Interface Description:

5.2.1. Inputs:

5.2.1.1. Name

5.2.1.2. Surname

5.2.1.3. Preferences

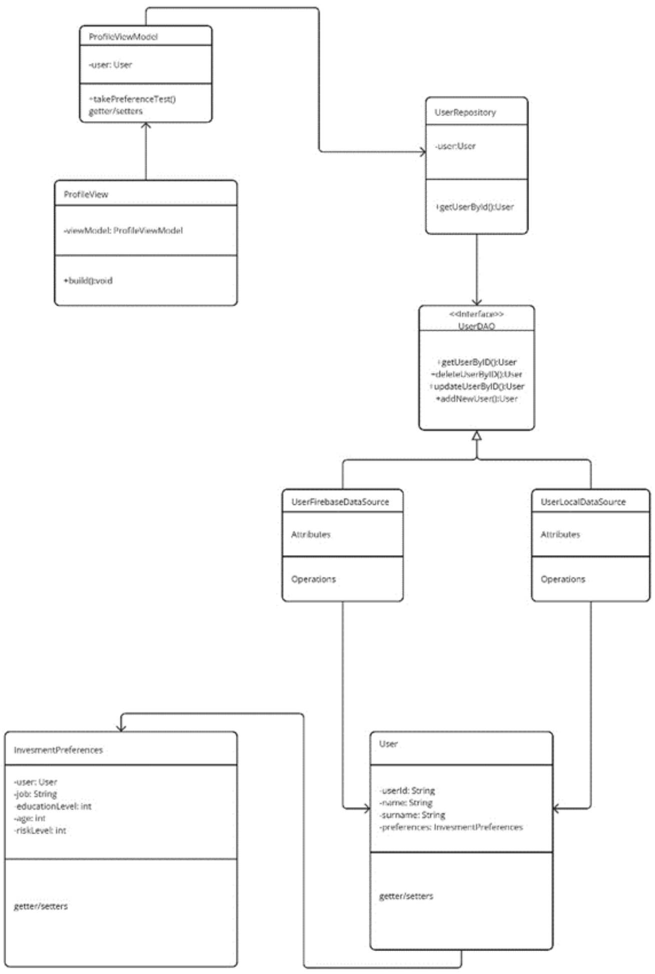
5.2.2. Outputs:

5.2.2.1. Show profile

5.2.2.2. Show preferences

5.3. Component Processing Detail:

5.3.1. Design Class Hierarchy:



5.3.2. Restrictions/limitations:

5.3.2.1. Users must have a name, surname and e-mail.

5.3.3. Performance issues:

5.3.3.1. We do not have any performance issues at this stage.

5.3.4. Design constraints:

5.3.4.1. The interaction between the databases should be secure with encrypted data.

5.3.4.2. The profile component should be responsive.

5.3.5. Processing detail for each operation:

5.3.5.1. User selects to edit his profile

5.3.5.2. Data is sent to the Local source

5.3.5.3. If internet connection is established data is synced with database

6. Homepage:

6.1. Processing Narrative (PSPEC):

6.1.1. This component will display elements of the income, expense, report and investment components and will give options to navigate to these components.

6.2. Interface Description:

6.2.1. Inputs:

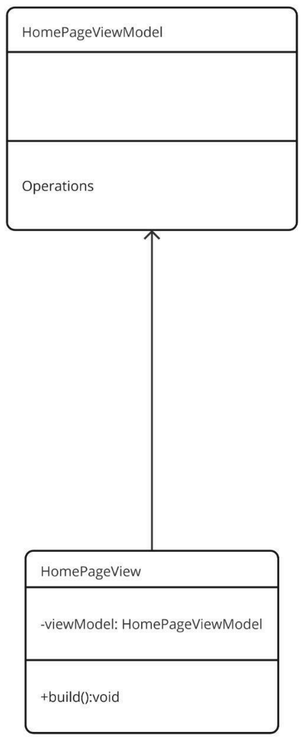
6.2.1.1. None

6.2.2. Outputs:

6.2.2.1. Show elements of investment, income and expenses.

6.3. Component Processing Detail:

6.3.1. Design Class Hierarchy:



6.3.2. Restrictions/limitations:

6.3.2.1. The homepage should respond adaptively to different platform and screen sizes.

6.3.3. Performance issues:

6.3.3.1. We do not have any performance issues at this stage.

6.3.4. Design constraints:

6.3.4.1. The homepage should have an intuitive design

6.3.4.2. The homepage should be responsive.

6.3.5. Processing detail for each operation:

6.3.5.1. The homepage screen appears after login

6.3.5.2. The homepage displays investment, income and expense components

6.3.5.3. The homepage gives an option to view different components.

7. Authentication:

7.1. Processing Narrative (PSPEC):

7.1.1. This component will provide the services to sign-in and log-in.

7.1.2. The user will use one of the log-in options, the data will be compared to the data in the database, if they match the user will be logged in and will be directed to the homepage.

