# EXPENSE TRACKER APP: DESIGN AND IMPLEMENTATION

## Introduction and Executive Summary

This document details the end-to-end design, development, and deployment of a comprehensive Expense Tracker Application, built using a full-stack technology approach. The goal of this application is to enable users to efficiently manage their personal or household finances by recording day-to-day expenses and incomes, setting budgets, and reviewing categorized spending data through interactive dashboards. By integrating both frontend and backend technologies with a secure database, the system empowers users with real-time financial tracking and analytical tools to make informed decisions.

The project incorporates Spring Boot as the backend framework, MySQL for persistent data storage, and a responsive frontend developed using HTML, CSS, and JavaScript, enhanced with visualization libraries like Chart.js. Through RESTful APIs, the application enables CRUD operations for key entities like Users, Expenses, Categories, Budgets, and Reminders. This project also emphasizes clean UI design, modular coding structure, scalability, and usability, offering a seamless experience across devices. By following best practices in software engineering, this application serves as a model implementation of a modern financial management system and showcases the practical application of full-stack development skills.

## Target Audience and Value Proposition

The target audience for the Expense Tracker Application includes a wide range of individuals and user groups who wish to gain better control over their personal finances. The primary users are **students, working professionals, and families** who are looking for an easy and intuitive way to log, monitor, and manage their expenses and incomes. These users typically need a system that helps track their day-to-day transactions, categorize them, and view summarized reports or visual insights to better understand their spending patterns. Additionally, individuals trying to develop saving habits or stick to a monthly budget will find this application extremely valuable.

The value proposition of the Expense Tracker lies in its ability to provide **a centralized, user-friendly platform** where users can not only log transactions in real-time but also set budgets, receive reminders for bills, and view graphical insights into their financial behavior. It simplifies the process of financial

tracking with easy navigation, secure data handling, and smart categorization of expenses (such as rent, food, utilities, etc.). The system encourages mindful spending and enables users to take proactive steps toward their financial goals. With features like edit/delete options, summary charts, and responsive design, the app delivers both **functionality and convenience**, making it a powerful personal finance tool for everyday use.

## Technology Overview

The Expense Tracker App is built using a modern full-stack architecture that integrates robust and efficient technologies to deliver a scalable, maintainable, and user-friendly financial management solution. The platform has been carefully crafted using industry-standard tools and frameworks to ensure a seamless interaction between the frontend, backend, and database layers. The design emphasizes performance, security, and responsiveness, making it suitable for a variety of devices and user environments.

- **Front-End**: The user interface is developed using **HTML**, **CSS**, and **JavaScript**, ensuring a lightweight and responsive design. Custom styling and layout are implemented to provide a clean and modern user experience. Bootstrap and modular CSS are optionally used for consistency and ease of layout design. The frontend includes multiple views such as *Add Expense*, *View Expenses*, and interactive charts for financial summaries.
- **Back-End**: The backend is implemented using **Java** with the **Spring Boot** framework. This enables efficient development of RESTful APIs that handle business logic, database communication, and HTTP requests. Spring Boot provides seamless integration with various components and supports dependency injection, robust exception handling, and configuration management.
- **Database**: The application uses **MySQL** as its relational database system. All data including user information, transaction details, categories, budgets, and reminders are stored in structured tables with relationships established using foreign keys. The schema is designed for normalization and optimized for fast queries and integrity.

- **Build and Dependency Management**: **Apache Maven** is used as the build automation and dependency management tool, allowing easy management of external libraries, compiling source code, and packaging the application into executable artifacts.
- **Deployment and Execution**: The application is developed and tested locally using **Eclipse IDE** and optionally **Spring Tool Suite (STS)**. The backend runs on a local server (default port: 8086), while the frontend is served via **Live Server** in Visual Studio Code or other local HTTP servers. This setup ensures modular development and easy debugging during development.
- **API Integration and Cross-Origin**: To allow the frontend and backend to communicate seamlessly, **CORS (Cross-Origin Resource Sharing)** is enabled in the backend controller layer, allowing secure API calls between localhost:5500 (frontend) and localhost:8082 (backend).

## Document Structure

This documentation provides a comprehensive and structured overview of the design, development, and deployment process of the **Expense Tracker App**. The content is organized in the following sections to ensure clarity, completeness, and alignment with real-world full-stack development practices:

- **Introduction and Executive Summary**
  Provides a high-level overview of the Expense Tracker application, its primary objectives, the problem it aims to solve, its value proposition, and the target audience. This section also summarizes the technology stack and development environment used.
- **System Architecture and Design**
  Describes the layered architecture of the application, including the front-end, back-end, and database layers. This section explains how these layers interact through RESTful APIs and discusses the overall flow of data between client and server.
- **Front-End Development**
  Details the development of the user interface using HTML, CSS, and JavaScript. It includes a breakdown of components such as the *Add Expense* and *View Expenses* modules, styling choices, form validation, and how the frontend communicates with the backend via API calls.
- **Back-End Development**
  Explores the implementation of the server-side logic using Java and Spring Boot. It covers the

creation of REST controllers, service layers, repository layers, and exception handling. API endpoints for CRUD operations on expenses, users, categories, budgets, and reminders are discussed.

- **Database Design and Implementation**
  Explains the relational schema created using MySQL. It includes detailed entity-relationship modeling, normalization principles, and foreign key relationships. This section also provides sample queries and explains how Spring Data JPA is used to interact with the database.
- **Security Considerations**
  Discusses measures taken to ensure secure API communication and data handling, such as enabling Cross-Origin Resource Sharing (CORS), securing HTTP endpoints, and ensuring input validation. Suggestions for implementing user authentication and role-based access control are also included.
- **Testing and Quality Assurance**
  Describes the testing methodologies used during development, including unit testing of service layers, manual testing of REST endpoints using Postman, and browser-based testing of the frontend using Live Server. Any automated test plans or validation scenarios are also covered.
- **Deployment and Infrastructure**
  Outlines the local development setup, including tools like Eclipse IDE, Spring Tool Suite, MySQL Workbench, and Live Server in VS Code. This section also provides instructions for running the backend and frontend, as well as potential cloud deployment considerations using platforms like Heroku or AWS.
- **Future Enhancements and Scalability**
  Explores ideas for extending the app's functionality, such as adding authentication, generating PDF reports, integrating third-party financial APIs, mobile app development, and deploying to cloud platforms. Strategies for scaling the application to handle larger datasets and concurrent users are discussed.
- **Conclusion**
  Summarizes the overall project journey, challenges faced during development, lessons learned, and reflections on applying full-stack development skills. It concludes with the project’s current status and readiness for deployment or further enhancements.

# Project Vision, Goals, and Problem Statement

## Vision

The vision behind the **Expense Tracker App** is to empower individuals to take full control of their personal finances through a simple, intuitive, and powerful digital platform. In a world where financial literacy and planning are critical, this application aims to bridge the gap between daily income and expenditure tracking and long-term financial well-being. The platform is designed for all kinds of users—students, working professionals, families—enabling them to easily log, visualize, and manage their expenses without needing advanced financial knowledge.

By providing actionable insights, personalized reminders, and budgeting tools, the application seeks to create a habit-forming environment that promotes mindful spending. The ultimate vision is to help users reduce financial stress, build better savings habits, and make informed financial decisions, all within a secure and user-friendly digital ecosystem that works seamlessly across devices.

## Problem Statement

Managing personal expenses often becomes overwhelming due to the lack of a systematic approach, absence of real-time insights, and difficulty in accessing consolidated reports. People commonly rely on paper-based records, scattered Excel sheets, or basic note apps, which are ineffective in tracking and categorizing expenses or visualizing financial trends. This leads to budget overruns, missed due dates, and limited understanding of where money is going—ultimately hampering financial planning.

This project addresses several key problems individuals face in expense management:

- **Lack of Organization**: Users have no centralized system to track all types of expenses and income.
- **Poor Visibility**: Difficulty in generating meaningful reports and understanding patterns in spending.
- **No Reminders**: Missed payment dates due to lack of automated alerts.

- **Limited Access**: Many tools are too complex, expensive, or not tailored for daily personal use. The **Expense Tracker App** solves these challenges through a feature-rich yet simple platform that allows users to easily record, edit, delete, and visualize financial activities.

**SMART Goals**

To effectively realize the project vision and address the identified problems, the following **SMART (Specific, Measurable, Achievable, Relevant, Time-bound)** goals have been established:

- **Specific**: Build a secure, responsive expense tracking system with modules for adding, editing, deleting, and analyzing income and expenses for individual users.
- **Measurable**: Support up to 1,000 users with smooth backend performance, capable of storing 10,000+ transactions and rendering visualizations in under 2 seconds.
- **Achievable**: Implement backend security using Spring Boot and database consistency using MySQL; ensure REST APIs are tested and function correctly across endpoints.
- **Relevant**: Provide useful financial insights, monthly summaries, and personalized budget alerts that help users track and manage their money efficiently.
- **Time-bound**: Complete core development within **3 months**, launch an MVP (Minimum Viable Product) in the 4th month, and begin collecting user feedback for further improvements.

## Project Scope Definition

The **scope of the Expense Tracker App** project includes the design, development, and deployment of a full-stack web-based personal finance management application. The application enables users to track income and expenses, categorize financial transactions, set reminders, and view summaries or insights into their spending behavior. It includes user authentication, CRUD operations for various modules (expenses, categories, budgets, reminders), and data visualization through charts. The front-end interface is designed for usability and responsiveness, while the backend ensures secure storage, API communication, and business logic.

Key functionalities included within the scope of this project are:

- **User Authentication**: Registration, login, and session handling for secure access.
- **Expense Management**: Adding, editing, deleting, and listing of expenses with date, title, and amount fields.
- **Category Handling**: Categorizing expenses for better organization (e.g., Food, Travel, Rent).
- **Budget Setting**: Setting monthly budget limits and comparing against actual expenses.
- **Reminders**: Scheduling reminders for recurring expenses like rent, bills, or subscriptions.
- **Data Visualization**: Generating pie and bar charts to show expense distribution and trends.
- **REST API Integration**: Use of Spring Boot REST APIs to handle all backend operations securely.
- **MySQL Database**: Structured data storage with entity relationships.

## Technology Stack Selection and Justification

The Expense Tracker App uses a modern and efficient full-stack technology architecture tailored for simplicity, scalability, and performance. Careful consideration was given to ensure ease of development, seamless integration between front-end and back-end, and secure, reliable operations.

**Front-End: HTML, CSS, and JavaScript**

The front-end of the application is built using HTML, CSS, and vJavaScript. This decision was made to ensure that the user interface remains lightweight, fast-loading, and easy to understand for both developers and users. HTML provides the structure, CSS is used for styling with a focus on a modern, professional UI, and JavaScript is responsible for DOM manipulation, form validations, and AJAX requests to the backend.

