SpendSmart - Your Personal Finance Companion

SmartBridge MERN Stack Project

Documentation

Submitted by

22BCE10385 Kavya
22BCE10914 Simarpreet Singh
22BHI10185 Nehal Nisha
22BCE10181 Shreenu Sutar

Team ID: SWTID1744280958

VIT Bhopal University
Sehore – 466114
Madhya Pradesh

April 2025

1. Introduction

1.1 Project Overview

The **SpendSmart -Expense Tracker Application** is a full-stack web-based software project that has been developed using the **MERN stack**, which includes **MongoDB**, **Express.js**, **React.js**, and **Node.js**. This combination of technologies enables the creation of a responsive, efficient, and dynamic user experience. The main purpose of this application is to help users keep track of their personal finances in an organized and accessible manner.

The application allows users to **record their income and expense transactions**, categorize them appropriately, and monitor their financial activities over time. It offers features that let users not only store this data securely but also **track and visualize their financial records in real-time**, providing immediate feedback on their spending and saving patterns.

It comes with a **user-friendly interface** designed to make navigation and usage simple for individuals with varying levels of technical expertise. The intuitive design ensures that users can easily add new transactions, view their financial summaries, and understand their monetary flow with clarity.

A key aspect of the application is its **secure login system**, which ensures that each user's data is protected and accessible only to them. By requiring user authentication, it maintains the privacy and integrity of personal financial information.

In addition, the application features **dynamic reporting capabilities**, allowing users to view their income and expenses in the form of lists and visual summaries such as charts or graphs. These features make it easier for users to analyse their financial health and make informed decisions based on accurate and up-to-date data.

Team Members and Their Roles

1. **Simarpreet Singh (22BCE10914)**
   Role: Documentation Lead & Frontend Developer
   Responsibilities:
   - **Documentation Lead:** Authored comprehensive documentation, including setup instructions, usage guidelines, and API references.
   - **Frontend Development:** Designed and developed responsive UI components and pages using React.js.

2. **Kavya (22BCE10385)**
   Role: Full Stack Developer & Documentation Contributor
   Responsibilities:
   - **Frontend Development:** Developed and integrated user interfaces using React.js.
   - **Backend Development:** Implemented API endpoints and managed database operations with MongoDB and Mongoose.
   - **Documentation:** Contributed to the documentation, focusing on both frontend and backend aspects.

3. **Shreenu Sutar**
   Role: Backend Developer & Documentation Contributor
   Responsibilities:
   - **Backend Development:** Developed server-side logic, including API endpoints and middleware using Express.js.
   - **Documentation:** Assisted in documenting backend architecture, API endpoints, and database schema.

4. **Nehal Nisha**
   Role: Backend Developer, Tester & Documentation Contributor
   Responsibilities:
   - **Backend Development:** Collaborated on developing backend services and integrating them with the frontend.
   - **Testing:** Conducted unit and integration tests to ensure application reliability and performance.
   - **Documentation:** Contributed to backend documentation, including testing procedures and API documentation.

2. Project Overview

Struggling financially is a well-known issue among students. Due to the rise of energy cost, the price of daily necessities and food has also been increased tremendously. According to research from the National Union of Students (NUS) in July, in the United Kingdom, one in three students is left with less than £50 a month after paying rent and bills, and 96 per cent were cutting back on their spending as a result of the cost of living crisis (Staton, 2022). Many students ought to seek part-time jobs, in which the fields of work are not related to their fields of study, to support themselves financially. As a result, they need to balance out their work and their study to work effectively while maintaining acceptable grades at the universities. Eventually, budget management has become a crucial skill that every student must learn and practice during their years of study.

Budget management is a feature that can be seen in most of the banking applications, which allow users to view their monthly credit card usage in different categories, such as food, utility, entertainment, or travelling. From there, users can gain better insights into their spending habits and adjust accordingly to their budget. This is basically the whole concept of the application which is presented in this thesis. The aim of this project is to build a mobile application utilizing the MERN stack (MongoDB, Express, React, and Node.js) that helps students in keeping track of their spending habits. The friendly user-interface of this application makes it easier for students to log their costs and track their spending trends over time. They may use this information to see where they might reduce their spending and improve their financial decisions. The research problems that this project seeks to address include:

- How can the MERN stack be leveraged to create an efficient and user-friendly application for tracking student expenses?
- How can the MERN stack be used to create a scalable and secure application that can handle large volumes of financial data while maintaining user privacy?
- What features should the application include to make it user-friendly and efficient?

