Personal Finance Tracker
Design Manual

South East Technological University Carlow
Software Development

Name: Matthew Ufumeli
Student no: C00273575
Supervisor: Chris Staff
Date: 06/12/2024

Table of Contents

Personal Finance Tracker

1. Application Model

The Personal Finance Tracking App is a web-app based tool designed to help users manage and analyze their financial data by uploading CSV files or manually entering transaction details. The app's architecture includes modules for data processing, categorization, and visualization, ensuring a seamless user experience. It operates within a secure and user-friendly environment to provide insightful reports and effective budgeting tools.

---

2. How the Application Will Be Used

Navigation Sequence:

- **User Registration/Login:**
  New users register by creating an account or using social logins (Google, Facebook). Returning users log in using their credentials.
- **Dashboard Overview:**
  After logging in, users land on a Dashboard that summarizes their financial status, showing total income, expenses, and savings.
  Key navigation options:
    - Dashboard
    - Upload CSV
    - Budgets
    - Reports
    - Settings

- **CSV Upload:**
  Users navigate to the Upload CSV section to upload their financial transaction data in CSV format. The app parses the transactions and categorizes them automatically or allows users to adjust the categories manually.

- **Bank Connection:**
  Users navigate to the Settings section and select the Connect Bank Account option. The app guides the user through a secure authentication process using Open Banking APIs. Users are redirected to their bank's login portal to grant permissions. Upon successful connection, the app syncs transactions automatically and processes them similarly to uploaded CSV files. Users can manage permissions or disconnect their bank account at any time through the Settings page.

- **Budgets:**
  Users can set up and monitor budgets for various categories (e.g., groceries, rent, entertainment). The Budget page displays progress through visual charts and sends notifications or alerts as thresholds are approached.

- **Reports**:
  Users generate detailed reports in the Reports section. Reports can be customized by time range, categories, and spending trends and are downloadable in CSV or PDF format.

Navigation Flow Diagram:

- Login/Sign Up Page → Dashboard →
    - Upload CSV → CSV Parsing/Transaction Categorization
    - Settings → Connect Bank → Authenticate
    - Budgets → Budget Setup/Monitoring → Budget vs. Actual Comparison
    - Reports → Generate/Download Reports

