**Design and**

**Implementation**

**for**

**Smart Expense**

**Tracker Application**

**Version 0.1**

**Prepared by Group D2**

**Tang Bok Hei (1155193297),**

**Siu Wing Yiu (1155192075),**

**Leung Ho Chun (1155193014),**

**So Kin Pui (1155193506)**

{Date}

Department of Computer Science and Engineering

Table of Contents

[1. Introduction 3](#_Toc192533033)

[2. System Architecture 3](#_Toc192533034)

[2.1 Major Components 3](#_Toc192533035)

[2.2 Component Interactions 4](#_Toc192533036)

[3. Data Models 4](#_Toc192533037)

[3.1 Entity Relationship Diagram (Visualizer in Supabase website) 4](#_Toc192533038)

[4. Interface Design 5](#_Toc192533039)

[4.1 API Endpoint Specifications (As of now) 5](#_Toc192533040)

[4.2 Error Handling 5](#_Toc192533041)

[5. Component Design 6](#_Toc192533042)

[5.1 Frontend Component Structure 6](#_Toc192533043)

[5.2 Backend Component Structure 6](#_Toc192533044)

[6. Components Design 7](#_Toc192533045)

[6.1 Sitemap 7](#_Toc192533046)

[6.2 Components Storyboard 8](#_Toc192533047)

[6.2.1 Login Page 8](#_Toc192533048)

[6.2.2 Sign Up page 9](#_Toc192533049)

[6.2.4 Expense Manager 11](#_Toc192533050)

[6.2.5 Reports 14](#_Toc192533051)

[6.2.6 Settings 15](#_Toc192533052)

[7. Assumptions and Constraints 17](#_Toc192533053)

[7.1 Technical Constraints 17](#_Toc192533054)

[7.2 Development Constraints 17](#_Toc192533055)

[7.3 Authentication and Security 17](#_Toc192533056)

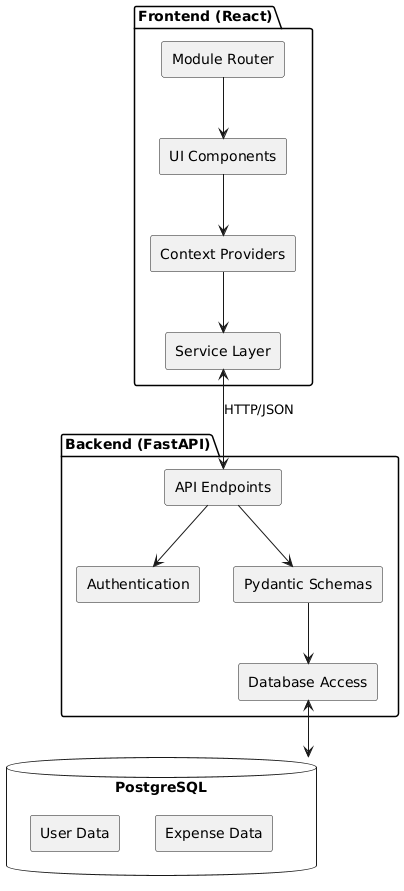
[7.4 Performance 17](#_Toc192533057)

# 1. Introduction

The Smart Expense Tracker Application (SETA) is designed to help users manage personal expenses with features for tracking, categorizing, and visualizing spending patterns. This document outlines the system design and implementation details.