**Justification**:

- **Simple Yet Effective**: Basic web technologies are universally supported, easy to debug, and suitable for MVP development.
- **Responsiveness**: Modern CSS (Flexbox, Grid) allows the application to be responsive across devices.

- **AJAX Integration**: Enables dynamic behavior such as adding and deleting expenses without page reload.
- **Lightweight & Fast**: Avoids overhead of heavy frameworks, which is perfect for focused, single-purpose tools like an expense tracker.

**Back-End: Java with Spring Boot**

The backend is implemented using **Spring Boot**, a production-ready Java-based framework. It supports rapid development and is ideal for building stand-alone RESTful APIs. All application logic, routing, validation, security, and data processing are handled in the backend. It also integrates with MySQL for data persistence and includes support for JPA and Hibernate ORM for mapping between Java objects and relational tables.

**Justification**:

- **Robust & Scalable**: Spring Boot is known for stability, scalability, and large community support.
- **Fast Development**: Spring Boot's built-in features like auto-configuration, dependency injection, and REST support accelerate development.
- **Security**: Integrates easily with Spring Security and supports role-based access control if needed.
- **Maintainability**: Follows clean architecture (MVC) and separates concerns via packages (model, service, repository, controller).

**Database: MySQL**

The application uses **MySQL** as its primary data storage engine. It holds user information, transactions, budgets, reminders, and category data. A normalized schema with foreign key relationships ensures data integrity. Sample data is provided to demonstrate functionality, and SQL scripts are available for initial setup.

**Justification**:

- **Reliability**: MySQL is a mature, stable relational database widely used in production environments.
- **Structured Data**: Fits well with the tabular format of expenses and categories.
- **Support for Relationships**: Enforces data consistency via foreign key constraints and joins.
- **Integration**: Seamlessly integrates with Spring Boot using JPA and Hibernate.

**Other Supporting Tools and Services**

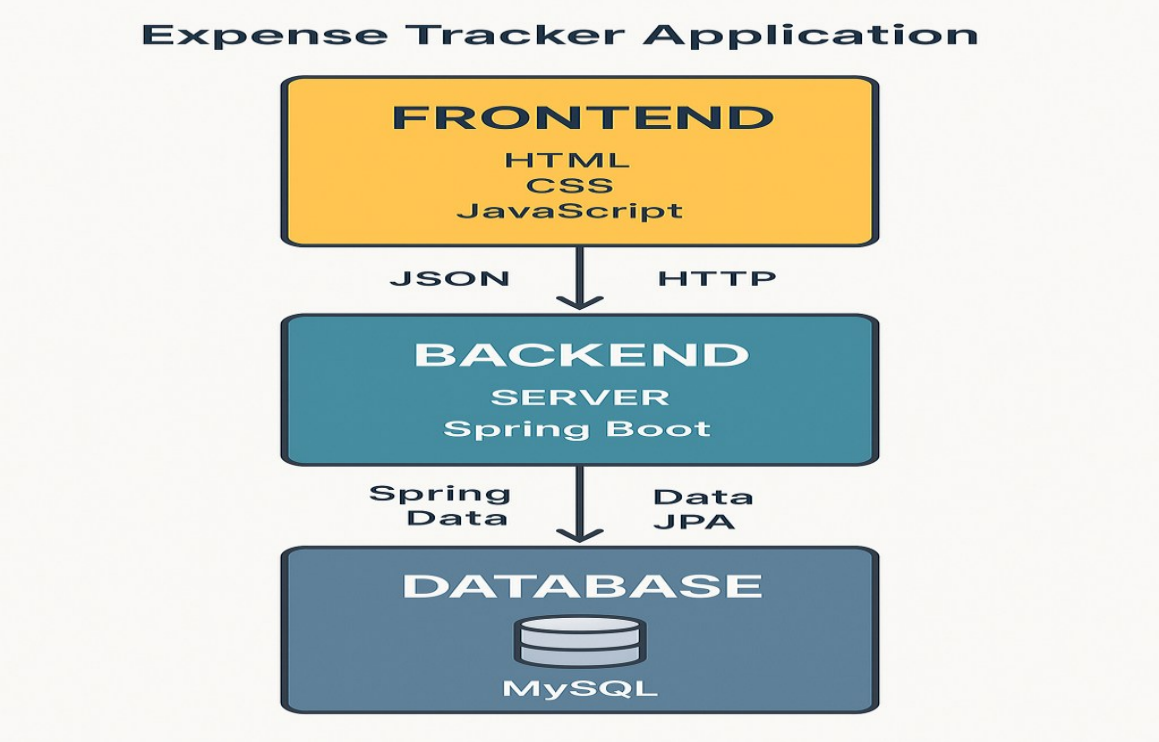
- **Apache Maven**: Used as the build tool and dependency manager, allowing easy inclusion of required libraries.
- **Spring DevTools**: Enhances development speed with hot reload and debugging support.
- **Live Server Extension** (VS Code): Used to serve frontend files and make live API calls during development.
- **Browser/Postman**: Helpful in testing and debugging REST APIs before full frontend integration.

## System Architecture and Design

The **Expense Tracker Application** follows a layered monolithic architecture, ideal for small to mid-scale personal finance tools that require simplicity, tight integration, and maintainability. Unlike a microservices architecture, where functionalities are distributed into independent services, the monolithic design consolidates all modules (user management, expense tracking, budget setting, reminders, etc.) into a single deployable unit. This ensures easier development and deployment without the complexities of inter-service communication, load balancing, or service discovery. Given the focused scope and predictable growth of this application, the monolithic approach is well-suited and cost-effective, while still allowing scalability through optimized code structure and modularization.

At a high level, the system is divided into three main layers: the **Frontend (Client)**, **Backend (Server)**, and **Database**. The **Frontend** is built using HTML, CSS, and JavaScript, providing a responsive and intuitive user interface for managing expenses. It communicates with the **Backend REST APIs** developed using Spring Boot over HTTP using JSON payloads. The backend encapsulates all business logic, service handling, request processing, and data validations. It communicates with the **MySQL**

**relational database** using Spring Data JPA for persistent data storage. This separation of concerns ensures that each layer can evolve independently, and the use of standard interfaces and protocols enables easy testing, debugging, and maintenance.




## Component Overview

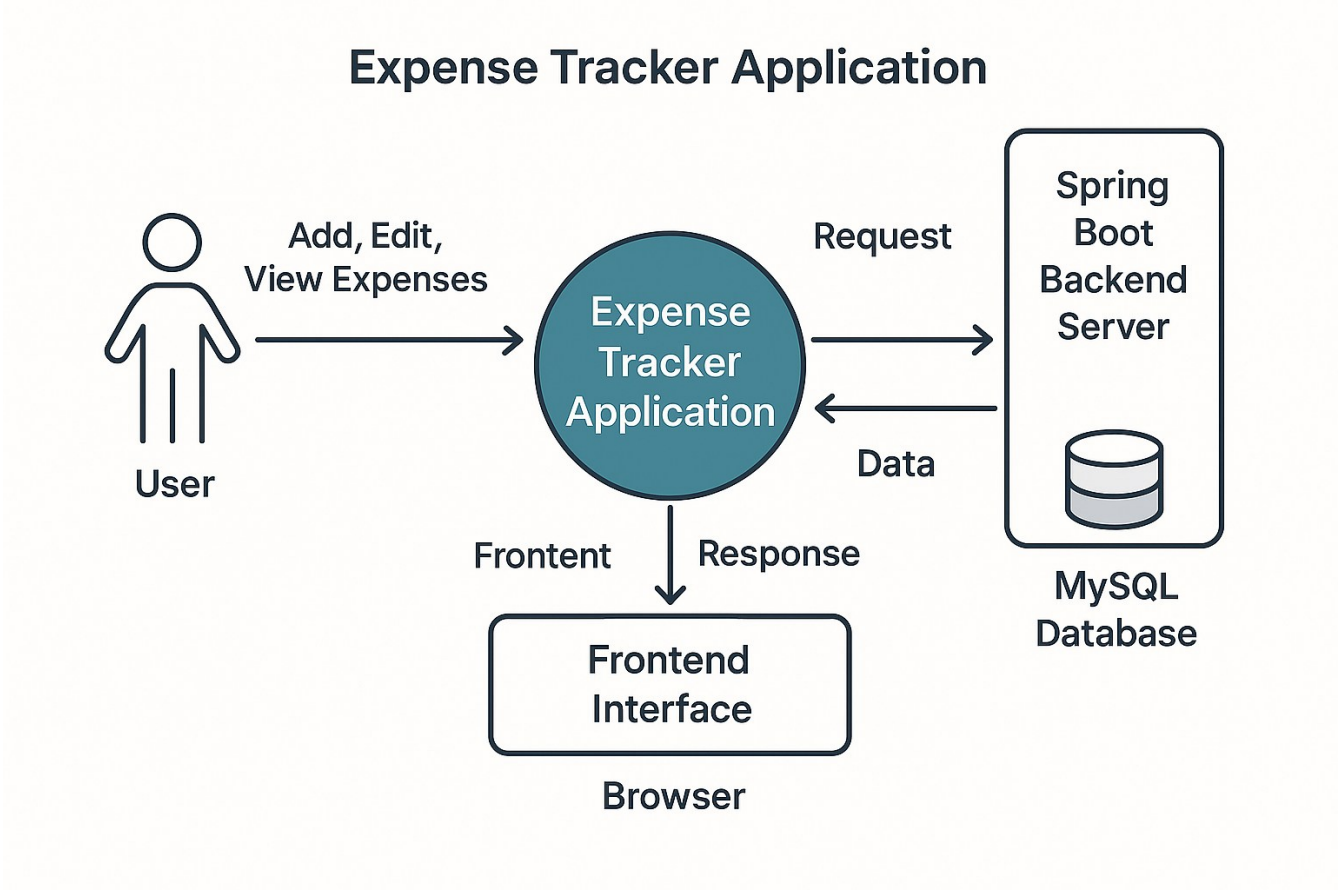
- **Client (Frontend):** The user interacts with the application via modern HTML pages (add-expense.html, view-expenses.html) styled with CSS and powered by JavaScript. The pages allow users to add, view, edit, and delete expenses and budgets in real-time using AJAX requests. All data input from users is handled by JavaScript, which communicates directly with the backend APIs to fetch and manipulate records.
- **Backend (Spring Boot Server):** The server acts as the central logic processor. It consists of various modules implemented using Java classes categorized into controller, service, repository, and model packages. Controllers expose RESTful endpoints for each feature such as /api/expenses, /api/budgets, /api/reminders, and so on. Services encapsulate the business logic, while Repositories interact with the database using JPA and Hibernate ORM.