---

3. User Interface Design (UI)

Budgets screen

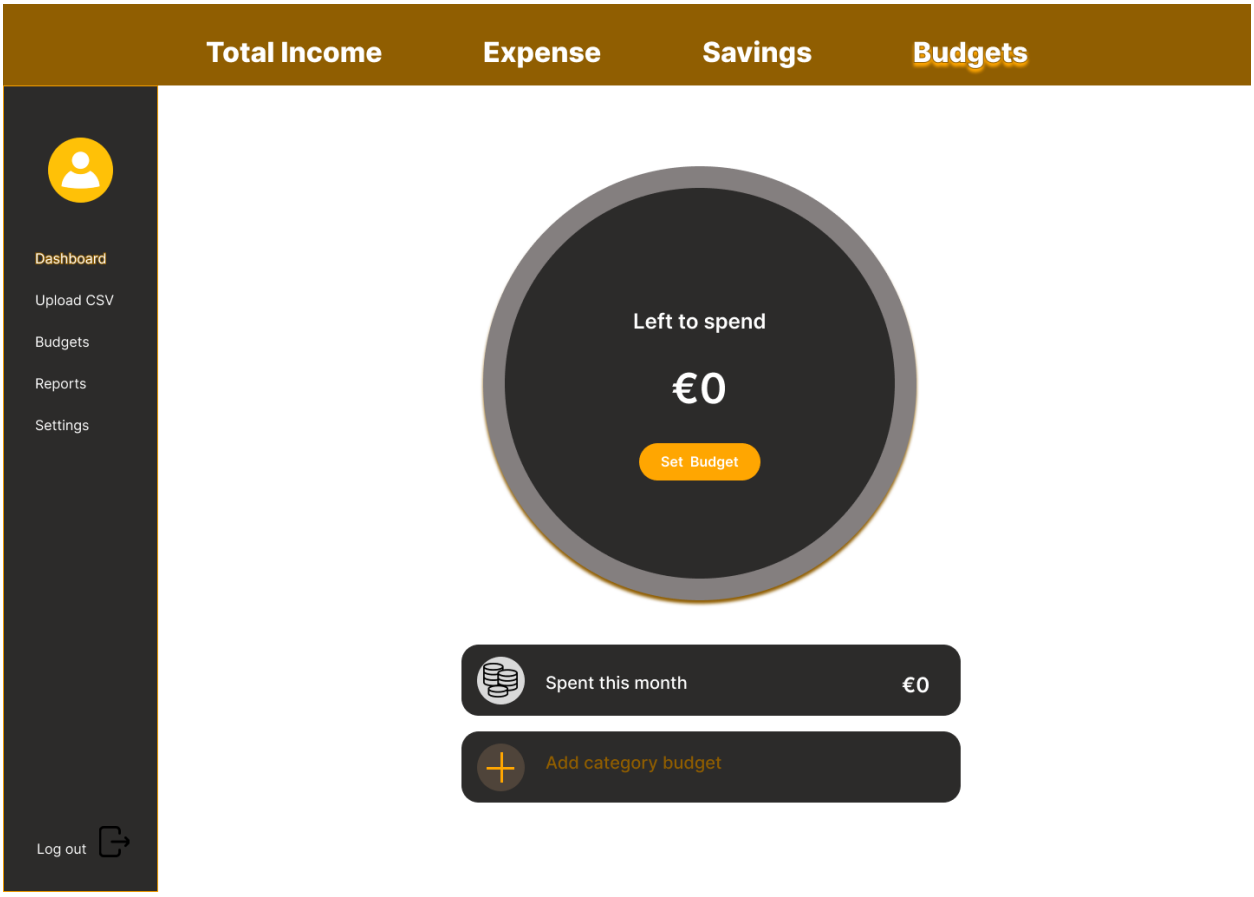

Figure 1: Budget screen mock design

CSV upload screen

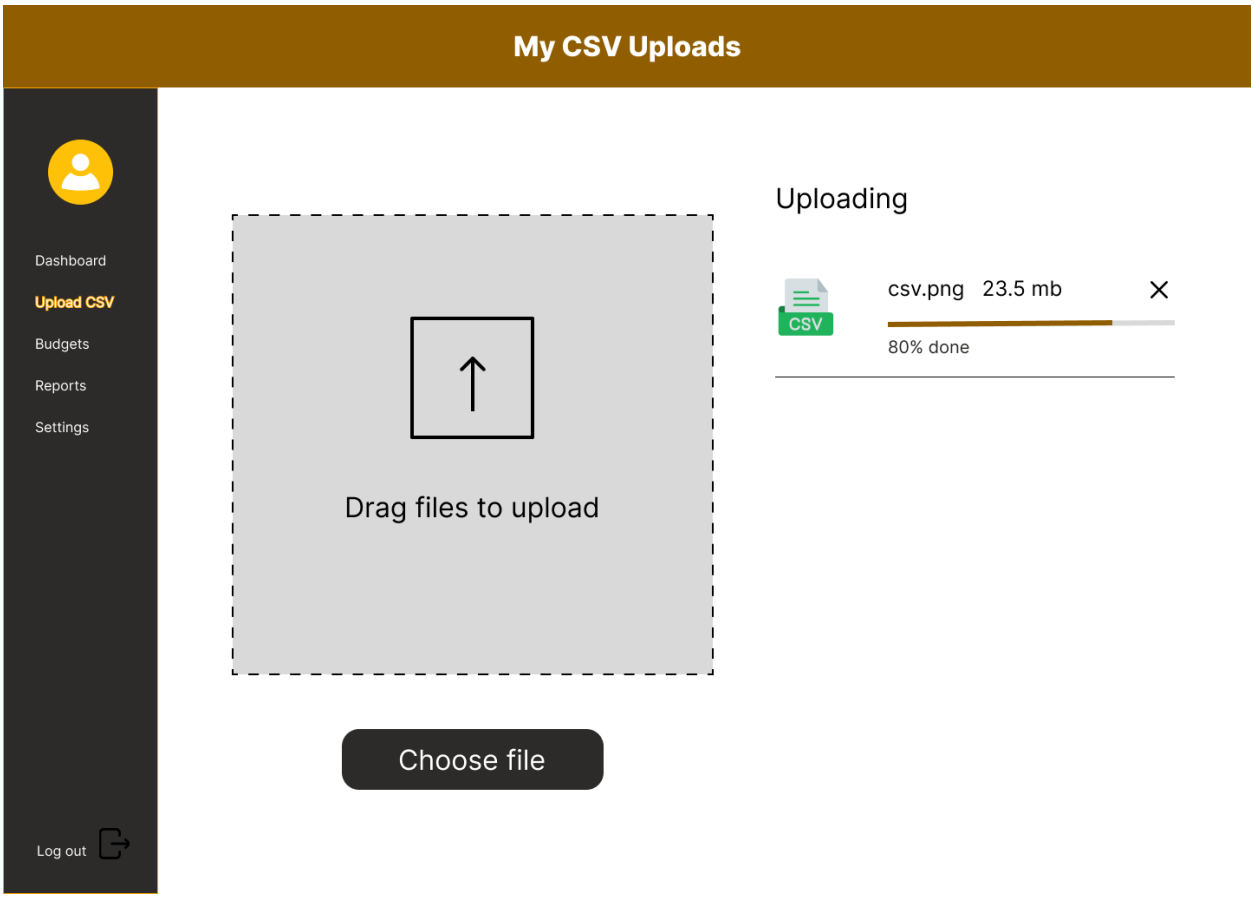

Figure 2: CSV upload screen mock design

Reports screen

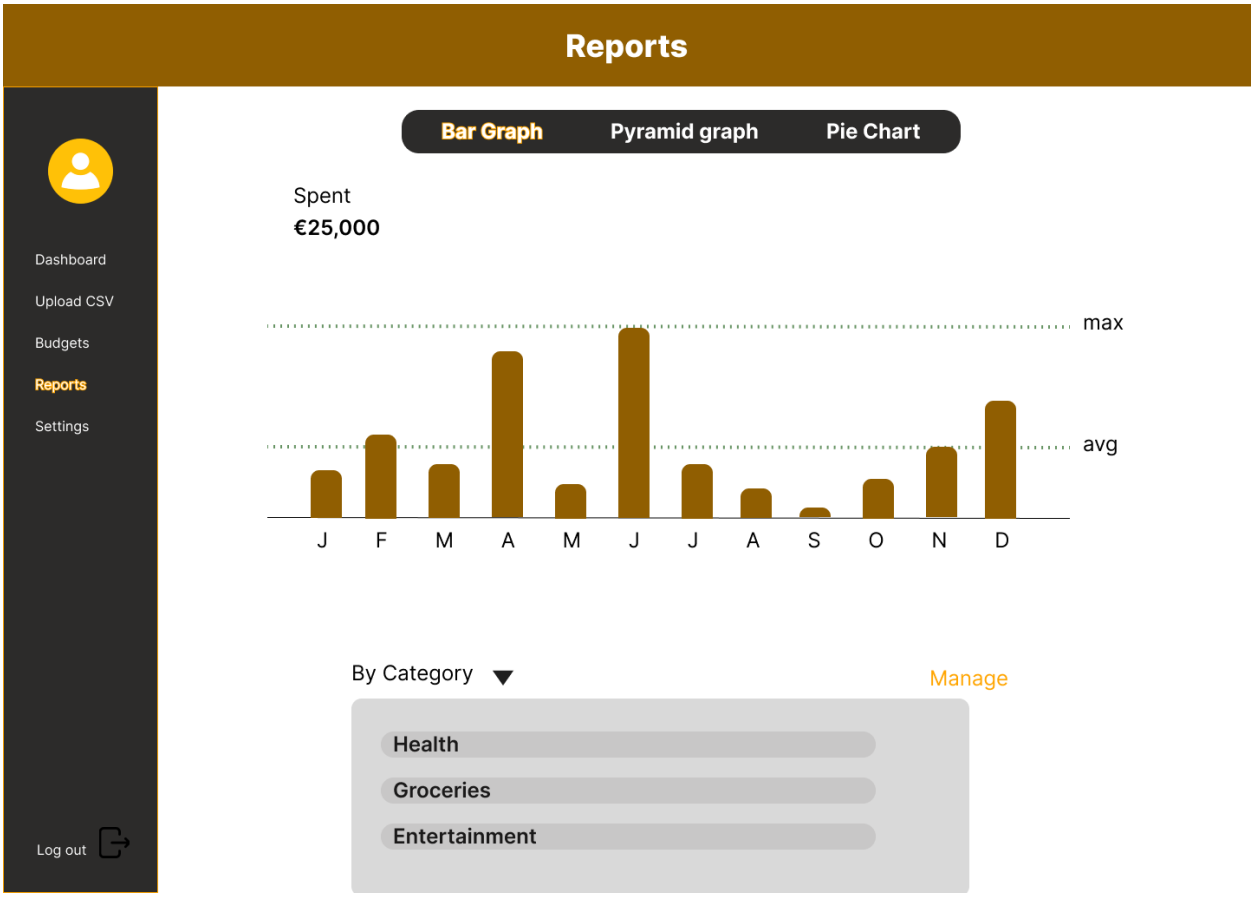

Figure 3: Financial Reports screen mock design

4. Major Architectural Components

4.1 Description of Connections

1. **Frontend (Client-Side Application):**
    - Built with a web framework (e.g., React.js) for the user interface.
    - Interacts directly with Firebase services (Firestore, Authentication, and Storage).
    - Sends user requests (e.g., upload CSV, set budgets) and displays data from Firestore in real-time.
2. **Backend Services (FastAPI):**
    - **Authentication:** Works with Firebase Authentication to handle secure user login and registration.
    - **Database (Firestore):** Stores and retrieves user financial data (transactions, budgets, settings).
    - **Data Processing:**