# 2. System Architecture

## 2.1 Major Components



## 2.2 Component Interactions

* **Frontend**: React-based single-page application using Material UI (MUI)
* **Backend**: FastAPI-based REST API service with SQLAlchemy ORM
* **Database**: PostgreSQL hosted on Supabase
* **Communication**: HTTP/REST with JSON data

# 3. Data Models

## A screenshot of a computer AI-generated content may be incorrect.

## 3.1 Entity Relationship Diagram (Visualizer in Supabase website

# 4. Interface Design

## 4.1 API Endpoint Specifications (As of now)

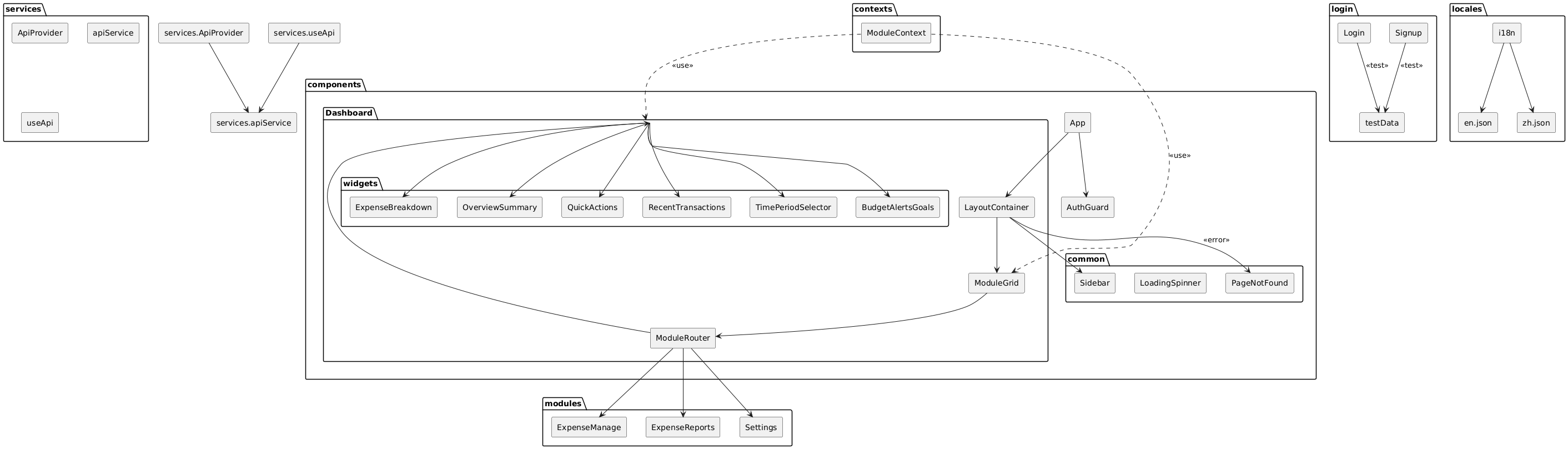
| **Endpoint** | **Method** | **Description** | **Input** | **Output** | **Status Codes** |
| --- | --- | --- | --- | --- | --- |
| /login | POST | User authentication | [**UserLogin**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | [**UserResponse**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | 200, 401, 404 |
| /signup | POST | User registration | [**UserCreate**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | [**UserResponse**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | 201, 400 |
| /expenses/{user\_id} | GET | Get user's expenses | [**user\_id**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | [**List[ExpenseResponse]**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | 200, 404 |
| /expenses | POST | Create new expense | [**CreateExpense**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | [**ExpenseResponse**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | 201, 404 |
| /expenses/{expense\_id} | DELETE | Delete expense | [**expense\_id**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | **-** | 204, 404 |
| /users/{user\_id}/settings | GET | Get user settings | [**user\_id**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | [**UserSettings**](vscode-file://vscode-app/c:/Users/wingy/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) | 200, 404 |

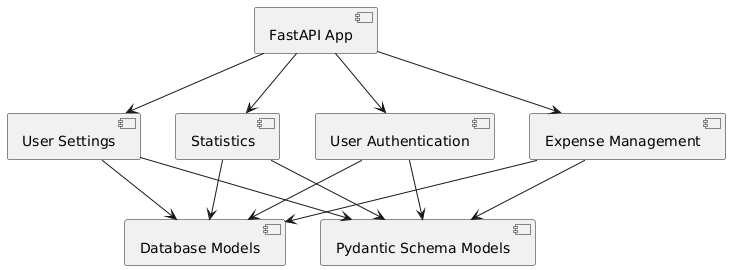
## 4.2 Error Handling

* HTTP status codes for common errors
* Detailed error messages in command lines that runs the API
* Frontend error notification system via specific components (e.g. Grid, Notification Bar)