7.1.3. The user will use one of the sign-in options to create an account. This data will be stored on the database.

7.2. Interface Description:

7.2.1. Inputs:

7.2.1.1. E-mail

7.2.1.2. Password

7.2.1.3. Name

7.2.1.4. Surname

7.2.1.5. Preferences

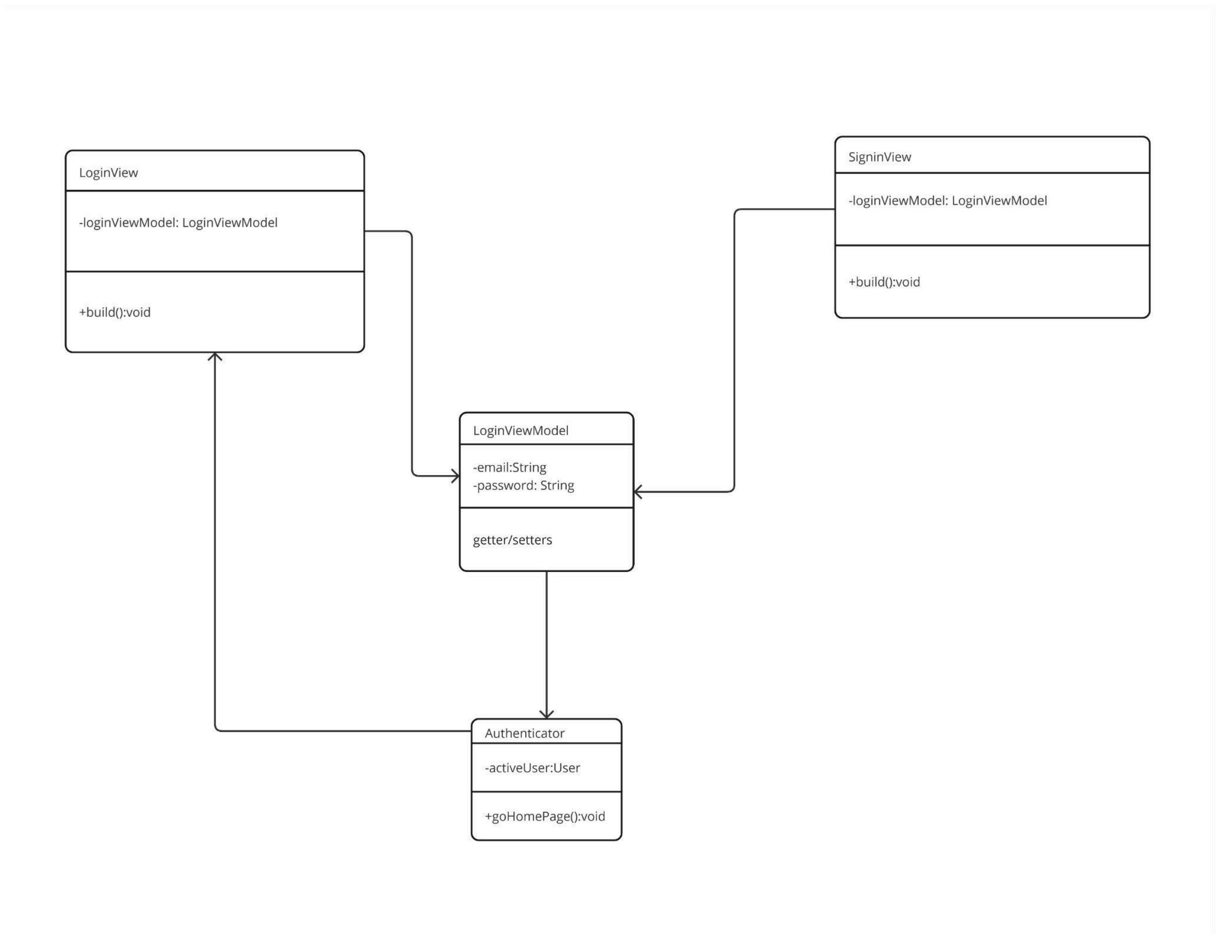
7.2.2. Outputs:

7.2.2.1. Show log-in

7.2.2.2. Sign-in screen

7.3. Component Processing Detail:

7.3.1. Design Class Hierarchy:



7.3.2. Restrictions/limitations:

7.3.2.1. User must enter correct information to log-in

7.3.2.2. User must enter email, password, name, surname and preferences while sign-in

7.3.3. Performance issues:

7.3.3.1. We do not have any performance issues at this stage.

7.3.4. Design constraints:

7.3.4.1. The password should be one-way encrypted with ssh256.

7.3.4.2. Other packages between the database and the component should encrypted

7.3.5. Processing detail for each operation:

7.3.5.1. User selects one of the available log-in methods to log-in

7.3.5.2. Data is sent to the database, if correct user will be logged in, if not an error message will be shown

7.3.5.3. User selects one of the available sign-in methods to create an account.

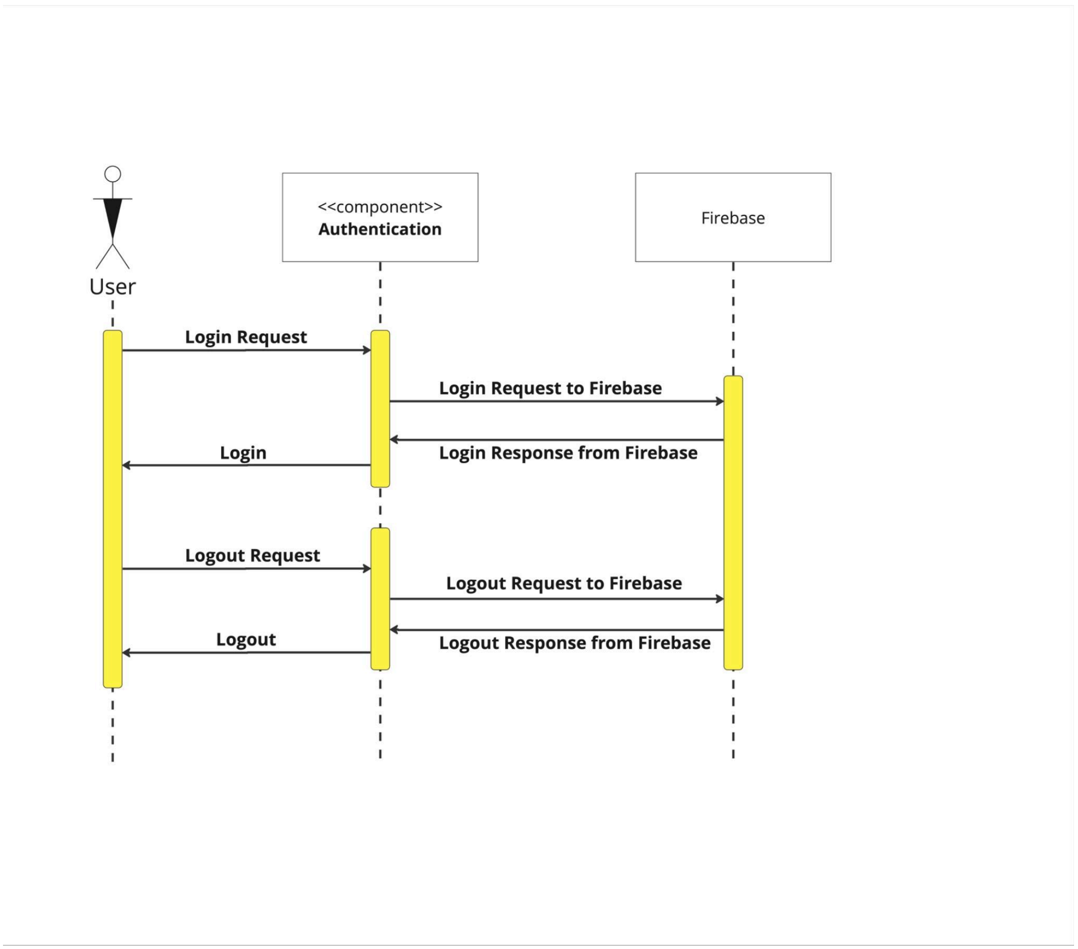
7.3.5.4. Account is created and data is saved in the database.