        i. FastAPI endpoints handle CSV uploads, transaction parsing, and categorization.
        ii. Executes ML-based predictive financial insights.
        iii. Generates reports and visualizations.
    - **API Communication:**
        i. FastAPI exposes RESTful endpoints for frontend interactions.
        ii. Handles Open Banking API integration for secure transaction syncing.
    - **File Storage:** Manages uploaded CSV files and processes them efficiently.
3. **Data Processing and Analytics:**
    - **AI/ML Models:** Implemented in Firebase Cloud Functions for automatic transaction categorization and predictive financial insights.
    - Processes raw data from the frontend and updates Firestore with the results.
4. **User Notifications:**
    - Managed via Firebase Cloud Messaging (FCM) to send alerts, reminders, or updates directly to users (e.g., "Budget exceeded").

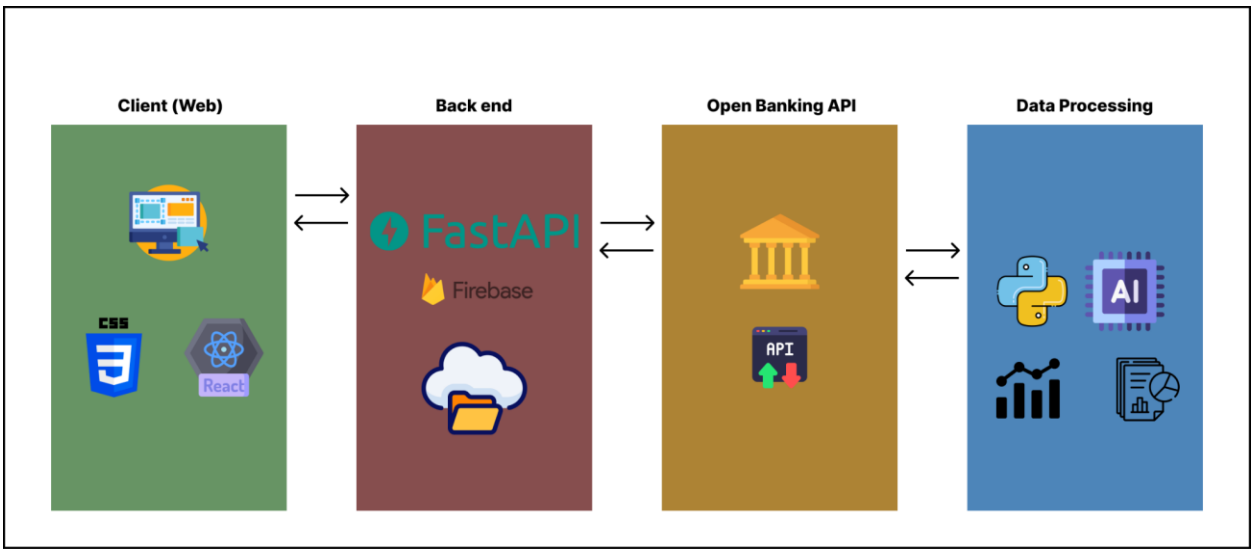

Figure 3: Architectural Diagram

Flow Explanation

Frontend:

1. Users authenticate and upload CSV files via the web interface.
2. Requests for reports, budgets, and transaction insights are sent to the FastAPI backend.
3. Updates from Firebase are reflected in real time on the UI.
4. Users will link their bank account via a secure authentication process (OAuth 2.0). They select their bank from a list, log in, and authorize the app to access their data.

Backend (Open Banking API Integration):

1. FastAPI communicates with the Open Banking API to fetch transaction data.
2. The data is retrieved securely and stored in Firestore for processing.

Data Processing:

1. FastAPI categorizes the fetched transaction data (e.g., income, groceries, entertainment) and integrates it with the user's overall financial records.

Frontend Updates:

1. Users see their updated financial data (from both manual CSV uploads and Open Banking transactions) in real-time on their dashboards.
2. Handles Open Banking API integration for secure transaction syncing.
3. Manages uploaded CSV files and processes them efficiently.

4.2 Database Schema

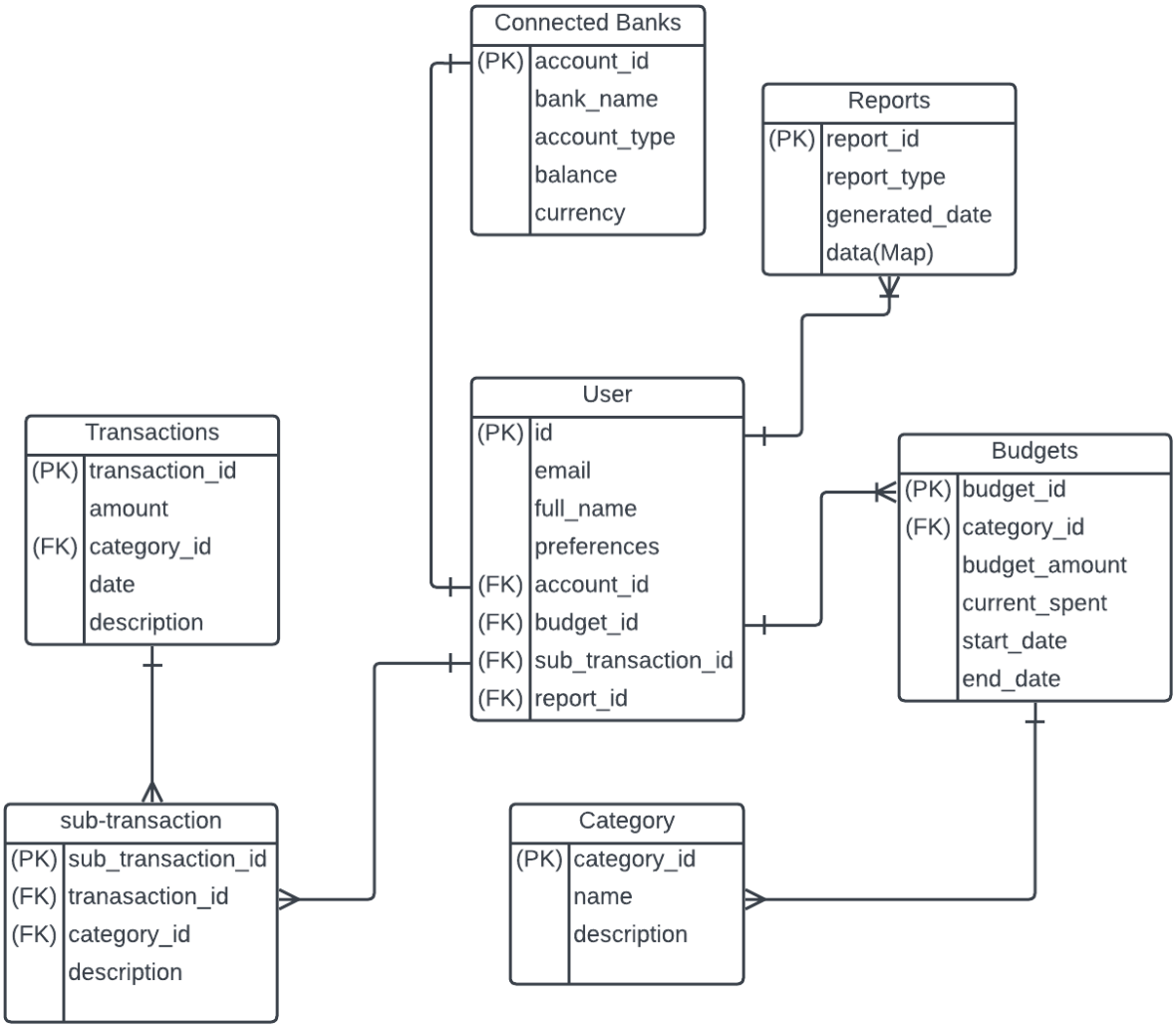

Figure 4: Database Schema Diagram

4.3 Component Diagram (Class Diagram)

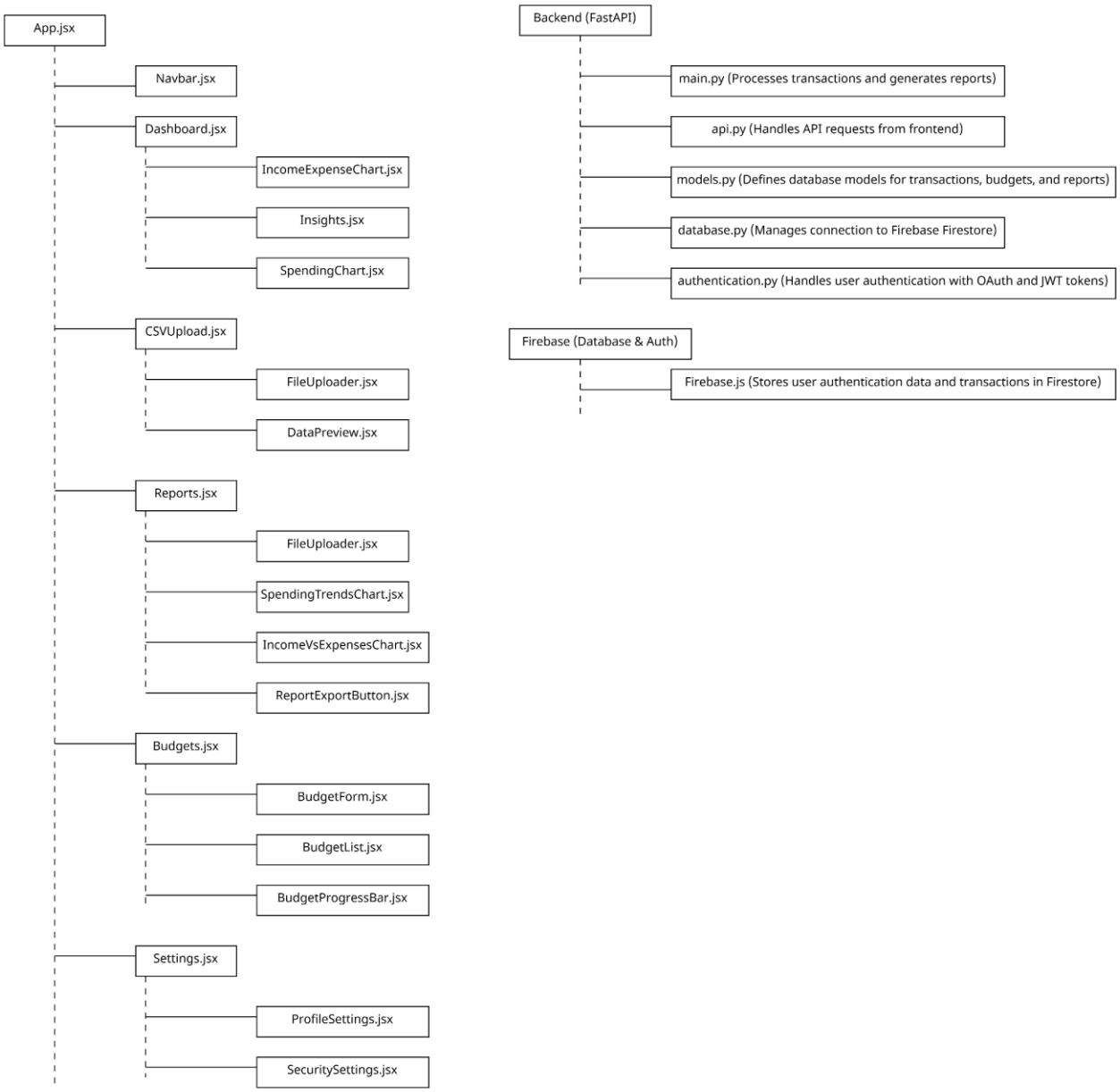

Figure 5: Component Diagram

4.4 Sequence Diagrams

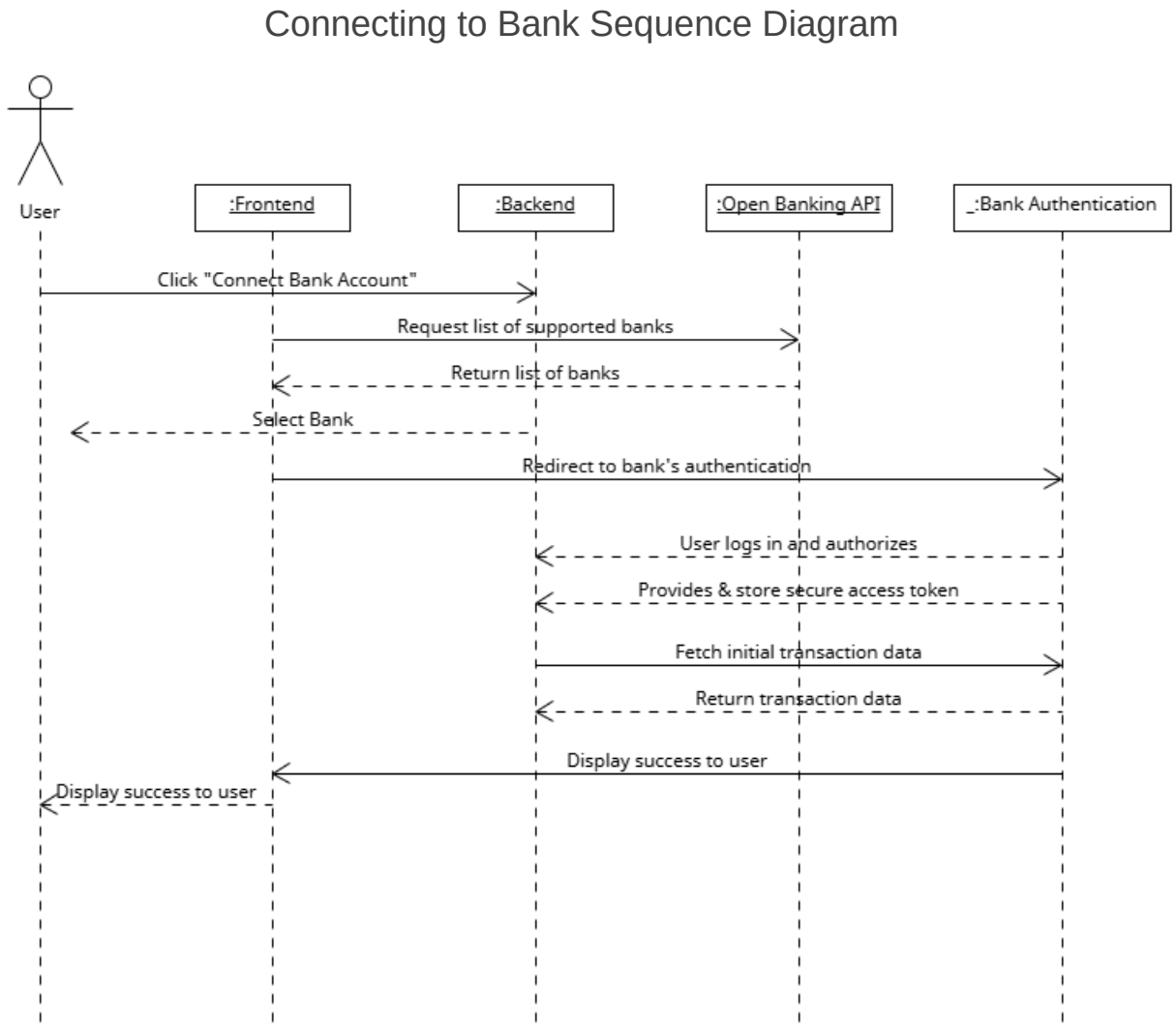

Figure 6: Bank Connection Sequence Diagram

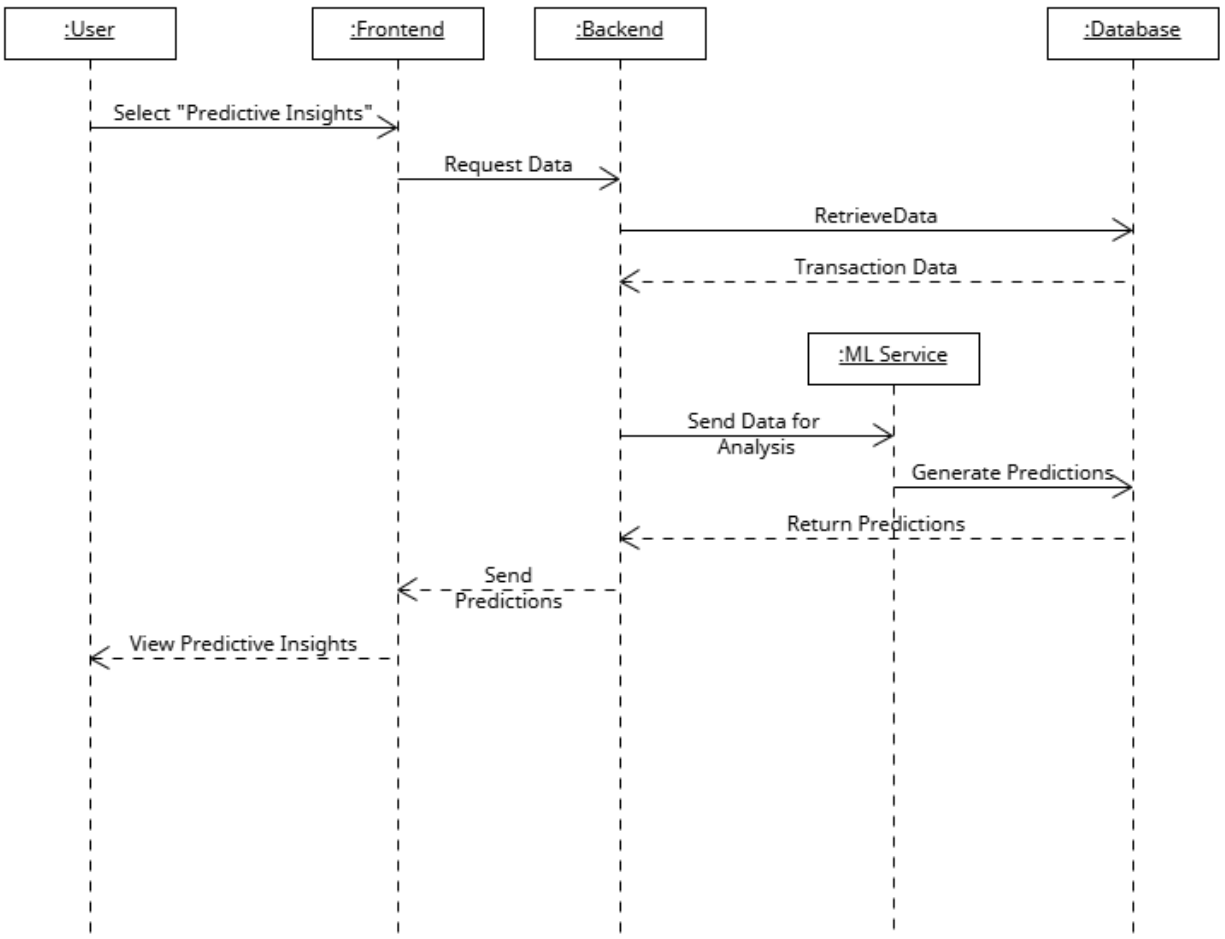

Figure 7: ML Predictions Sequence Diagram

View Financial Reports Sequence Diagram

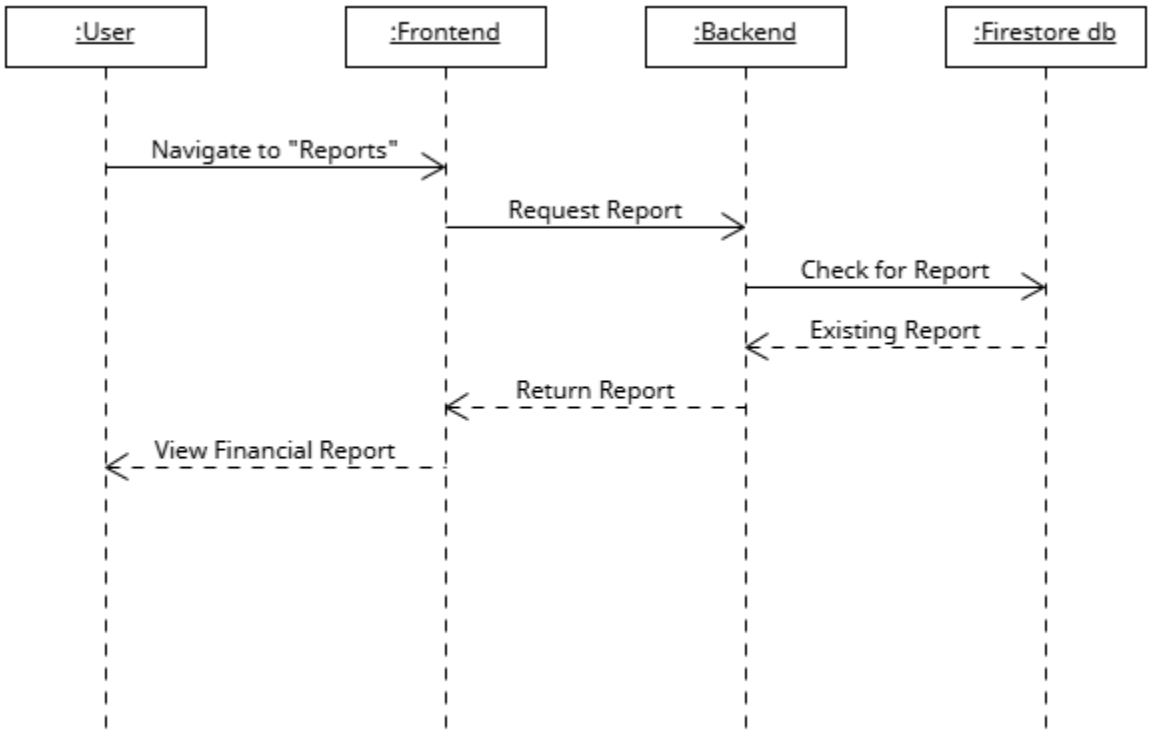

Figure 8: ML Predictions Sequence Diagram

---

4. Detailed Use cases

Name	Predictive Financial Insights
Actor(s)	Registered User