In this documentation, we will assess the functionality, usability, and security of the application and demonstrate how well it fulfills the requirements of users.

Key Features of SpendSmart

1. **Secure User Authentication**
    - Implements JWT-based authentication for secure user registration and login processes.
2. **Real-Time Expense Tracking**
    - Allows users to add and monitor expenses instantly, ensuring up-to-date financial records.
3. **Income Management**
    - Enables users to record and categorize various income sources for comprehensive financial tracking.
4. **Expense Categorization**
    - Provides options to categorize expenses (e.g., Food, Transport, Entertainment) for better analysis and budgeting.
5. **Detailed Transaction History**
    - Maintains a comprehensive log of all income and expense transactions for user review and planning.
6. **Data Visualization**
    - Offers visual representations of financial data through charts and graphs, facilitating easier understanding of spending patterns.
7. **User-Friendly Dashboard**
    - Features an intuitive dashboard displaying summaries of income, expenses, and overall financial health.
8. **Responsive Design**
    - Ensures a seamless user experience across various devices, including desktops, tablets, and smartphones.
9. **Profile Management**
    - Allows users to update personal information and manage account settings efficiently.

3. Architecture

Frontend Architecture (React.js)

The frontend of **SpendSmart** is built using **React.js**, a component-based JavaScript library. It provides a modular structure where different parts of the UI (login form, dashboard, income/expense form) are encapsulated into reusable components. Key aspects include:

- **Component-Based Structure**: All features such as login, register, add transaction, dashboard, and report generation are built as independent components.
- **Routing**: The app uses *react-router-dom* to navigate between pages like login, signup, dashboard, income, and expenses without refreshing the page.
- **State Management**: React's *useState*, *useContext*, or custom hooks manage state across components, ensuring reactive updates.
- **Form Validation**: Built-in form validations help prevent incorrect data entry during registration and transaction logging.
- **Responsive Design**: Styled-components or CSS modules are used for styling, ensuring mobile responsiveness and clean UI.

Backend Architecture (Node.js + Express.js)

The backend server is built with **Node.js** and **Express.js** to handle all the API operations. It acts as the middle layer between the frontend and the database. Its main functionalities include:

- **RESTful APIs**: Defined endpoints for handling authentication, adding, editing, deleting, and retrieving incomes and expenses.
- **Authentication**: Implements JWT (JSON Web Tokens) for secure, token-based user sessions.
- **Middleware**:
    - **Authentication Middleware** to protect private routes and ensure only authenticated users can access their data.
    - **Error-handling Middleware** to manage API failures gracefully.
- **Controllers & Routes**:
    - userRoutes.js, incomeRoutes.js, expenseRoutes.js manage endpoint definitions.
    - Controllers manage the logic for CRUD operations related to users, income, and expenses.

Database Schema and MongoDB Interactions

SpendSmart uses **MongoDB** via **Mongoose**, a popular ODM (Object Document Mapper), to manage the NoSQL database.

- **Schemas**:
    - UserSchema: Stores user credentials, email, password (hashed), and timestamps.
    - IncomeSchema: Includes fields like title, amount, category, description, userId, and date.
    - ExpenseSchema: Similar structure to Income, used for expense tracking.
- **Relations**: Though MongoDB is non-relational, user-specific documents (like incomes or expenses) are linked using the *userId* reference, which ensures data integrity and allows filtering by logged-in users.
- **MongoDB Atlas** is used for cloud database hosting, ensuring data is accessible, scalable, and secure.

4. Setup Instructions

1. Prerequisites

Ensure the following software is installed on your system:

• **Node.js (v14 or higher):** JavaScript runtime environment.

• **npm (Node Package Manager):** Usually comes with Node.js.

• **MongoDB:** NoSQL database for storing application data.

• **Git:** Version control system.

• **Code Editor:** Such as Visual Studio Code.