- **Database (MySQL):** All persistent data including users, expenses, categories, budgets, and reminders are stored in a normalized relational structure. Each table is mapped to a corresponding Java entity using @Entity annotations. Foreign key relationships are used to ensure data consistency and integrity, such as linking expenses to categories or users.

## Data Flow Diagrams (DFD)

**Level 0**: Shows the high-level flow between user, frontend, backend, and database.

**Level 1**: Explains individual modules like how an expense is added, fetched, updated, and deleted.




## Database Schema and Design

**Overview**

In Expense Tracker App project, the primary focus was on designing and implementing a robust and scalable relational database using **MySQL**. This phase established the foundation for data persistence,

enabling the application to store and retrieve essential information such as user profiles, expenses, budgets, categories, and reminders. The database schema was carefully structured to support future scalability and ensure efficient querying, integrity, and maintainability.

The database design phase followed a logical and structured approach, beginning with requirement gathering, followed by Entity-Relationship (ER) modeling, normalization, and actual SQL implementation. Tools such as **MySQL Workbench** were used to visually design and execute the schema, ensuring the relationships and constraints were properly enforced.

**Entities and Relationships**

The Expense Tracker database consists of the following key entities:

1. **User**
    - Represents registered users of the application.
    - Attributes: user_id, name, email, password, created_at.
2. **Category**
    - Defines different types of expenses (e.g., Food, Rent, Travel).
    - Attributes: category_id, name, description.
3. **Expense**
    - Stores individual expense records for each user.
    - Attributes: expense_id, title, amount, date, user_id, category_id.
    - Relationships:
        - Many-to-One with User (each expense belongs to a user).
        - Many-to-One with Category (each expense belongs to a category).
4. **Budget**
    - Allows users to set monthly budgets by category.

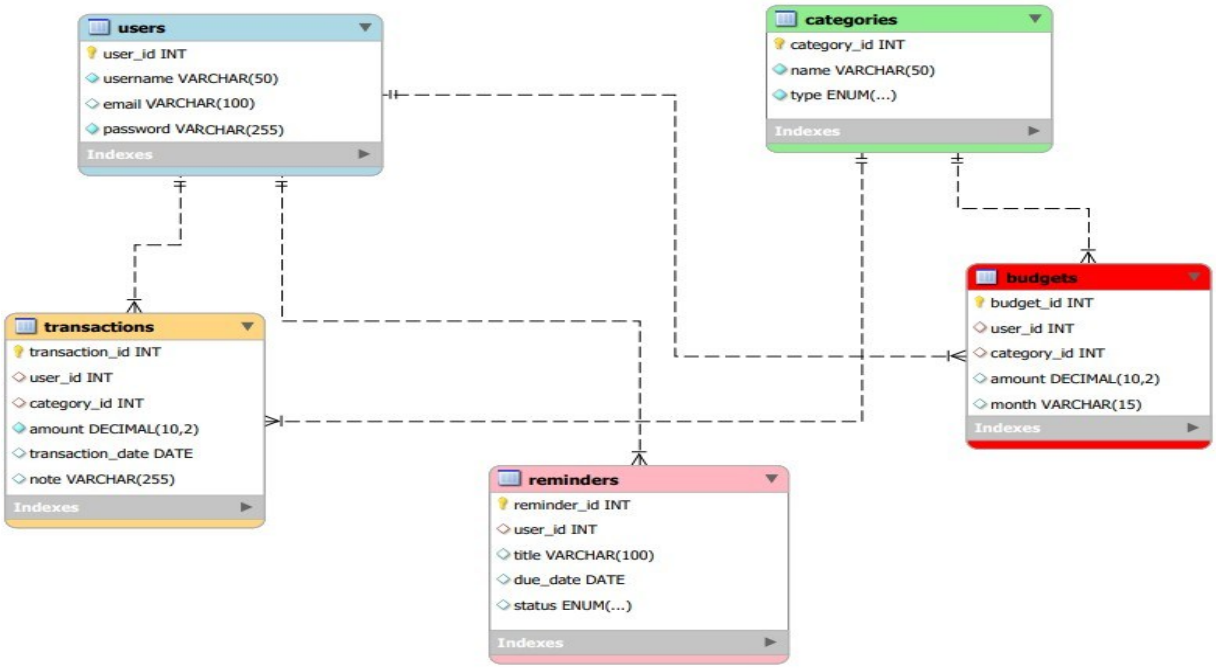
- Attributes: budget_id, user_id, category_id, amount, month.

5. **Reminder**
    - Stores user-defined reminders for bills or payments.
    - Attributes: reminder_id, user_id, message, reminder_date.

## EER Diagram

The Enhanced Entity-Relationship (EER) diagram was designed in MySQL Workbench and includes all entities and their foreign key relationships. Key relationships:

- **User ⇨ Expense** (1:N)
- **Category ⇨ Expense** (1:N)
- **User ⇨ Budget** (1:N)
- **Category ⇨ Budget** (1:N)
- **User ⇨ Reminder** (1:N)



This diagram visually validated our design and ensured all constraints (like ON DELETE CASCADE) were correctly applied.

**Normalization**

To maintain data integrity and reduce redundancy, the database schema was normalized to **Third Normal Form (3NF)**:

- Repetitive fields were extracted into separate tables (e.g., categories).
- All non-key attributes are fully functionally dependent on the primary key.
- Transitive dependencies were removed by introducing bridge tables where needed.

This ensured optimal storage efficiency and simplified data operations.

## SQL Implementation

After designing the schema, the database was created using SQL scripts.

Examples:

```
CREATE TABLE User (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100) UNIQUE,
    password VARCHAR(100),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Category (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    description TEXT
```

```
);
CREATE TABLE Expense (
    expense_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100),
    amount DECIMAL(10,2),
    date DATE,
    user_id INT,
    category_id INT,
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (category_id) REFERENCES Category(category_id)
);
```

Additional tables for Budget and Reminder were also created with proper keys and constraints.

**Data Population and Testing**

Once the schema was implemented, sample data was inserted into the tables to validate the relationships and test CRUD operations. SQL queries such as JOIN, GROUP BY, and ORDER BY were run to simulate reports like:

- Total expenses per month
- Budget vs. actual spending
- Upcoming reminders

## API Documentation and Authentication

**Overview of API Endpoints**

The Expense Tracker backend exposes several RESTful API endpoints for managing expenses. These endpoints are built using Spring Boot and follow REST principles to allow integration with various frontends (e.g., HTML, React, mobile apps).

**Example Endpoints:**

- GET /api/expenses – Retrieves all expenses.

- POST /api/expenses – Adds a new expense.
- PUT /api/expenses/{id} – Updates an existing expense.
- DELETE /api/expenses/{id} – Deletes an expense by ID.

**API Testing and Swagger Integration**

To facilitate testing and documentation, the project can integrate **Swagger UI** using SpringFox (or OpenAPI). This tool provides an interactive API explorer where developers can try requests, see schemas, and validate input/output formats.

**Authentication Mechanism**

Although current project operations are open (no login), future enhancement includes **JWT-based Authentication**:

- Upon login, users receive a signed JWT token.
- Protected endpoints will require this token for access.
- Token contains user role, ID, and expiry.

## Testing Strategy

**Overview of Testing Layers**

Testing ensures application reliability, maintainability, and correctness. This project adopts a multi-layered testing strategy:

- **Unit Testing** for individual methods/services.
- **Integration Testing** for interaction between components (e.g., controller + service).
- **End-to-End Testing** for simulating actual user operations.

**Manual Testing**

Manual test cases were written and executed using test scenarios:

- Adding a valid expense

- Editing and deleting expenses
- Checking UI responsiveness

  Each test case was documented with expected vs actual results.

**Automation Scope**

Although manual testing was the focus, automation can be incorporated using:

- **JUnit** and **Mockito** for backend
- **Selenium/Cypress** for frontend testing

## Deployment and Infrastructure

The Expense Tracker application is designed for easy deployment across development, staging, and production environments. The backend is built using Spring Boot and can be deployed on any cloud server supporting Java . It runs on port 8082 and connects to a MySQL database. The frontend consists of modular HTML, CSS, and JavaScript files that can be hosted using services like GitHub Pages, Netlify, or an Apache/Nginx web server.

Infrastructure-wise, the backend server is configured to connect with MySQL using secure credentials stored in `application.properties`. Deployment is manual for now but can be enhanced using CI/CD pipelines like GitHub Actions or Jenkins. Docker can be incorporated to containerize the application, ensuring portability across different environments. Monitoring tools such as Prometheus or Spring Boot Actuator can be integrated for tracking application health, uptime, and metrics.

## Security Considerations

Security is an essential pillar of the Expense Tracker App. To protect sensitive user data, especially expense-related information, several security practices are implemented. HTTPS is enforced for all client-server communication to prevent data interception. Backend APIs are secured using CORS policies, JWT authentication, and proper role-based access where required. Input validation and sanitation techniques are applied to prevent SQL injection and XSS attacks.

Sensitive configuration like database passwords are never hardcoded and are externalized using environment variables or configuration files. All database operations are performed using prepared statements through Spring Data JPA, mitigating the risk of injection attacks. Future security enhancements include implementing multi-factor authentication (MFA), logging suspicious login attempts, and conducting periodic vulnerability assessments to identify and fix security loopholes proactively.

## Future Enhancements and Roadmap

The Expense Tracker App is a scalable and extensible financial management tool designed with long-term innovation in mind. Although the current system supports essential operations such as adding, viewing, editing, and deleting expenses, several valuable enhancements are planned for future versions to expand its functionality, improve usability, and enhance decision-making.

- **AI-Powered Spending Insights**

Future updates will include artificial intelligence and machine learning algorithms to analyze spending behavior. These smart insights will detect patterns, provide predictive analytics (e.g., forecast next month's expenses), and recommend budget adjustments. This will make the app more than just a tracker—it will become a financial advisor.

- **Budget Planning and Limit Alerts**
  A built-in budgeting system will allow users to set limits per category or per month. Real-time alerts will notify users when they approach or exceed those limits. Visual indicators like progress bars and warnings will ensure better control over finances.
- **Email Notifications and Reminders**

The app will support scheduled email reminders and alerts. For instance, users can receive:

- Weekly or monthly expense summaries
- Budget breach notifications
- Reminders to input daily expenses

This improves user engagement and ensures consistent tracking.

- **Native Mobile Application**
- A mobile version of the Expense Tracker App for **Android** and **iOS** will provide users the ability to manage finances on the go. The app will feature biometric login, push notifications, and offline support with cloud synchronization.
- **Export to PDF and CSV**

The system will offer functionality to export expense reports in PDF and CSV formats. This is especially useful for users who want to:

- Share reports with family or financial advisors
- Keep digital or printed backups
- Analyze data in spreadsheet software

- **Multi-User and Role-Based Access**

The application will allow multiple users with personal accounts, and roles such as **User** and **Admin**. Each user can manage their own data, and shared family accounts will support aggregated tracking. Authentication will be handled using secure JWT tokens.