Description	This use case begins when a user wishes to receive predictions about their future financial performance based on historical data. The system processes the user's uploaded transactions, applies data analysis techniques, and forecasts trends such as future spending, income growth, and potential budget overruns. Users can view these insights through charts and recommendations to better plan their financial decisions.
Main Success Scenario:	1. The user uploads a CSV file with financial transactions or manually enters their data. 2. The system processes the data using financial algorithms and AI models. 3. The user selects the "Predictive Insights" feature. 4. The system analyzes trends and generates predictions on future spending, income, and potential savings. 5. The user views detailed charts and recommendations for financial planning. 6. Predictions are stored and updated as new data is uploaded.
Alternatives	3.a **The system detects incomplete or corrupted data in the CSV file.** ● It notifies the user to correct or re-upload the data. 5.a **The user finds that predictions are not aligned with their financial habits.** ● The user can adjust the algorithm's settings or manually fine-tune the insights. 6.a **The system fails to connect to the cloud for prediction storage.**

Name	Connecting to Bank
Actor(s)	Registered User, Bank API
Description	This use case begins when a user wishes to connect their bank account to the Personal Finance Tracker app to enable automatic transaction syncing. The system interacts with an Open Banking API, allowing the user to authenticate and authorize access to their bank data. Upon successful connection, the system retrieves and processes the user's transactions, updating the dashboard and enabling seamless financial tracking.