2. Installation Steps

a. Clone the Repository

Open your terminal and execute the following commands:

git clone https://github.com/CaptainCode2024/SpendSmartMERN.git

cd CodeBase

b. Set Up the Backend

1. **Navigate to the backend directory:** cd Backend
2. **Install backend dependencies:** npm install
3. Create a `.env` file in the Backend directory and add the following environment variables:

   PORT=5000
   MONGO_URI=your_mongodb_connection_string
   JWT_SECRET=your_jwt_secret_key

c. Set Up the Frontend

1. **Navigate to the frontend directory**: cd ../Frontend
2. **Install frontend dependencies:** npm install

d. Run the Application

1. **Start the backend server:**

   cd ../Backend

   npm start

2. **In a new terminal window, start the frontend development server:**

   cd ../Frontend

   npm start

3. **Open your browser and navigate to:** http://localhost:3000

5. Folder Structure

Client (Frontend - React Application)

The frontend directory is structured as follows:

- **public/**: Contains static assets (e.g., index.html, images, etc.)
- **src/**: Contains the source code for the React application.
    - **api/**: Functions to handle API requests to the backend.
    - **components/**: Reusable React components used throughout the application.
    - **context/**: Context API files for managing global state.
    - **img/**: Images and assets used in the application.
    - **styles/**: Styled-components and global styles for consistent theming.
    - **utils/**: Utility functions and custom hooks for various operations.
    - **App.js**: The root component that sets up routing and layout.
    - **index.js**: The entry point of the React application.

Server (Backend - Node.js Application)

The backend directory is structured as follows:

- **controllers/**: Contains the logic for handling requests and responses.
- **db/**: Contains database connection setup and configuration files.
- **middleware/**: Custom middleware functions for authentication and error handling.
- **models/**: Defines the Mongoose schemas and models for MongoDB collections.
- **routes/**: Defines the API endpoints and associates them with controller functions.
- **node_modules/**: Directory for all installed backend dependencies.
- **.env**: Environment variables for sensitive information like database URI and JWT secret.
- **.gitignore**: Git ignore file to exclude unnecessary files from version control.
- **app.js**: The main entry point of the backend application, setting up the Express server.
- **package-lock.json**: Lock file for backend dependencies.
- **package.json**: Backend dependencies and scripts configuration.

6. Running the Application

Starting the Backend

1. Navigate to the backend directory:

   cd CodeBase/Backend

2. Install backend dependencies:

   npm install

3. Start the backend server:

   npm start

Starting the Frontend

1. Open a new terminal window and navigate to the frontend directory:

   cd CodeBase/Frontend

2. Install frontend dependencies:

   npm install

3. Start the frontend development server:

   npm start

By executing these commands in their respective directories, both the backend and frontend servers will run locally. This allows for efficient development and testing of the application within a local environment.

7. API Documentation

All backend APIs are protected by **JWT-based authentication** and require a valid token in the request headers.

User Authentication APIs

POST /api/users/register

Registers a new user

- Body Parameters:

  {

  "name": "Ram",

  "email": "Ram@example.com",

  "password": "yourPassword"

  }

- Response (201 Created):
  {
  "message": "User registered successfully",
  "token": "<JWT_TOKEN>"
  }

POST /api/users/login

Authenticates user and returns JWT token

- Body Parameters:

  {

  "email": "john@example.com",

  "password": "yourPassword"

  }

- **Response (200 OK):**
  {
  "message": "Login successful",
  "token": "<JWT_TOKEN>"
  }

Income APIs

POST /api/income

Adds a new income entry

- Headers: Authorization: Bearer <JWT_TOKEN>
- Body Parameters**:**

  {

  "title": "Salary",

  "amount": 10000,

  "category": "Job",

  "description": "Monthly Salary",

  "date": "2025-04-01"

  }

- Response (201 Created):
  {
  "message": "Income added successfully",
  "income": { ... }
  }

Expense APIs

POST /api/expense

Adds a new expense entry

- Headers: Authorization: Bearer <JWT_TOKEN>
- Body Parameters:

  {

"title": "Groceries",

"amount": 1200,

"category": "Food",

"description": "Weekly groceries",

"date": "2025-04-02"

}

- **Response (201 Created):**
  {
  "message": "Expense added successfully",
  "expense": { ... }
  }

GET /api/expense

Fetches all expense entries

- Headers: Authorization: Bearer <JWT_TOKEN>
- Response (200 OK):

{

"expenses": [

{

"_id": "...",

"title": "Groceries",

...

}

]

}

DELETE /api/expense/:id

Deletes an expense entry

- Headers: Authorization: Bearer <JWT_TOKEN>
- Response (200 OK):

```
{
  "message": "Expense deleted successfully"
}
```

8. Authentication

The **SpendSmart** application employs JSON Web Tokens (JWT) for stateless authentication and authorization. This approach ensures secure access to protected routes without maintaining server-side sessions.