# 5. Component Design

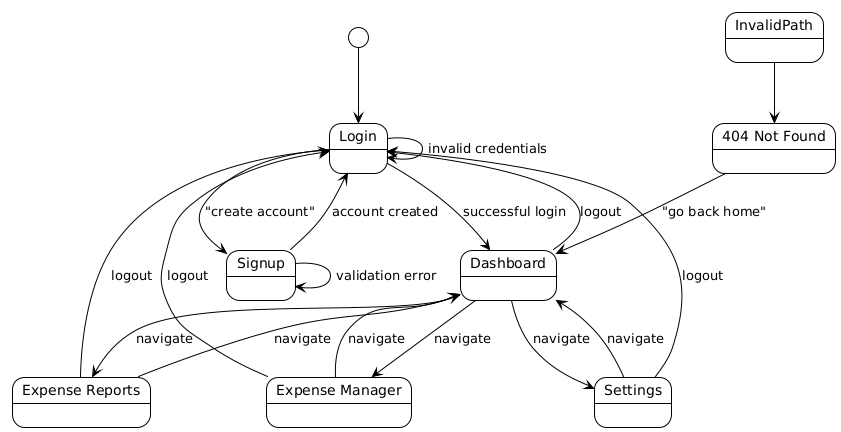
## 5.1 Frontend Component Structure

5.2 Backend Component Structure



# 6. Components Design

## 6.1 Sitemap



## 6.2 Components Storyboard

### 6.2.1 Login Page

#### 1. State Management

|  |  |  |
| --- | --- | --- |
| **States** | **Set** | **Usage** |
| *isLoading* | *true*: clicking the “Sign in” button  *false*: after getting response from backend | For conditionally rendering the LoadingSpinner component and to set the disable value of the “Sign in” button |
| *showPassword* | true/false: toggled when clicking the “show password” icon | Handle the visibility of the password |
| *error* | “*Invalid username or password”,*  “*User not found*”,  “*Account is disabled*”,  “*Login failed. Please try again.*”,  “*An error occurred during login*”  These error messages are set depending on the error codes receive from the backend | An error message will be conditionally rendered using the Alert component of the MUI library |
| *formData* | username: *null (default)*  password: *null (default)*  Will be set to the value input in username and password text boxes | Store credential inputs |

A diagram of a software program

AI-generated content may be incorrect.UML diagram to show the structure and general flow of the login page.

### 6.2.2 Sign Up page

#### 1. State Management

|  |  |  |
| --- | --- | --- |
| **States** | **Set** | **Usage** |
| *isLoading* | *true*: clicking the “Create Account” button  *false*: after getting response from backend | For conditionally redering the LoadingSpinner component and to set the disable value of the “Create Account” button |
| *showPassword* | true/false: toggled when clicking the “show password” icon | Handle the visibility of the password |
| *showRePassword* | true/false: toggled when clicking the “show password” icon for the re-entered password | Handle the visibility of the re-entered password |
| *error* | Error messages from frontend (for validation checks): *“Username should not contain special characters”,*  *“Please enter a valid email address”,*  *“Contact number must be 8 digits”,*  *“Password must contain at least 8 characters, an upper-case letter, a lower-case letter, a number and a special symbol”,*  *“Passwords do not match”,*  *“{field name} is required”*  Error messages from backend:  *“Username already exists”,*  *“Signup failed. Please try again”,*  *“Server not responding. Please try again later”,*  *“An error occurred during signup”* | Error messages will be displayed on top of each corresponding text box.  The state variable is an object with the field names such as username, email, … as the index |
| *formData* | username: *null (default)*  first\_name: *null (default)*  last\_name: *null (default)*  email: *null (default)*  contact\_number: *null (default)*  password: *null (default)*  rePassword: *null (default)*  Will be set to the value input in the required text boxes | Store the input values of each field |

A screenshot of a diagram

AI-generated content may be incorrect.