- **Advanced Charts and Analytics**

Future versions will integrate with charting libraries like **Chart.js** to display:

- Monthly breakdowns
- Pie charts by category
- Year-over-year comparisons

These analytics will help users visualize their financial behavior and identify spending leaks.

- **Cloud Sync and Data Backup**Cloud storage options (such as Firebase or AWS) will be used to automatically back up data. Users can switch devices or recover data without losing records. Offline-first design will allow data entry without internet and sync it once connected.

- **Voice and OCR Input**

To simplify input, future versions may include:

- **Voice Commands** to add expenses (e.g., "Add ₹500 for groceries today").
- **OCR Scanning** to capture data from bills/receipts using the camera.

  This will enhance accessibility and save time.

- **Localization and Currency Support**

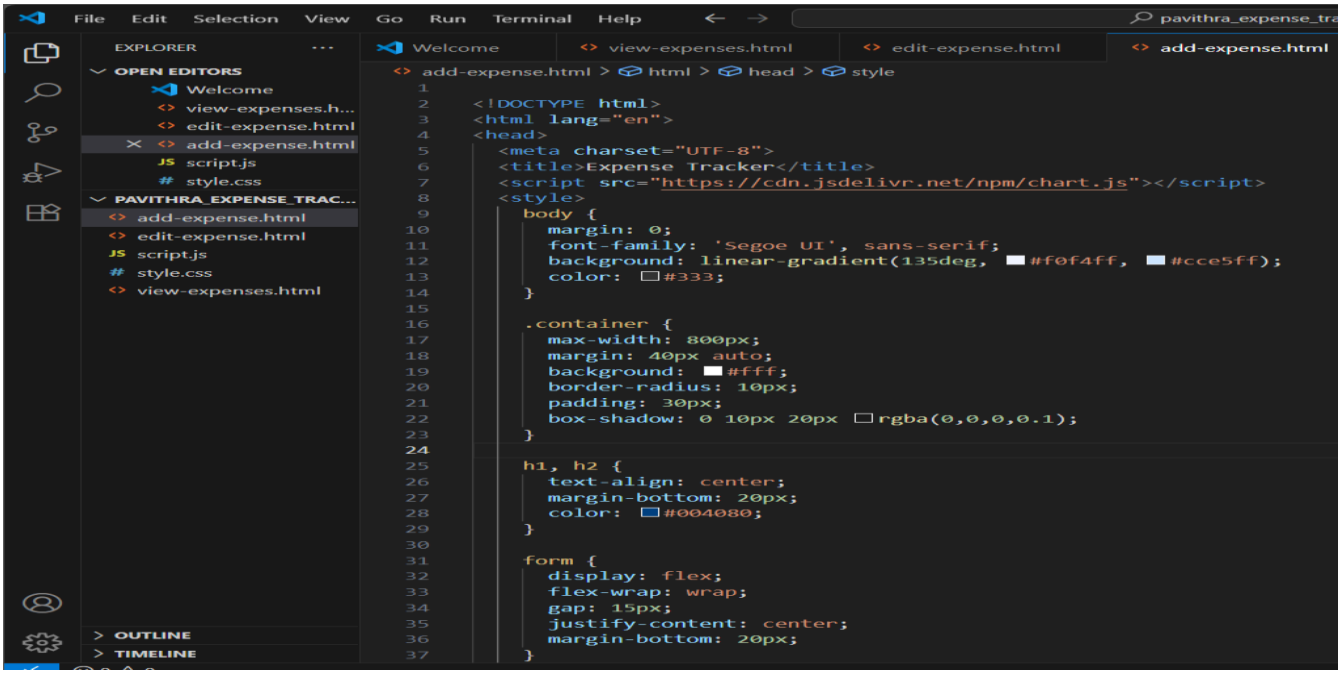
The app will support **multiple languages** (e.g., Telugu, Hindi) and regional currencies. Users will also be able to configure date formats and fiscal years based on location.

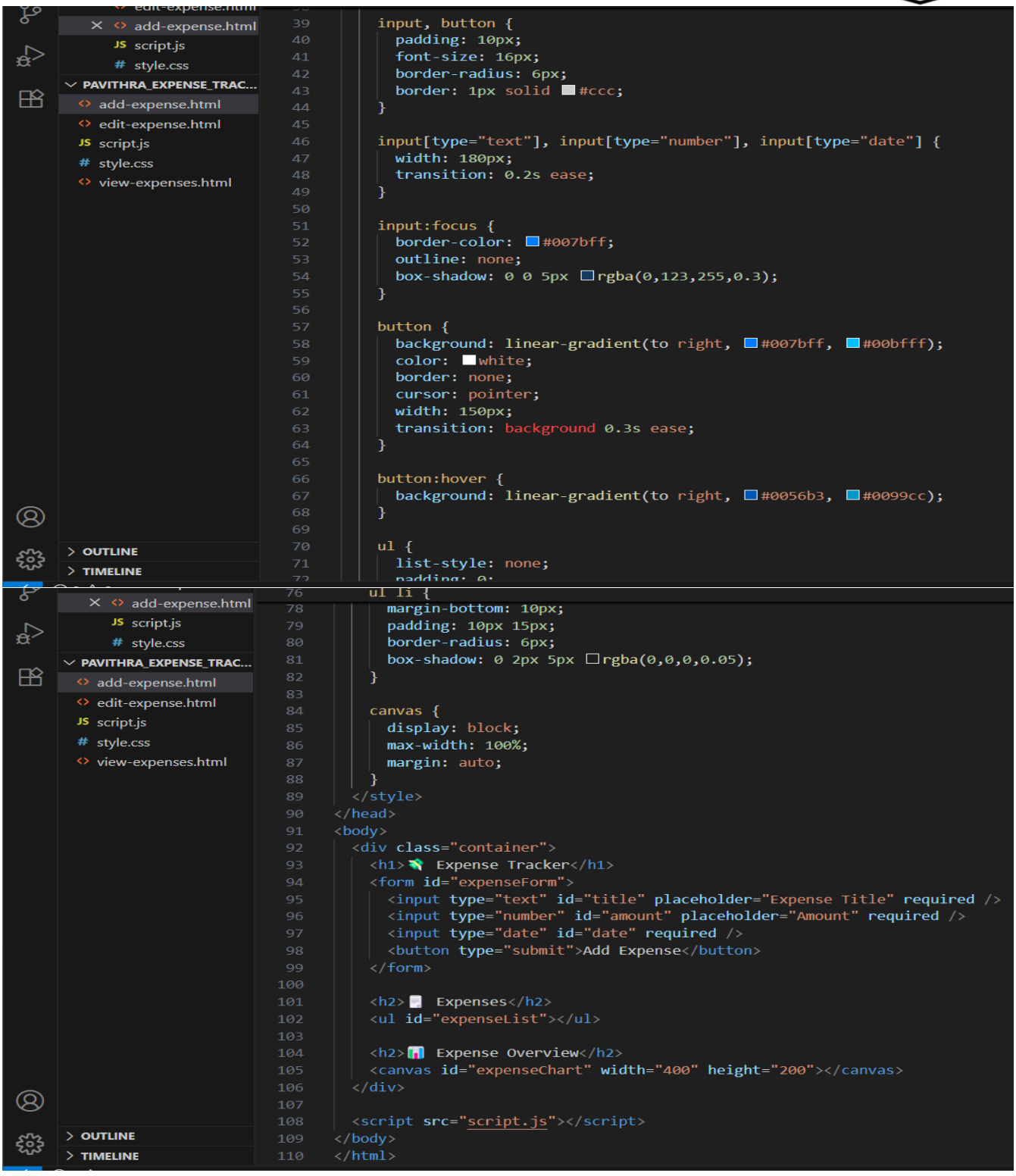
# APPENDIX

## Screenshots:

**Source Code:**

**add-expense.html**

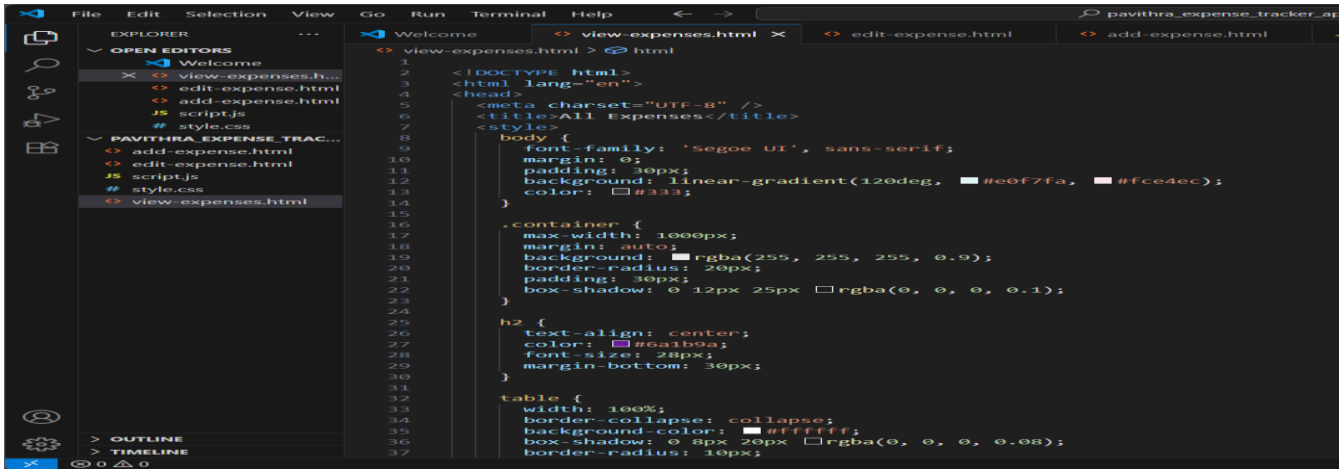



add-expense.html
script.js
style.css
PAVITHRA_EXPENSE_TRAC...
add-expense.html
edit-expense.html
script.js
style.css
view-expenses.html
OUTLINE
TIMELINE
input, button {
padding: 10px;
font-size: 16px;
border-radius: 6px;
border: 1px solid #ccc;
}
input[type="text"], input[type="number"], input[type="date"] {
width: 180px;
transition: 0.2s ease;
}
input:focus {
border-color: #007bff;
outline: none;
box-shadow: 0 0 5px rgba(0,123,255,0.3);
}
button {
background: linear-gradient(to right, #007bff, #00bfff);
color: white;
border: none;
cursor: pointer;
width: 150px;
transition: background 0.3s ease;
}
button:hover {
background: linear-gradient(to right, #0056b3, #0099cc);
}
ul {
list-style: none;
ul li {
margin-bottom: 10px;
padding: 10px 15px;
border-radius: 6px;
box-shadow: 0 2px 5px rgba(0,0,0,0.05);
}
canvas {
display: block;
max-width: 100%;
margin: auto;
}
</style>
</head>
<body>