User Registration & Login

- Registration (POST /api/users/register): Users provide their name, email, and password. The password is securely hashed using *bcrypt* before being stored in MongoDB.
- Login (POST /api/users/login): Upon providing valid credentials, the server verifies the password and issues a JWT. This token encapsulates the user's ID and is signed with a secret key.

Token Structure

A JWT comprises three parts:

1. Header: Specifies the token type and signing algorithm.
2. Payload: Contains user-specific data (e.g., user ID).
3. Signature: Ensures the token's integrity and authenticity.

The token format is:

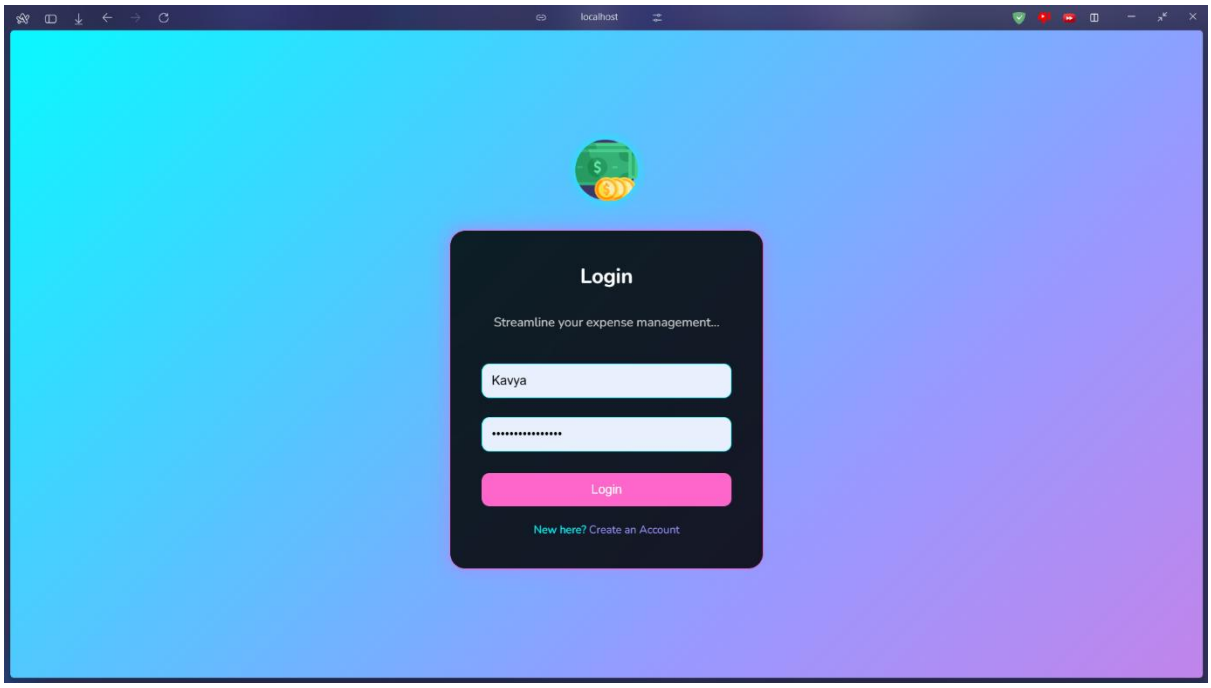
<header>.<payload>.<signature>

Token Storage & Usage

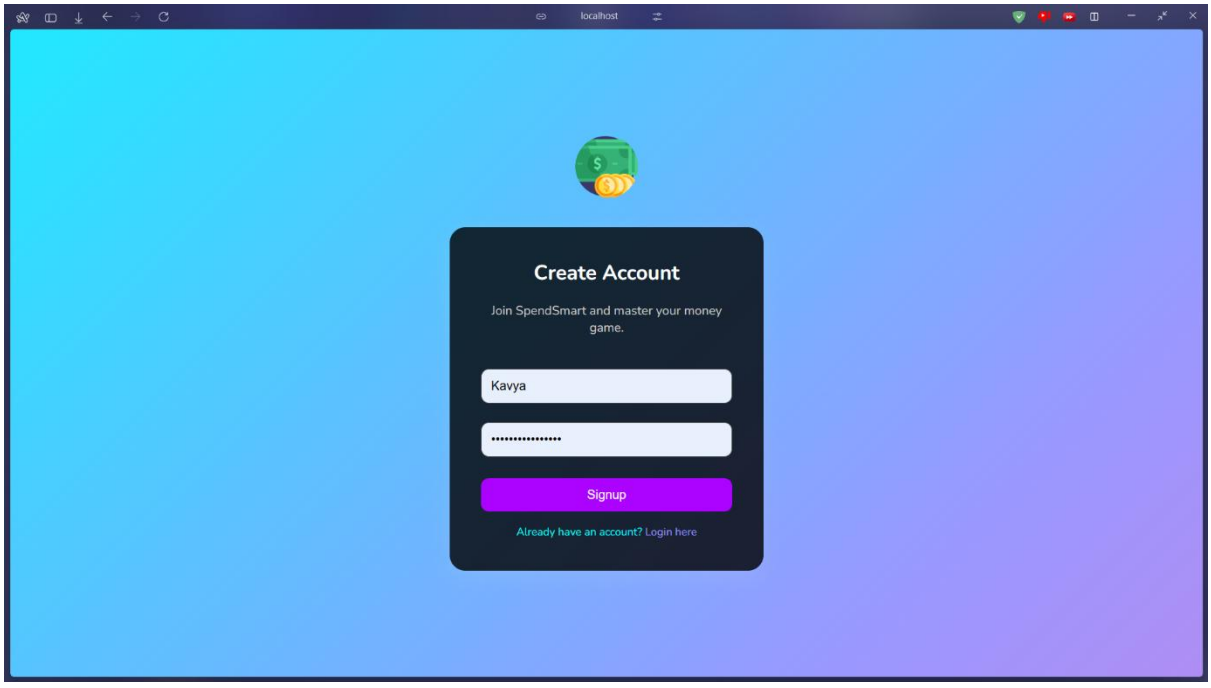
Client-Side Storage: The JWT is stored on the client side, typically in localStorage or session Storage.

Authorization Header: For subsequent requests to protected routes, the client includes the token in the Authorization header

