**CAPSTONE PROJECT: Expense Tracker Application**

**Project Overview**

**The goal is to build a Full-Stack Expense Tracker Application that allows users to manage their personal finances. Users can add expenses, view categorized spending summaries, and generate monthly reports. The frontend will leverage Vuex or any other equivalent for state management to manage the user's session and expense data, ensuring seamless updates across components. The backend will handle data persistence and expose REST APIs using Spring Boot.**

**Why This Use Case?**

* **Real-world Scenario: Personal finance management is relatable and practical.**
* **Vuex Emphasis: State management is crucial as expenses, filters, and reports need to be shared across components.**
* **CRUD + Advanced Functionality: Beyond basic CRUD operations, you’ll implement reporting and filtering features.**
* **Complexity: This project introduces challenges like data categorization, real-time UI updates, and persistent authentication.**

**Project Goals**

1. **Vue.js with Vuex:**
   * **Centralize state management for expenses, user session, and filters.**
   * **Demonstrate real-time updates across components.**
2. **Spring Boot Backend:**
   * **Design RESTful APIs for CRUD operations, data retrieval, and reporting.**
   * **Implement secure user authentication.**
3. **Git & GitHub Actions:**
   * **Proper version control using feature branching and pull requests.**
   * **CI/CD pipeline automation for testing and deployment.**
4. **Deployment:**
   * **Deploy a working version locally (Dockerized) or on the cloud.**

**Functional Requirements**

**1. User Management**

* **Register & Login: Allow users to register and log in using basic authentication.**
* **Session Management: Keep users authenticated using JWT or Vuex state management.**

**2. Expense Management**

* **CRUD Operations:**
  + **Add an expense with fields: Title, Amount, Category, and Date.**
  + **View a list of expenses.**
  + **Edit and Delete existing expenses.**
* **Expense Categorization:**
  + **Categories: Food, Transport, Utilities, Entertainment, Miscellaneous.**
  + **Allow users to filter expenses by category, date range, or keyword.**
* **Real-Time State Updates:**
  + **Vuex will manage and update expenses globally across components.**
  + **Changes like adding or editing an expense should instantly reflect in the list.**

**3. Reporting and Summaries**

* **Monthly Summary:**
  + **Show total expenses for the current month.**
  + **Breakdown by categories (e.g., pie chart or bar graph using Chart.js).**
* **Search & Filter:**
  + **Filter expenses by category and date range.**

**4. UI/UX Expectations**

* **Clean, Responsive UI:**
  + **Use Bootstrap or Tailwind CSS for a modern and responsive design.**
* **Componentization:**
  + **Separate components for Login, Expense List, Expense Form, and Reports.**
* **Dynamic Feedback:**
  + **Notifications for actions (e.g., expense added, deleted).**

**Technical Expectations**

**Frontend (Vue.js + Vuex)**

1. **State Management with Vuex:**
   * **State:**
     + **user: Stores user authentication state.**
     + **expenses: List of user expenses fetched from the backend.**
   * **Mutations: Modify expenses and user state (e.g., ADD\_EXPENSE, DELETE\_EXPENSE).**
   * **Actions: Asynchronous operations for API calls (e.g., fetch expenses, delete expense).**
   * **Getters: Retrieve filtered data (e.g., total expenses, expenses by category).**
2. **Components:**
   * **LoginComponent.vue: Handles user login and authentication.**
   * **ExpenseListComponent.vue: Displays the list of expenses with filters.**
   * **ExpenseFormComponent.vue: Form to add or edit expenses.**
   * **ReportComponent.vue: Displays charts or summaries for expenses.**
3. **API Integration: Use Axios to communicate with Spring Boot backend APIs.**

**Backend (Spring Boot)**

1. **APIs:**
   * **POST /api/login: Authenticate user.**
   * **GET /api/expenses: Retrieve expenses for a user.**
   * **POST /api/expenses: Add a new expense.**
   * **PUT /api/expenses/{id}: Edit an existing expense.**
   * **DELETE /api/expenses/{id}: Delete an expense.**
2. **Database:**
   * **Use PostgreSQL or H2 for data persistence.**
   * **Entities: User and Expense.**
3. **Security:**
   * **Secure all endpoints with Basic Authentication.**
4. **Testing:**
   * **Write JUnit tests for APIs and service layers.**

**Git & GitHub Actions**

1. **Git:**
   * **Use feature branches (feature/expenses, feature/reports).**
   * **Follow version control best practices (commits, pull requests).**
2. **GitHub Actions CI/CD:**
   * **Build the Spring Boot backend using Maven.**
   * **Build the Vue.js frontend using npm run build.**
   * **Run backend unit tests during the pipeline.**
   * **Generate the deployable.**

**Deliverables**

1. **Source Code: A GitHub repository with separate folders for frontend (Vue.js) and backend (Spring Boot).**
2. **CI/CD Workflow: GitHub Actions pipeline file (.github/workflows/ci-cd.yml).**
3. **Screenshots: UI screenshots for login, expense management, and reports.**
4. **Deployed Application: Provide a URL to the deployed application (optional).**
5. **Documentation:**
   * **README.md with setup instructions.**
   * **Explanation of Vuex implementation and GitHub Actions workflow.**

**Evaluation Criteria**

1. **Vuex Integration: Proper state management for user and expense data.**
2. **Spring Boot APIs: Functional and well-tested REST APIs.**
3. **CI/CD Workflow: Working GitHub Actions pipeline.**
4. **Code Quality: Clean, modular, and maintainable code.**
5. **Functionality: Complete expense management features and reporting.**