3.3. Dynamic Behavior for Component n

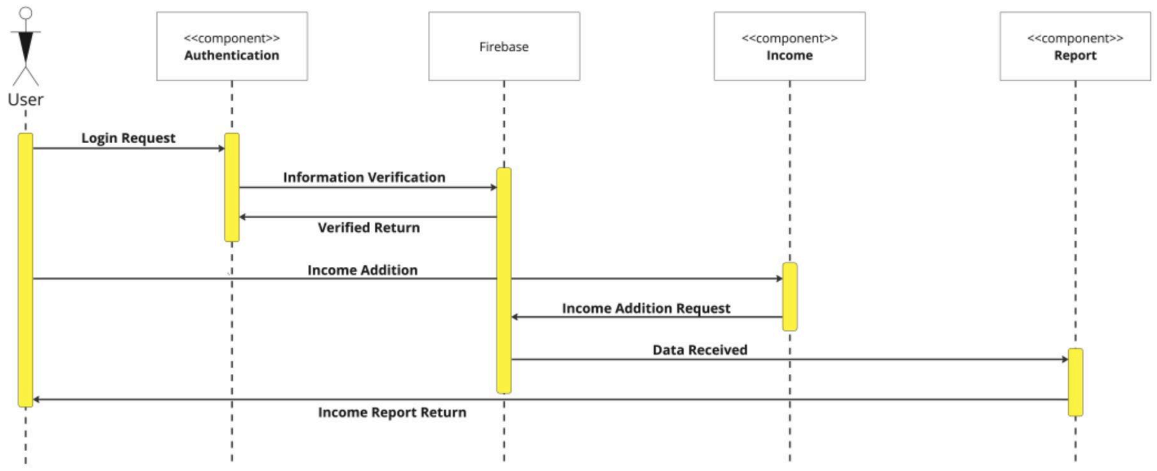
In our project, there are interactions between various classes that constitute the core functionality of the Savetify application. Firstly, the 'User' class represents user accounts and provides functions such as user login, managing personal data, and setting financial goals. The 'Expense' class enables users to record, categorize, and analyze their expenses. This class interacts with the 'User' class to retrieve user expense data and provide expense reports to the user. The 'Investment' class facilitates the management of user investments and tracking their performance. This class collaborates with the 'User' class to obtain the user's investment portfolio and present information about investment performance. All these classes come together to enhance the user experience and streamline the financial management process, forming the fundamental functionality of the Savetify application.

3.3.1. Interaction Diagrams

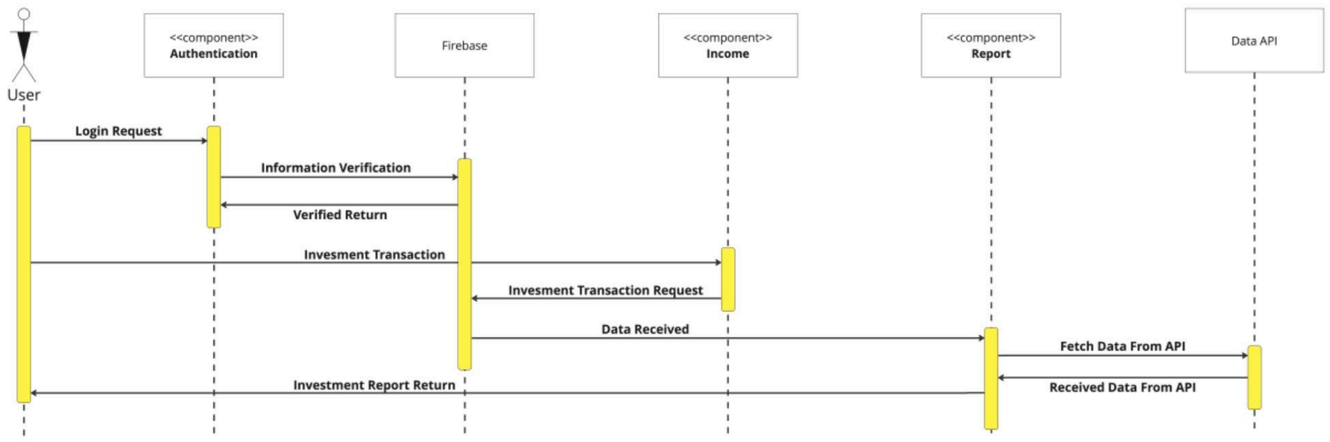
3.3.1.1. Authentication Component Sequence Diagram