UML diagram to show the structure and general flow of the sign up page.

### 6.2.3 Dashboard

* **TimePeriodSelector**: Allows users to select a time period (Week, Month, Quarter, Custom) for filtering data.
* **OverviewSummary**: Displays key metrics like Total Income, Total Expenses, Net Balance, and Savings Progress.
* **ExpenseBreakdown**: Shows a pie chart of expense categories and a table of top expenses.
* **RecentTransactions**: Lists recent transactions with details.
* **BudgetAlertsGoals**: Displays budget alerts and savings goal progress.
* **QuickActions**: Provides buttons for quick actions like adding a new expense or generating a report.

**1. State Management**

|  |  |  |
| --- | --- | --- |
| **State Variable** | **Update Function** | **Purpose** |
| expenses | setExpenses | Stores the list of expenses fetched from the API. |
| totalExpense | setTotalExpense | Stores the total expenses calculated from the API. |
| loading | setLoading | Controls the loading state of the Dashboard. |
| timePeriod | setTimePeriod | Stores the selected time period for filtering expenses. |
| error | setError | Stores any error messages from API calls. |

**2. API Integration**

* **Endpoint**: /expenses/{user\_id}
  + **Method**: GET
  + **Description**: Fetches all expenses for the logged-in user.
  + **Input**: user\_id
  + **Output**: JSON array of expense objects.
* **Endpoint**: /expenses/{user\_id}/total
  + **Method**: GET
  + **Description**: Fetches the total expenses for the logged-in user.
  + **Input**: user\_id
  + **Output**: JSON object with total field.

**3. Use Cases**

|  |  |
| --- | --- |
| **Use Case** | **Description** |
| **Viewing Summary Metrics** | User views key financial metrics (e.g., total income, expenses, net balance). |
| **Analyzing Expenses** | User views a breakdown of expenses by category and top expenses. |
| **Monitoring Transactions** | User reviews a list of recent transactions for quick reference. |
| **Setting Budget Goals** | User sets or updates budget goals and receives alerts for overspending. |
| **Filtering by Time Period** | User selects a time period (Week, Month, Quarter) to filter expenses. |

### 6.2.4 Expense Manager

**1. State Management**

|  |  |  |
| --- | --- | --- |
| **State Variable** | **Update Function** | **Purpose** |
| expenses | setExpenses | Array that temporarily stores expense records retrieved from the database |
| isLoading | setIsLoading | Boolean variable that controls whether the LoadingSpinner component is displayed while fetching data |
| isSubmitting | setIsSubmitting | Boolean that disables buttons in the confirm dialog during API interactions to prevent issues |
| apiError | setApiError | Stores error messages that occur during front-end to backend API interactions |
| notification | setNotification | Object controlling notification display with the following properties: open (boolean), message (string), and severity (“success”/” error”) |
| formData | setFormData | Object model for expenses with properties: amount, category\_name, date, and description |
| confirmDialogOpen | setConfirmDialogOpen | Boolean that controls whether the expense confirmation dialog is displayed |
| showOtherCategoryField | setShowOtherCategoryField | Boolean that controls whether the custom category input field appears when "Others" is selected |
| deleteDialogOpen | setDeleteDialogOpen | Boolean that controls whether the delete confirmation dialog is displayed |
| expenseToDelete | setExpenseToDelete | Stores the ID of the expense selected for deletion when the red trash bin icon is clicked |
| transactions | setTransactions | Stores the list of recent transactions. |
| categoryData | setCategoryData | Stores data for the pie chart (expenses by category). |
| topExpenses | setTopExpenses | Stores the top 5 expenses by amount. |

**2. API Integration and General Dataflow**

const API\_URL = 'http://localhost:8000';

Note: Our python backend API currently is set up to run at port 8000 of localhost.