<h1>💸 Expense Tracker</h1>
<form id="expenseForm">
<input type="text" id="title" placeholder="Expense Title" required />
<input type="number" id="amount" placeholder="Amount" required />
<input type="date" id="date" required />
<button type="submit">Add Expense</button>
</form>
<h2>📄 Expenses</h2>
<ul id="expenseList"></ul>
<h2>📊 Expense Overview</h2>
<canvas id="expenseChart" width="400" height="200"></canvas>

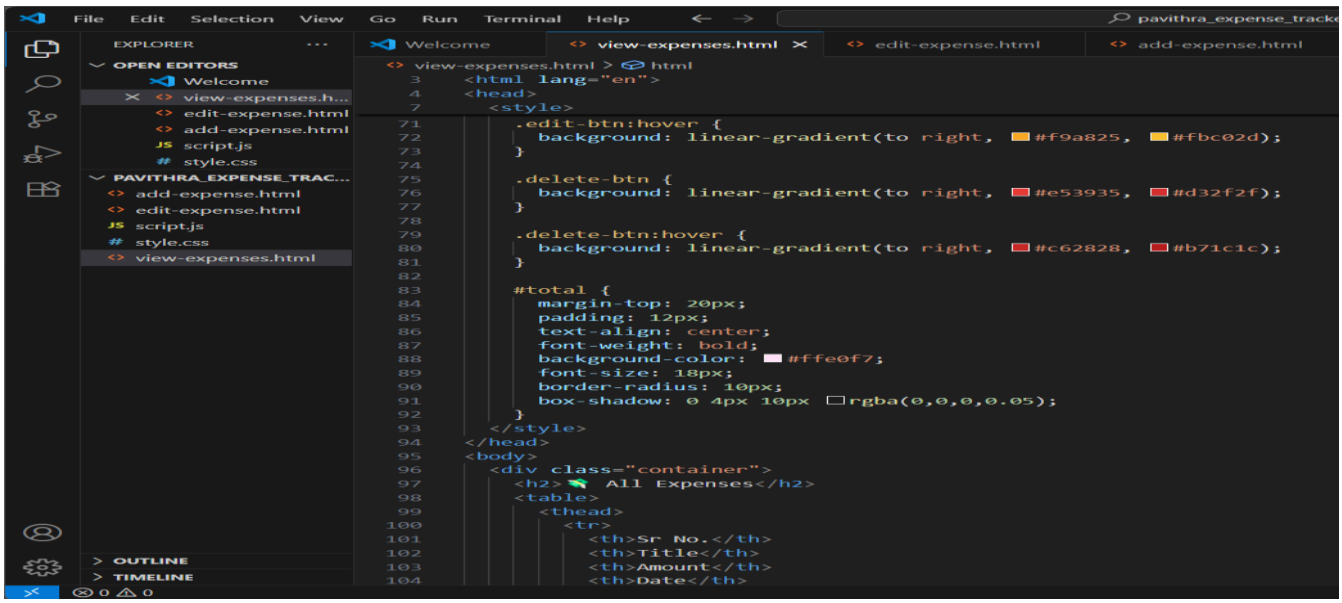
<script src="script.js"></script>
</body>
</html>

## View-expense.html

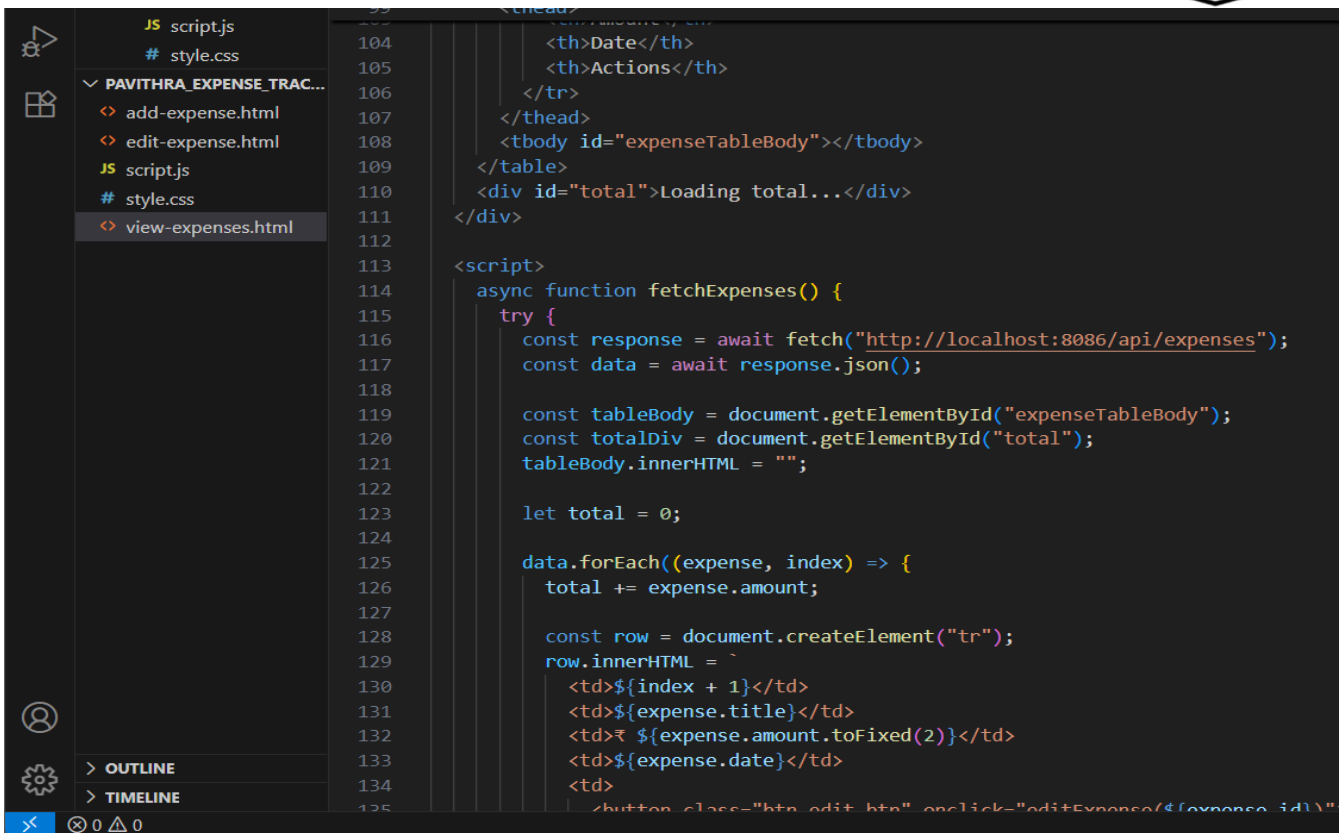


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>All Expenses</title>
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      margin: 0;
      padding: 30px;
      background: linear-gradient(120deg, #e0f7fa, #fce4ec);
      color: #333;
    }

    .container {
      max-width: 1000px;
      margin: auto;
      background: rgba(255, 255, 255, 0.9);
      border-radius: 20px;
      padding: 30px;
      box-shadow: 0 12px 25px rgba(0, 0, 0, 0.1);
    }

    h2 {
      text-align: center;
      color: #6a1b9a;
      font-size: 28px;
      margin-bottom: 30px;
    }

    table {
      width: 100%;
      border-collapse: collapse;
      background-color: #ffffff;
      box-shadow: 0 8px 20px rgba(0, 0, 0, 0.08);
      border-radius: 10px;
```

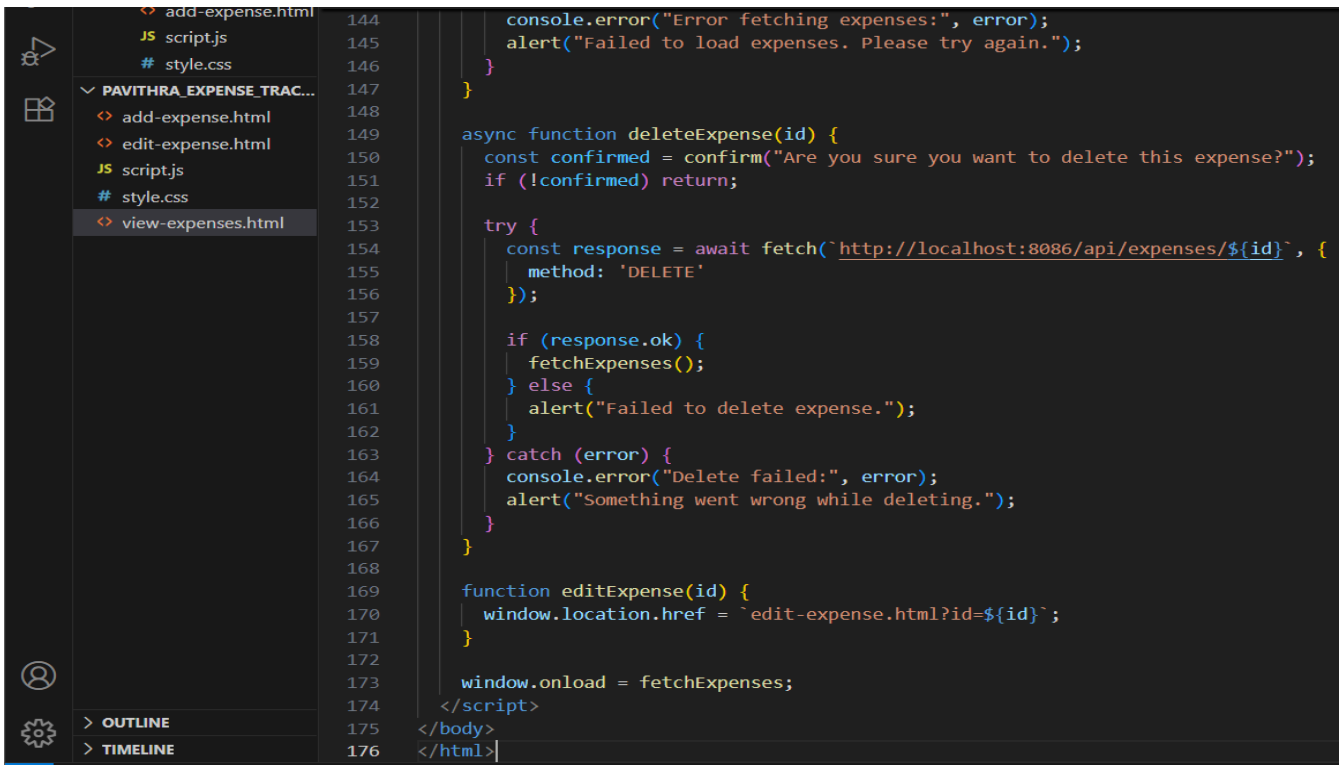
```
    table {
      overflow: hidden;
    }

    th, td {
      padding: 14px;
      text-align: center;
      border-bottom: 1px solid #eeeeee;
    }

    th {
      background: linear-gradient(to right, #2196f3, #21cbf3);
      color: white;
      font-size: 16px;
    }

    tr:hover {
      background-color: #e8f5ff;
    }

    .btn {
      padding: 8px 14px;
      border: none;
      border-radius: 6px;
      color: white;
      font-weight: bold;
      cursor: pointer;
      transition: background 0.3s ease;
    }