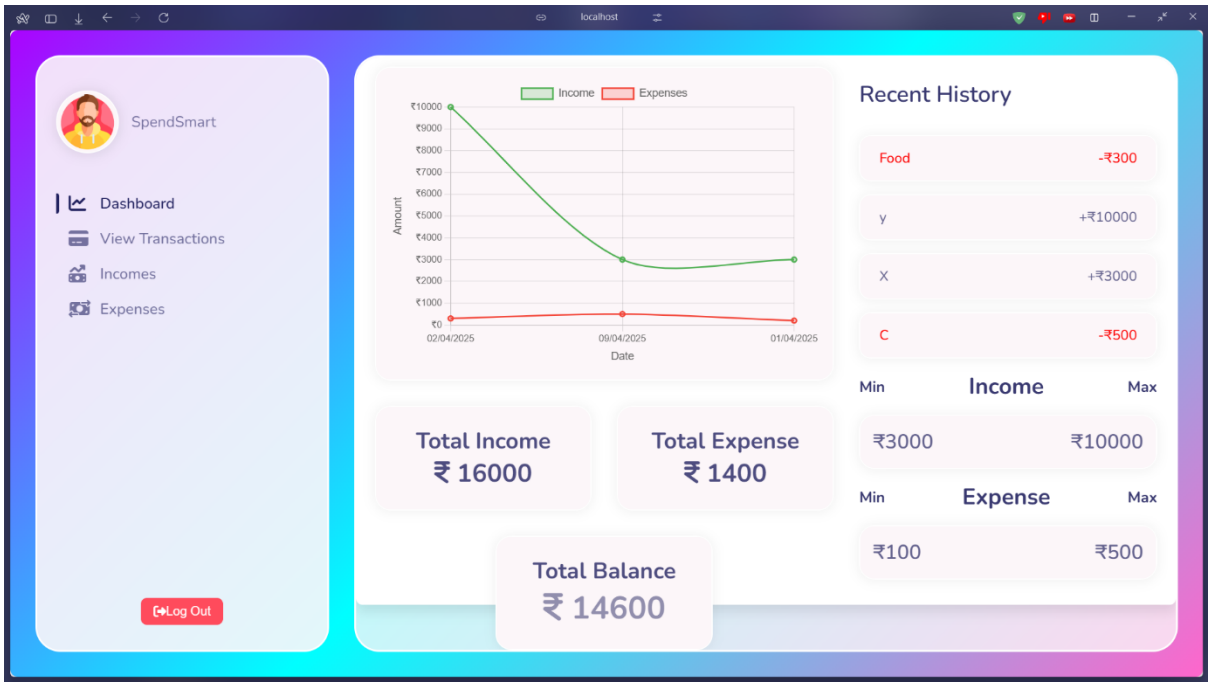
9. User Interface



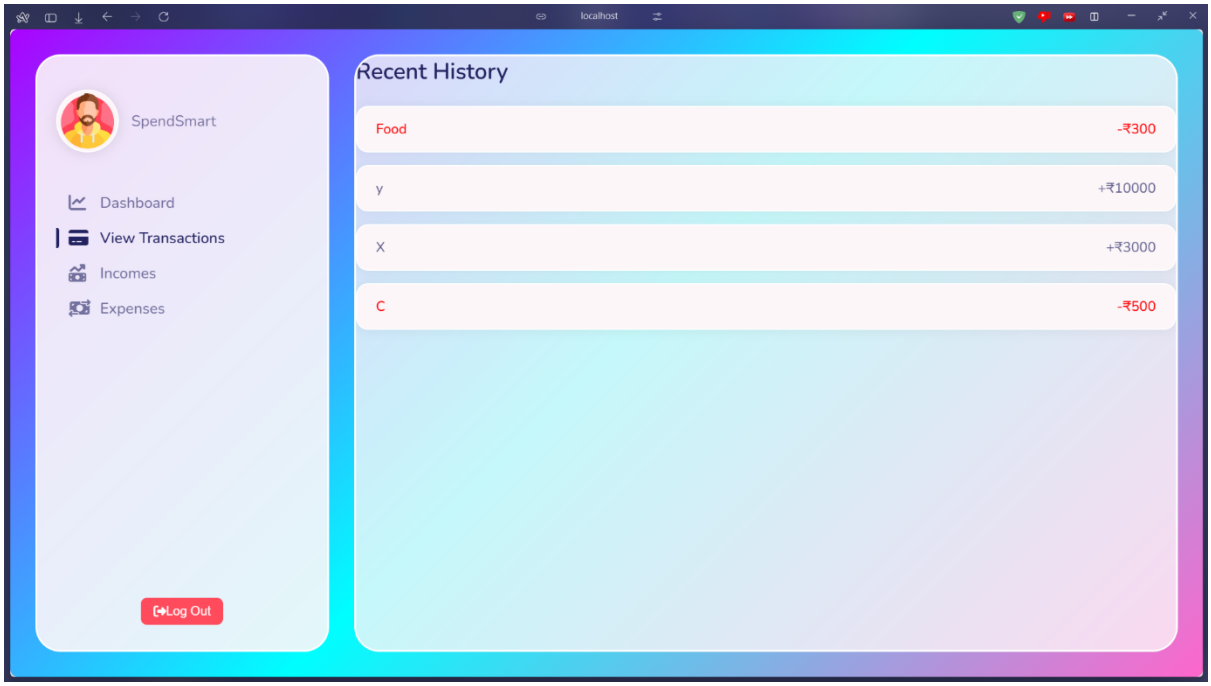
Login Page



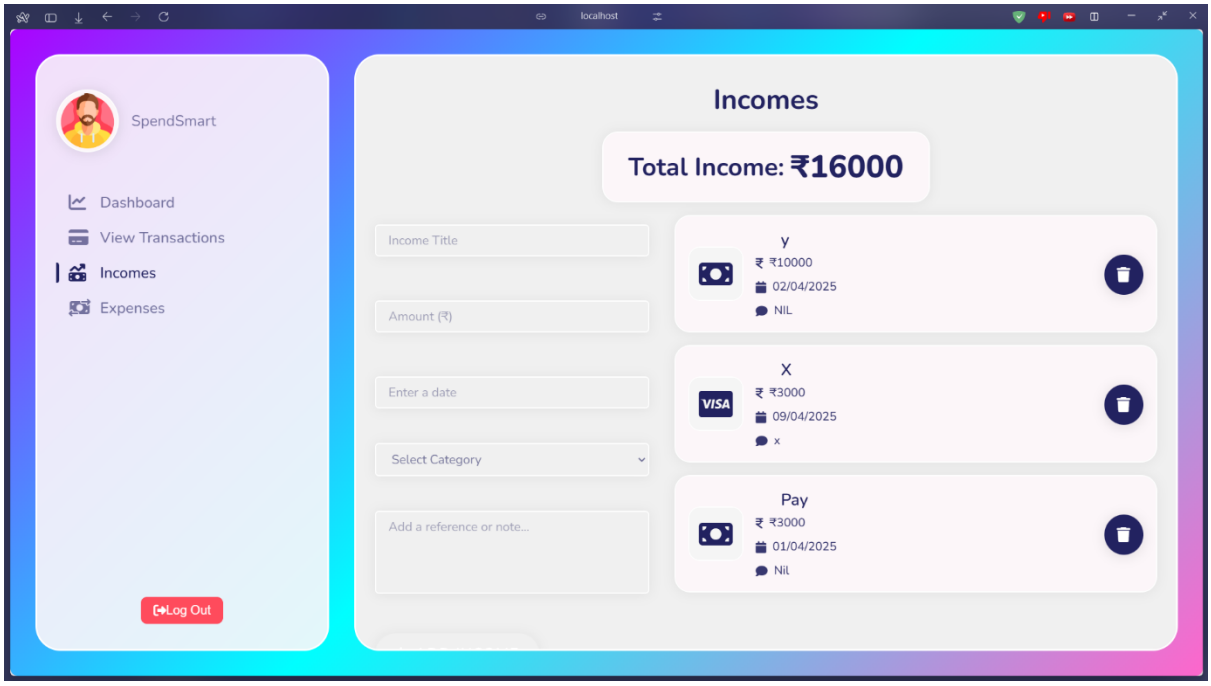
Registration Page



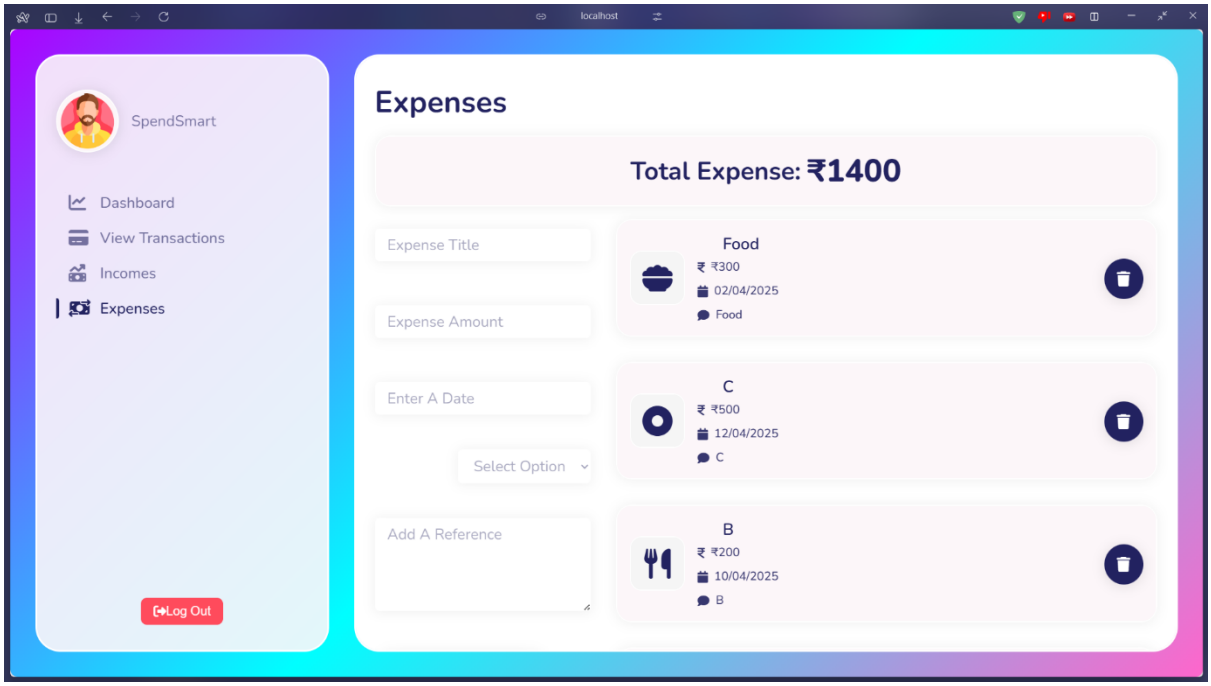
Dashboard Page



Transaction History Page



Incomes Page



Expenses Page

10. Testing

Testing Scope:

Features and Functionalities to be Tested:

- User Registration and Authentication
- Expense Entry and Management
- Income Tracking
- Dashboard Visualization

User Stories or Requirements to be Tested:

- As a user, I want to register and log in securely.
- As a user, I want to add, edit, and delete expenses.
- As a user, I want to view my income and expenses on a dashboard.
- As a user, I want to generate reports of my financial activities.