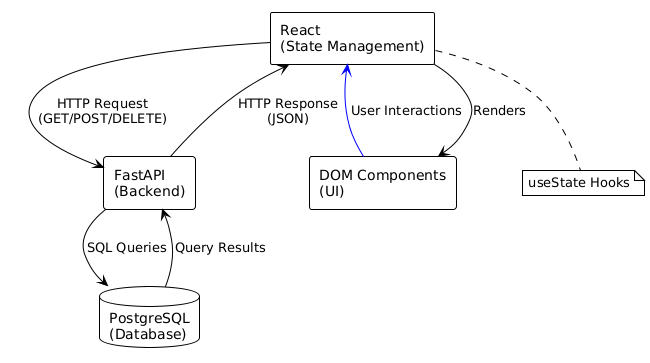
Dataflow example illustrated from fetching expense records from the database:

* 1. A get request will be sent to the python backend API by axios: const response = await axios.get(`${API\_URL}/expenses/${userId}`);
  2. Python backend API at the specified port receives the get request, and then performs a query to the database and the response dataset is stored in variable “expense”: expenses = db.query(models.Expense).filter(models.Expense.user\_id == user\_id).all()

return expenses

* 1. “response” variable in step 1 will receive the metadata from “expenses” in step 2, thus returning the data to the DOM component that requested it.

UML Diagram to illustrate the general dataflow:



**3. Use Cases**

|  |  |
| --- | --- |
| **Use Case** | **Description** |
| **Adding a New Expense** | **User enters details of a new expense (amount, category, date, description), and confirms the entry by reviewing the inputted information to record the transaction in their expense history by clicking submit again.** |
| **Viewing Expense History** | **User accesses a list of all expenses they have previously recorded, with complete details organized in a concise table format for quick and easy scanning and reference.** |
| **Deleting an Expense** | **User identifies an incorrect or unwanted expense entry in their history and removes it from the system after confirming the deletion action.** |
| **Monitoring Total Expenditure** | **User views the aggregated sum of all recorded expenses at a glance through the summary cards at the top of the screen to understand their overall spending in an intuitive way.** |
| **Viewing Budget Alerts** | User sees alerts for categories where spending is nearing or exceeding limits. |
| **Tracking Savings Goals** | User tracks progress toward savings goals with a progress bar. |
| **Viewing Expense Breakdown** | User sees a pie chart of expenses by category and a table of top expenses. |

### 6.2.5 Reports

1. **ExpenseReportPage**: The main component that handles the page logic

* Contains basic methods for loading, filtering, and exporting data

1. **ExpenseTable**: Simple table to display the expense data

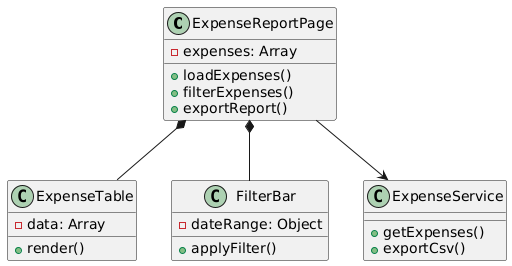
* Just renders the expense information

1. **FilterBar**: Basic filtering interface

* Primarily for date range selection

1. **ExpenseService**: Service for data operations

* Fetches expense data and handles exports



### 6.2.6 Settings

The Settings page will show the user profile details (first name, last name, contact number, username and email address) and the user can modify their profile information as well as their account password. Also, the user can change the theme (light, dark or system default), language (English or Chinese) in this page.