    .edit-btn {
      background: linear-gradient(to right, #fbc02d, #fdd835);
    }
```



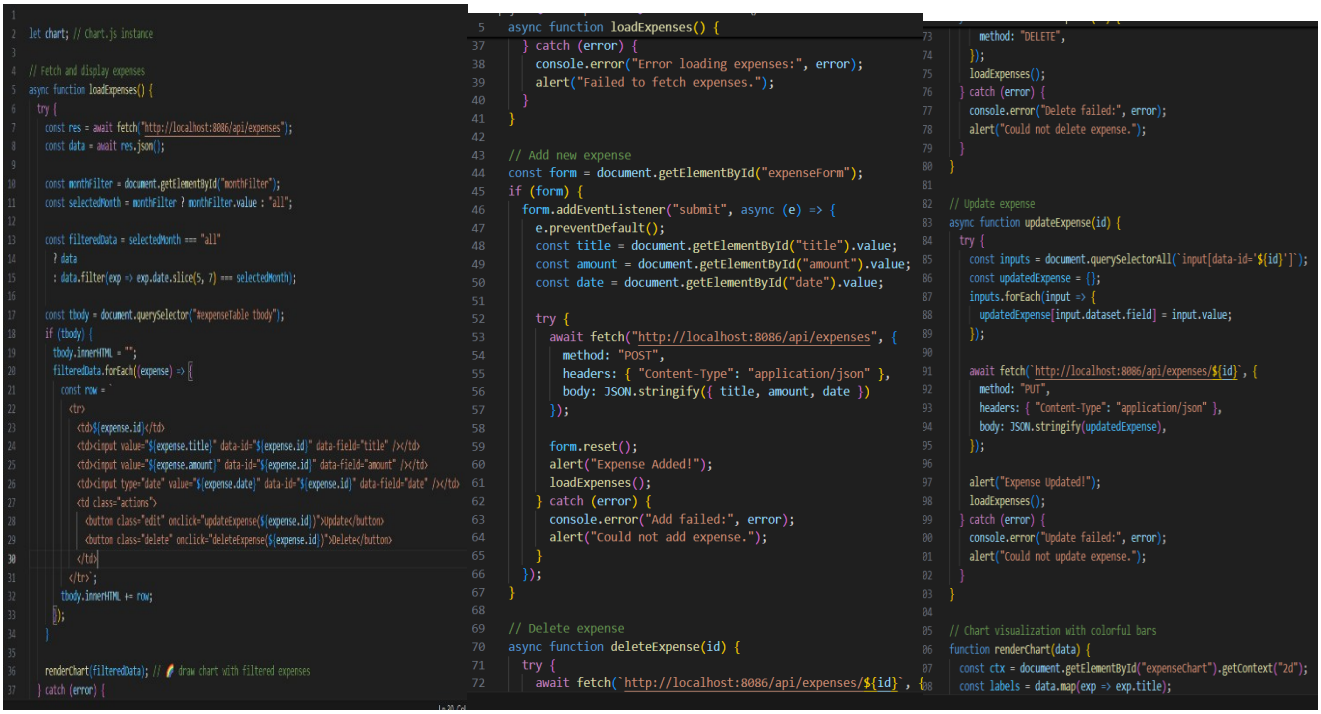
```
    .edit-btn:hover {
      background: linear-gradient(to right, #f9a825, #fbc02d);
    }

    .delete-btn {
      background: linear-gradient(to right, #e53935, #d32f2f);
    }

    .delete-btn:hover {
      background: linear-gradient(to right, #c62828, #b71c1c);
    }

    #total {
      margin-top: 20px;
      padding: 12px;
      text-align: center;
      font-weight: bold;
      background-color: #ffe0f7;
      font-size: 18px;
      border-radius: 10px;
      box-shadow: 0 4px 10px rgba(0,0,0,0.05);
    }
  </style>
</head>
<body>
  <div class="container">
    <h2>🌸 All Expenses</h2>
    <table>
      <thead>
        <tr>
          <th>Sr No.</th>
          <th>Title</th>
          <th>Amount</th>
          <th>Date</th>
```


script.js
style.css
PAVITHRA_EXPENSE_TRAC...
add-expense.html
edit-expense.html
script.js
style.css
view-expenses.html
OUTLINE
TIMELINE
0 0
<th>Date</th>
<th>Actions</th>
</tr>
</thead>
<tbody id="expenseTableBody"></tbody>
</table>
Loading total...

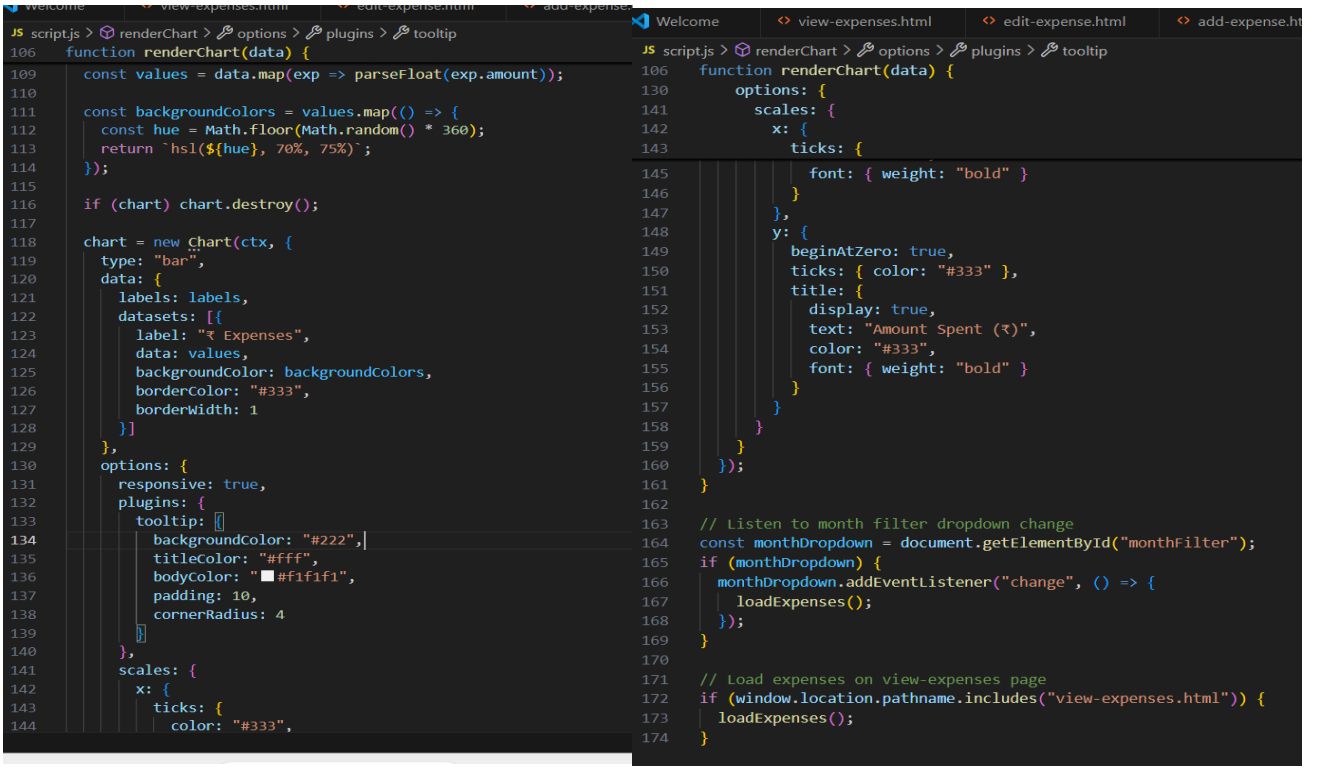
<script>
async function fetchExpenses() {
try {
const response = await fetch("http://localhost:8086/api/expenses");
const data = await response.json();
const tableBody = document.getElementById("expenseTableBody");
const totalDiv = document.getElementById("total");
tableBody.innerHTML = "";
let total = 0;
data.forEach((expense, index) => {
total += expense.amount;
const row = document.createElement("tr");
row.innerHTML = `
<td>${index + 1}</td>
<td>${expense.title}</td>
<td>₹ ${expense.amount.toFixed(2)}</td>
<td>${expense.date}</td>
<td>


add-expense.html
script.js
style.css
PAVITHRA_EXPENSE_TRAC...
add-expense.html
edit-expense.html
script.js
style.css
view-expenses.html
OUTLINE
TIMELINE
console.error("Error fetching expenses:", error);
alert("Failed to load expenses. Please try again.");
}
}
async function deleteExpense(id) {
const confirmed = confirm("Are you sure you want to delete this expense?");
if (!confirmed) return;
try {
const response = await fetch(`http://localhost:8086/api/expenses/${id}`, {
method: 'DELETE'
});
if (response.ok) {
fetchExpenses();
} else {
alert("Failed to delete expense.");
}
} catch (error) {
console.error("Delete failed:", error);
alert("Something went wrong while deleting.");
}
}
function editExpense(id) {
window.location.href = `edit-expense.html?id=${id}`;
}
window.onload = fetchExpenses;
</script>
</body>
</html>

## Script.js

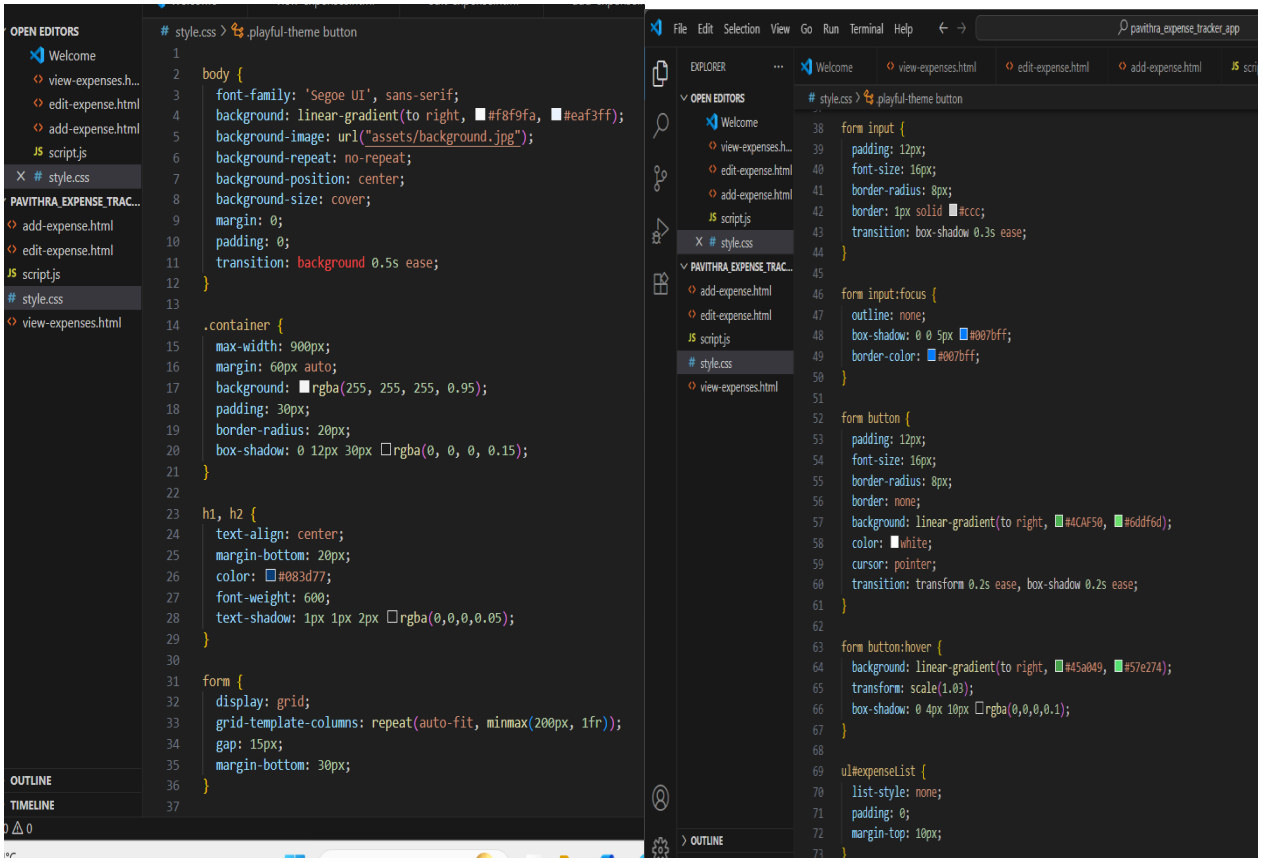