User
<<component>>
Authentication
Firebase
Login Request
Login Request to Firebase
Login Response from Firebase
Login
Logout Request
Logout Request to Firebase
Logout Response from Firebase
Logout

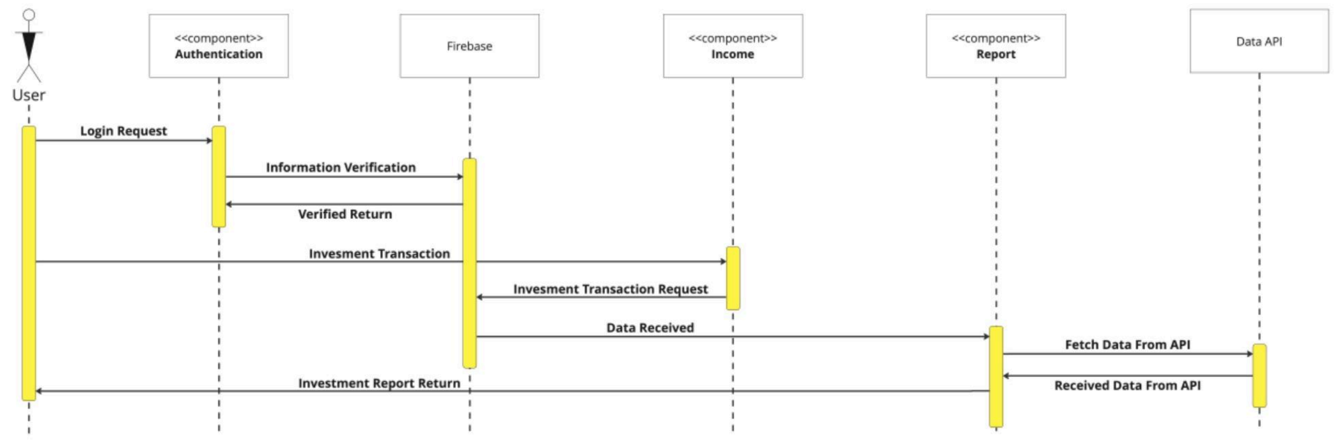
3.3.1.2. Income Component Sequence Diagram



3.3.1.3. Investment Component Sequence Diagram



3.3.1.4. Expense Component Sequence Diagram



4. Restrictions, limitations, and constraints

4.1. Restrictions

4.1.1. No Automated Financial Connections: Due to security and complexity, Savetify won't directly connect to users' bank accounts or investment platforms. Data input will be primarily manual.

4.1.2. No Binding Financial Advice: The app must make it clear that it is a tracking and analysis tool, not a source of financial advice with legal or tax implications.

4.2. Limitations

4.2.1. API Reliance: Using third-party APIs like Firebase introduces potential vulnerabilities tied to their reliability and pricing models. Changes in their availability or cost structure could impact Savetify's functionality.

4.2.2. Cross-Platform Compatibility: Ensuring a consistent, polished experience across web, Android, iOS, and Windows requires careful consideration of platform-specific differences.

4.2.3. Scope for Financial Analysis: Targeting a broad user base necessitates simplifying complex financial concepts to keep the app accessible.

4.3. Constraints

4.3.1. Development Time: The semester time frame limits the features that can be realistically implemented.

4.3.2. Budget: The project has no budget for any paid API's or assets.

4.3.3. Team Expertise: The project team has little experience with Flutter.

5. Conclusion

In conclusion, the Savetify software design specification provides a detailed blueprint for the development of a personal finance management application. The document covers the system's purpose, scope, design considerations, architectural components, and constraints. By sticking to this specification, Savetify will be a user-friendly and functional application that empowers users to effectively manage their finances. While the current design focuses on essential features like expense and income tracking, future enhancements may include advanced analytics and integration with financial APIs. This specification serves as a valuable guide for ensuring the successful development of Savetify as an advanced personal finance tool.

6. References

[1] https://en.wikipedia.org/wiki/Waterfall_model

[2] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

[3] https://www.geeksforgeeks.org/repository-design-pattern/

[4] https://www.geeksforgeeks.org/test-driven-development-tdd/

[5] https://en.wikipedia.org/wiki/Agile_software_development