#### 1. State Management

|  |  |  |
| --- | --- | --- |
| **States** | **Set** | **Usage** |
| *isLoading* | *true*: when loading user profile data from API  *false*: after profile data is loaded or when an error occurs | For conditionally rendering the LoadingSpinner component and to disable form inputs |
| *error* | null by default,  “*Failed to load profile data. Please try again later*” when failed to fetch data from API | Conditionally rendering an Alert component with the error message |
| *snackbarOpen* | *true:* after saving settings, changing password, or encountering errors  *false*: when user closes the snackbar or after timeout | Controls visibility of the feedback Snackbar component |
| *snackbarMessage* | Success messages*: "Settings saved successfully!", "Password changed successfully!"*  Error messages*: "Failed to load profile data", "Failed to save settings", "Please correct the errors in the form"* | Display text content within the Snackbar’s Alert component |
| *snackbarSeverity* | "*success*": for successful operations  "*error*": when operations fail | Controls visibility of the password change Dialog component |
| *passwordDialogOpen* | *true*: when clicking "Change Password" button  *false*: when closing the dialog or after successful password change | Controls visibility of the password change Dialog component |
| *passwordData* | Initially: currentPasword: “”,  newPassword: “”,  confirmPassword: “”  Updated via handlePasswordDataChange function when user types in password fields | Provides values for password Dialog form fields |
| *passwordErrors* | Same as above initially,  Set to validation messages when password fields fail validation | Cleared for a specific field when user edits that field.  Error text display below password form fields.  Determines whether fields show error state |
| *formErrors* | Initially: username: “”  email: “”,  contactNumber: “”,  firstName: “”,  lastName: “”  Set to the validation messages when failed validation | Same as the above |
| budgetAlerts | setBudgetAlerts | Stores budget alerts for overspending or nearing limits. |
| savingsGoals | setSavingsGoals | Stores the user's savings goals and progress. |

A screenshot of a computer

AI-generated content may be incorrect.UML diagram to show the structure and general flow of the Settings page.

# 7. Assumptions and Constraints

## 7.1 Technical Constraints

* Browser compatibility: Modern browsers (Chrome, Firefox, Safari, Edge)
* Responsive design for desktop and tablets (mobile support limited, layouts are not optimized for mobile view)

## 7.2 Development Constraints

* Four developers
* Limited backend infrastructure in this version
* PostgreSQL database hosted on Supabase, if Supabase is down then the product will not work
* No CI/CD pipeline just yet (Docker will be implemented if time allows)

## 7.3 Authentication and Security

* Session-based authentication with timeout (currently set to 30 minutes)
* Password hashing using SHA-256
* No password recovery mechanism in current version

## 7.4 Performance

* Designed for small to moderate number of expenses
* No optimizations for large datasets
* Limited reporting capabilities, its function is not fully developed

References

1. **PlantUML**. (n.d.). *PlantUML - Open-source tool for creating UML diagrams from text*. Retrieved from [https://plantuml.com](https://plantuml.com/)
   1. Used for generating UML diagrams from textual descriptions.
2. DeepSeek. (n.d.). *AI-generated PlantUML code for UML diagrams*.
   1. The PlantUML code used in this document was generated with the assistance of an AI tool to represent the frontend, backend component structure and system architecture.
3. **Material-UI (MUI)**. (n.d.). *React components for faster and easier web development*. Retrieved from [https://mui.com](https://mui.com/)
   1. Used for building the frontend components of the Smart Expense Tracker Application.
4. **FastAPI**. (n.d.). *FastAPI - Modern, fast (high-performance), web framework for building APIs with Python*. Retrieved from [https://fastapi.tiangolo.com](https://fastapi.tiangolo.com/)
   1. Used for building the backend API of the Smart Expense Tracker Application.
5. **Supabase**. (n.d.). *Supabase - Open-source Firebase alternative*. Retrieved from [https://supabase.com](https://supabase.com/)
   1. Used for hosting the PostgreSQL database and providing backend services.
6. **React**. (n.d.). *React - A JavaScript library for building user interfaces*. Retrieved from [https://reactjs.org](https://reactjs.org/)
   1. Used for building the frontend of the Smart Expense Tracker Application.