```
let chart; // Chart.js instance

// Fetch and display expenses
async function loadExpenses() {
  try {
    const res = await fetch("http://localhost:8086/api/expenses");
    const data = await res.json();

    const monthFilter = document.getElementById("monthFilter");
    const selectedMonth = monthFilter ? monthFilter.value : "all";

    const filteredData = selectedMonth === "all"
      ? data
      : data.filter(exp => exp.date.slice(5, 7) === selectedMonth);

    const tbody = document.querySelector("#expenseTable tbody");
    if (tbody) {
      tbody.innerHTML = "";
      filteredData.forEach((expense) => {
        const row = `
          <tr>
            <td>${expense.id}</td>
            <td><input value="${expense.title}" data-id="${expense.id}" data-field="title" /></td>
            <td><input value="${expense.amount}" data-id="${expense.id}" data-field="amount" /></td>
            <td><input type="date" value="${expense.date}" data-id="${expense.id}" data-field="date" /></td>
            <td class="actions">
              <button class="edit" onclick="updateExpense(${expense.id})">Update</button>
              <button class="delete" onclick="deleteExpense(${expense.id})">Delete</button>
            </td>
          </tr>`;
        tbody.innerHTML += row;
      });
    }

    renderChart(filteredData); // 🌈 draw chart with filtered expenses
  } catch (error) {
    console.error("Error loading expenses:", error);
    alert("Failed to fetch expenses.");
  }
}

// Add new expense
const form = document.getElementById("expenseForm");
if (form) {
  form.addEventListener("submit", async (e) => {
    e.preventDefault();
    const title = document.getElementById("title").value;
    const amount = document.getElementById("amount").value;
    const date = document.getElementById("date").value;

    try {
      await fetch("http://localhost:8086/api/expenses", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ title, amount, date })
      });

      form.reset();
      alert("Expense Added!");
      loadExpenses();
    } catch (error) {
      console.error("Add failed:", error);
      alert("Could not add expense.");
    }
  });
}

// Delete expense
async function deleteExpense(id) {
  try {
    await fetch(`http://localhost:8086/api/expenses/${id}`, {
      method: "DELETE",
    });
    loadExpenses();
  } catch (error) {
    console.error("Delete failed:", error);
    alert("Could not delete expense.");
  }
}

// Update expense
async function updateExpense(id) {
  try {
    const inputs = document.querySelectorAll(`input[data-id='${id}']`);
    const updatedExpense = {};
    inputs.forEach(input => {
      updatedExpense[input.dataset.field] = input.value;
    });

    await fetch(`http://localhost:8086/api/expenses/${id}`, {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(updatedExpense),
    });

    alert("Expense Updated!");
    loadExpenses();
  } catch (error) {
    console.error("Update failed:", error);
    alert("Could not update expense.");
  }
}

// Chart visualization with colorful bars
function renderChart(data) {
  const ctx = document.getElementById("expenseChart").getContext("2d");
  const labels = data.map(exp => exp.title);
```



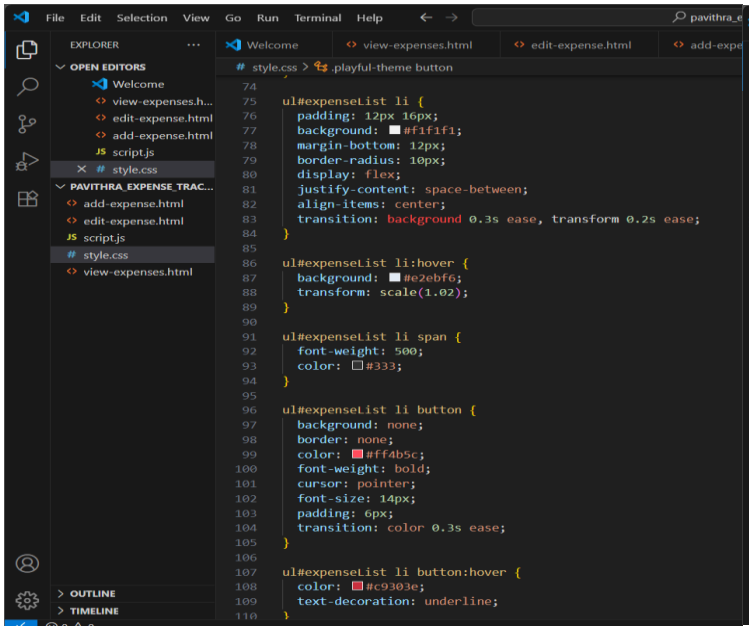
```
function renderChart(data) {
  const values = data.map(exp => parseFloat(exp.amount));

  const backgroundColors = values.map(() => {
    const hue = Math.floor(Math.random() * 360);
    return `hsl(${hue}, 70%, 75%)`;
  });

  if (chart) chart.destroy();

  chart = new Chart(ctx, {
    type: "bar",
    data: {
      labels: labels,
      datasets: [{
        label: "₹ Expenses",
        data: values,
        backgroundColor: backgroundColors,
        borderColor: "#333",
        borderWidth: 1
      }]
    },
    options: {
      responsive: true,
      plugins: {
        tooltip: {
          backgroundColor: "#222",
          titleColor: "#fff",
          bodyColor: "#f1f1f1",
          padding: 10,
          cornerRadius: 4
        }
      },
      scales: {
        x: {
          ticks: {
            color: "#333",
            font: { weight: "bold" }
          }
        },
        y: {
          beginAtZero: true,
          ticks: { color: "#333" },
          title: {
            display: true,
            text: "Amount Spent (₹)",
            color: "#333",
            font: { weight: "bold" }
          }
        }
      }
    }
  });
}

// Listen to month filter dropdown change
const monthDropdown = document.getElementById("monthFilter");
if (monthDropdown) {
  monthDropdown.addEventListener("change", () => {
    loadExpenses();
  });
}

// Load expenses on view-expenses page
if (window.location.pathname.includes("view-expenses.html")) {
  loadExpenses();
}
```

# Style.css

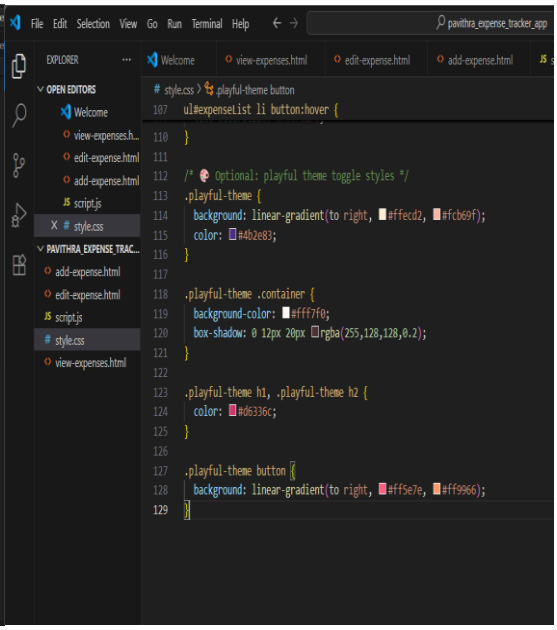


```
body {
  font-family: 'Segoe UI', sans-serif;
  background: linear-gradient(to right, #f8f9fa, #eaf3ff);
  background-image: url("assets/background.jpg");
  background-repeat: no-repeat;
  background-position: center;
  background-size: cover;
  margin: 0;
  padding: 0;
  transition: background 0.5s ease;
}

.container {
  max-width: 900px;
  margin: 60px auto;
  background: rgba(255, 255, 255, 0.95);
  padding: 30px;
  border-radius: 20px;
  box-shadow: 0 12px 30px rgba(0, 0, 0, 0.15);
}

h1, h2 {
  text-align: center;
  margin-bottom: 20px;
  color: #083d77;
  font-weight: 600;
  text-shadow: 1px 1px 2px rgba(0,0,0,0.05);
}

form {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 15px;
  margin-bottom: 30px;
}
```