Main Success Scenario:	1. The user logs into the application. 2. The user selects "Connect Bank Account." 3. The user selects their bank. 4. The user logs into their bank and authorizes access. 5. The system retrieves transaction data. 6. The system updates the user dashboard. 7. The system confirms the connection.
Alternatives	**3.a The user's selected bank is not supported.** ● The system displays a message: "This bank is not supported. Please try another bank or use manual CSV upload." ● The user can return to the bank selection screen or exit the process. **4.a The user denies authorization at the bank's authentication portal.** ● The system receives a denial response from the Open Banking API. ● The system notifies the user: "Bank connection canceled. You can retry the process anytime." **5.a The Open Banking API fails to fetch transaction data.** ● The system displays a message: "Unable to retrieve transactions at this time. Please try again later." ● The user can proceed without automatic syncing or retry the operation. **6.a The system detects an expired or invalid authentication token.** ● The system prompts the user to reauthorize their bank account connection. ● The user is redirected to the bank's authentication portal to generate a new token.

5. Project Plan

Iteration 2: Final Design Specification and Second Product Release

Timeline: December 7, 2024 – February 14, 2025
Goal: Implement the core backend and frontend functionality to produce a working version of the app for budget management and CSV uploads.

Phases, Milestones, and Deliverables

Phase 1: Backend Development (Weeks 1-3, Dec 7–Dec 27)

- Set up Firebase services (Firestore, Authentication, Cloud Functions).
- Create database collections (users, transactions, budgets).
- Implement CSV upload and parsing.
  Deliverables:
- Fully functional Firebase backend with CSV processing.

Phase 2: Frontend Development (Weeks 4-6, Dec 28–Jan 17)

- Develop a basic dashboard with summary cards (income, expenses, budgets).
- Integrate CSV upload functionality with the backend.
  Deliverables:
- Web app with functional CSV upload and financial summary display.

Phase 3: Testing and Deployment (Weeks 7-8, Jan 18–Feb 7)

- Conduct basic functionality tests.
- Deploy the updated app with core features (CSV processing, budget tracking).
  Deliverables:
- Working web app deployed with backend integration.

Phase 4: Final Design Specification (Week 9, Feb 8–Feb 14)

- Document final design specifications.
  Deliverables:
- Final design document detailing backend, frontend, and data flow.

Iteration 3: Final Report and Final Product Release

Timeline: February 15 – April 28, 2025
Goal: Complete the app with advanced features, final testing, and reporting.

Phases, Milestones, and Deliverables

Phase 1: Advanced Features Development (Weeks 1-6, Feb 15–Mar 28)

- Implement Open Banking API integration (simulated initially).
- Add predictive financial insights and AI-based transaction categorization.
- Develop visual reporting (charts for spending trends, budget analysis).
  Deliverables:
- Integrated bank connection simulation.
- Predictive insights feature.
- Detailed reports section in the app.

Phase 2: Testing and Optimization (Weeks 7-8, Mar 29–Apr 12)

- Conduct extensive unit, integration, and usability testing.
- Optimize performance and address bugs.
  Deliverables:
- Bug-free, optimized application.

Phase 3: Finalization and Deployment (Weeks 9-10, Apr 13–Apr 28)

- Prepare and deliver the final project report.
- Deploy the completed app with all features.
- Present a polished demo.
  Deliverables:
- Final report.
- Fully functional web app with all features deployed.
- Final presentation/demo.

---

6. Methodology

I'm following a structured methodology to ensure the project meets its goals. By combining Agile practices with an iterative development approach, I can break the project into manageable chunks and deliver tangible results at every step.

1. Iterative Development [1]

Each iteration involves six key steps that keep me on track:

- **Requirement Gathering:**
  I start by defining what I need to accomplish in each iteration. For example, Iteration 2 focuses on building the backend and integrating CSV uploads.
- **Planning:**
  I break the requirements into smaller tasks, like setting up Firebase or designing the dashboard UI.
- **Implementation:**
  I develop and test each feature individually before bringing everything together.
- **Integration and Testing:**
  Once the features are ready, I test how they work together to make sure there are no issues.
- **Deployment:**
  I deploy incremental versions of the app so that progress is visible and feedback can be gathered.
- **Review and Feedback:**
  I reflect on what worked, what didn't, and how I can improve for the next iteration.

---

2. Analysis of Methodology [2]

Strengths:

- **Incremental Progress:** This approach ensures I can see tangible results at every stage.
- **Adaptability:** I can adjust the scope based on feedback or unforeseen challenges.
- **Real-World Testing:** Using simulated APIs and predictive models allows me to create realistic scenarios and test the app thoroughly.

Weaknesses:

- **Complex Features:** Advanced functionalities like predictive insights may take longer to refine.
- **Time Dependency:** Integrating external APIs (like Open Banking) could face delays if the APIs or their requirements change.

References

Resources/ Tools used:

[1], Asana, 2024. *Agile Methodology: What it is, how it works, and why it matters*. Available at: https://asana.com/resources/agile-methodology [Accessed 27 November 2024].

[2], Wikipedia, 2024. *Iterative and incremental development*. Available at: https://en.wikipedia.org/wiki/Iterative_and_incremental_development [Accessed 27 November 2024].

Flaticon
Flaticon, 2024. *Flaticon: Free icons for your projects*. Available at: https://www.flaticon.com/ [Accessed 23 November 2024].

Figma
Figma, 2024. *Figma: The collaborative interface design tool*. Available at: https://www.figma.com/ [Accessed 23 November 2024].

ChatGPT
OpenAI, 2024. *ChatGPT: AI language model for various applications*. Available at: https://chatgpt.com/ [Accessed 23 November 2024].