Test Case ID	Test Scenario	Test Steps	Expected Result	Actual Result	Pass/Fail
TC-001	User Registration	1.Navigate to the registration page. 2. Enter valid user details. 3. Submit the form.	User account is created, and a confirmation message is displayed.	As Expected	Pass
TC-002	User Login	Navigate to the login page. 2. Enter valid credentials. 3. Click on the login button	User is logged in and redirected to the dashboard.	As Expected	Pass

TC-003	Add Expense	Navigate to the expense page. 2. Click on 'Add Expense'. 3. Enter expense details. 4. Save the expense.	Expense is added and displayed in the expense list.	As Expected	Pass
TC-004	Delete Expense	Navigate to the expense list. 2. Select an expense to delete. 3. Confirm deletion.	Expense is removed from the list.	As Expected	Pass
TC-005	View Dashboard	Log in to the application. 2. Navigate to the dashboard.	Dashboard displays a summary of income and expenses with visual charts.	As Expected	Pass

Bug Tracking:

Bug ID	Bug Description	Steps To Reproduce	Severity	Status	Additional Feedback
BG-001	Expense not saving correctly	Navigate to 'Add Expense'. 2. Enter details. 3. Click 'Save'.	High	Resolved	Issue was due to missing field validation.
BG-002	Dashboard not loading	Log in to the application. 2. Navigate to the dashboard. 3. Observe that the dashboard does not load.	Medium	In Progress	Investigating potential API issues.

11. Known Issues

1. Error Handling Consistency

- Some API endpoints may lack comprehensive error handling, potentially leading to unhandled exceptions or unclear error messages.

2. Input Validation

- Certain forms might not enforce strict validation rules, allowing for the submission of incomplete or improperly formatted data.

3. Responsive Design

- The user interface may not be fully optimized for all screen sizes, which could affect usability on smaller devices.

4. Token Expiry Management

- There might be insufficient handling of expired JWT tokens, possibly resulting in unexpected user logouts or access issues.

12. Screenshots or Demo

Demo Video:

https://drive.google.com/file/d/107NMPkReSd0NoZ7AbRu0fzsbTQ9Ydmdu/view?usp=sharing

13. Future Enhancements

The **SpendSmart** project has established a solid foundation in personal finance management. To enhance its capabilities and user experience, the following future developments are proposed:

1. **Integration with Financial Institutions:**
    - Automated Data Synchronization: Establish secure connections with banks and financial entities to automatically import and categorize transactions, reducing manual input and improving accuracy.
2. **Artificial Intelligence (AI) and Machine Learning (ML) Enhancements:**
    - Predictive Analytics: Implement AI algorithms to analyze spending patterns and forecast future expenses, aiding users in proactive financial planning.
    - Smart Expense Categorization: Utilize ML models to automatically classify expenses based on historical data, enhancing user convenience.
3. **Mobile Application Development**:
    - Cross-Platform Support: Develop native mobile applications for iOS and Android to provide users with on-the-go access to their financial data.
    - Offline Functionality: Enable users to record expenses without an internet connection, with data synchronization upon reconnection.
4. **Multi-Currency and Localization Support**:
    - Currency Conversion: Allow users to manage expenses in multiple currencies with real-time exchange rate updates.
    - Language Localization: Offer the application in various languages to cater to a diverse user base.
5. **Enhanced Data Visualization and Reporting:**
    - Customizable Dashboards: Provide users with the ability to tailor their dashboards to highlight preferred financial metrics.
    - Advanced Reporting Tools: Introduce detailed reports with insights into spending habits, savings trends, and budget adherence.
6. **Integration with Emerging Financial Technologies**:
    - Cryptocurrency Tracking: Incorporate features to monitor and manage cryptocurrency assets alongside traditional finances.

- Open Banking APIs: Leverage open banking standards to facilitate seamless data sharing and enhanced financial services.

7. **Gamification Elements:**

   - Financial Goals and Rewards: Introduce goal-setting features with rewards for achieving savings milestones to motivate users.
   - Spending Challenges: Create challenges that encourage users to limit spending in certain categories, promoting better financial habits.

Implementing these advancements will position **SpendSmart** at the forefront of personal finance management solutions, offering users a comprehensive, intelligent, and user-centric platform.