```
form input {
  padding: 12px;
  font-size: 16px;
  border-radius: 8px;
  border: 1px solid #ccc;
  transition: box-shadow 0.3s ease;
}

form input:focus {
  outline: none;
  box-shadow: 0 0 5px #007bff;
  border-color: #007bff;
}

form button {
  padding: 12px;
  font-size: 16px;
  border-radius: 8px;
  border: none;
  background: linear-gradient(to right, #4CAF50, #6ddf6d);
  color: white;
  cursor: pointer;
  transition: transform 0.2s ease, box-shadow 0.2s ease;
}

form button:hover {
  background: linear-gradient(to right, #45a049, #57e274);
  transform: scale(1.03);
  box-shadow: 0 4px 10px rgba(0,0,0,0.1);
}

ul#expenseList {
  list-style: none;
  padding: 0;
  margin-top: 10px;
}
```

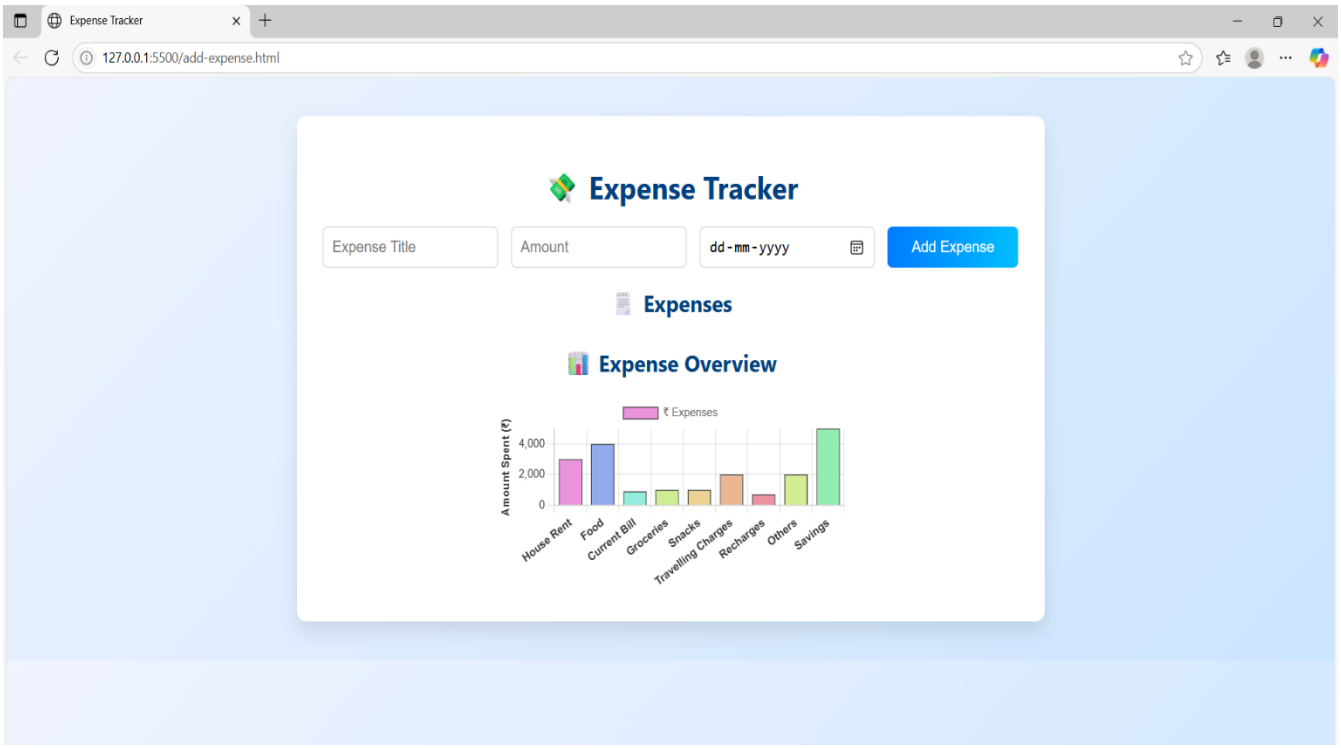


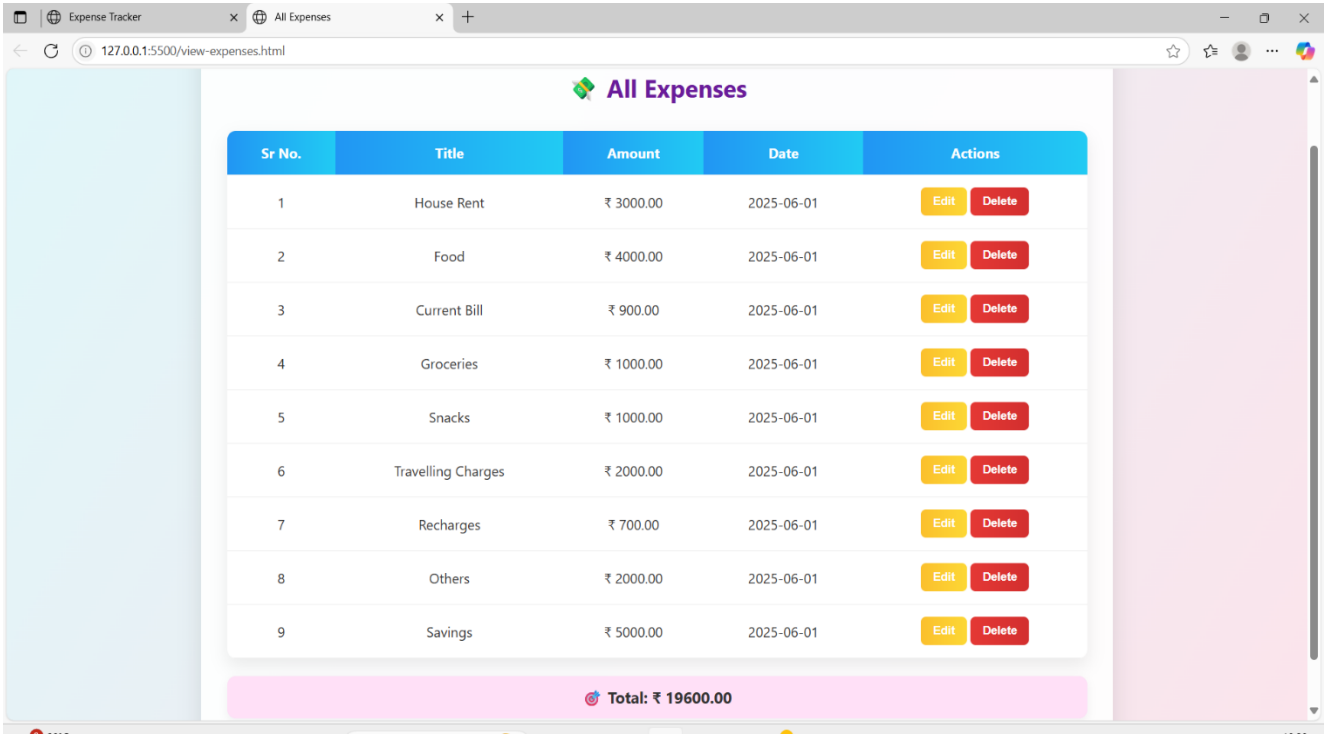
```
ul#expenseList li {
  padding: 12px 16px;
  background: #f1f1f1;
  margin-bottom: 12px;
  border-radius: 10px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  transition: background 0.3s ease, transform 0.2s ease;
}

ul#expenseList li:hover {
  background: #e2ebf6;
  transform: scale(1.02);
}

ul#expenseList li span {
  font-weight: 500;
  color: #333;
}

ul#expenseList li button {
  background: none;
  border: none;
  color: #ff4b5c;
  font-weight: bold;
  cursor: pointer;
  font-size: 14px;
  padding: 6px;
  transition: color 0.3s ease;
}

ul#expenseList li button:hover {
  color: #c9303e;
  text-decoration: underline;
}
```



```
ul#expenseList li button:hover {
}

/* 🎨 Optional: playful theme toggle styles */
.playful-theme {
  background: linear-gradient(to right, #ffecd2, #fcb69f);
  color: #4b2e83;
}

.playful-theme .container {
  background-color: #fff7f0;
  box-shadow: 0 12px 20px rgba(255,128,128,0.2);
}

.playful-theme h1, .playful-theme h2 {
  color: #d6336c;
}

.playful-theme button {
  background: linear-gradient(to right, #ff5e7e, #ff9966);
}
```

# Outputs:


Expense Tracker
127.0.0.1:5500/add-expense.html
Expense Tracker
Expense Title
Amount
dd-mm-yyyy
Add Expense
Expenses
Expense Overview
₹ Expenses
Amount Spent (₹)
4,000
2,000
0
House Rent
Food
Current Bill
Groceries
Snacks
Travelling Charges
Recharges
Others
Savings


Expense Tracker
All Expenses
127.0.0.1:5500/view-expenses.html
All Expenses
Sr No. | Title | Amount | Date | Actions
1 | House Rent | ₹ 3000.00 | 2025-06-01 | Edit Delete
2 | Food | ₹ 4000.00 | 2025-06-01 | Edit Delete
3 | Current Bill | ₹ 900.00 | 2025-06-01 | Edit Delete
4 | Groceries | ₹ 1000.00 | 2025-06-01 | Edit Delete
5 | Snacks | ₹ 1000.00 | 2025-06-01 | Edit Delete
6 | Travelling Charges | ₹ 2000.00 | 2025-06-01 | Edit Delete
7 | Recharges | ₹ 700.00 | 2025-06-01 | Edit Delete
8 | Others | ₹ 2000.00 | 2025-06-01 | Edit Delete
9 | Savings | ₹ 5000.00 | 2025-06-01 | Edit Delete
Total: ₹ 19600.00

## Conclusion

The Expense Tracker App was developed to provide users with a robust, user-friendly platform to monitor and manage their personal finances effectively. Through the integration of a powerful Spring Boot backend and a dynamic HTML/CSS/JavaScript frontend, the application ensures seamless interaction for adding, editing, deleting, and analyzing expenses. With key features like budget planning, category-wise expense classification, reminders for due payments, and visualization through charts, the app simplifies complex financial tracking into an intuitive and organized system.

The successful implementation of this full-stack solution demonstrates not only technical proficiency but also the importance of user-centric design and practical functionality. The application was built with scalability and modularity in mind, allowing future enhancements like authentication, detailed analytics, and mobile compatibility. Overall, the Expense Tracker App serves as a complete and real-world example of a full-stack Java project that addresses essential financial management needs while offering a solid foundation for further innovation and expansion.