

Full Stack Development with MERN – Project Documentation

1. Introduction

Project Title: Personal Expense Tracker

Team Members:

- Jai – Backend Developer
- Abhay – Full Stack Integration
- Devansh – Frontend Developer

2. Project Overview

Purpose

The purpose of this project is to develop a responsive, secure, and user-friendly Personal Expense Tracker that allows users to manage their finances by tracking income and expenses, setting monthly budgets, and visualizing financial habits. Many individuals, especially students and early professionals, struggle with financial discipline and lack tools to monitor where their money goes. This application solves that by providing a centralized platform to log, categorize, and analyze transactions in real-time.

This system is designed to encourage better budgeting practices, provide insights into spending patterns, and help users become more financially aware and responsible.

Goals

- Help users gain better control of their daily, weekly, and monthly expenses.
- Offer a seamless experience on both desktop and mobile platforms.
- Ensure secure access to financial data using modern authentication protocols.
- Provide actionable insights and visualization through charts and summaries.
- Support a scalable architecture that can grow with user demand.

Feature

User Authentication & Authorization

- Secure login and registration using email/password.
- JWT-based session handling with protected routes.

Expense & Income Management

- Add, update, and delete income/expense entries.
- Categorize transactions (e.g., Food, Rent, Utilities).
- Filter by date, type, and category.

Dashboard & Visual Insights

- Real-time summary of income, expenses, and balance.
- Pie and bar charts for category-wise analysis.
- Monthly trends to track financial behavior.

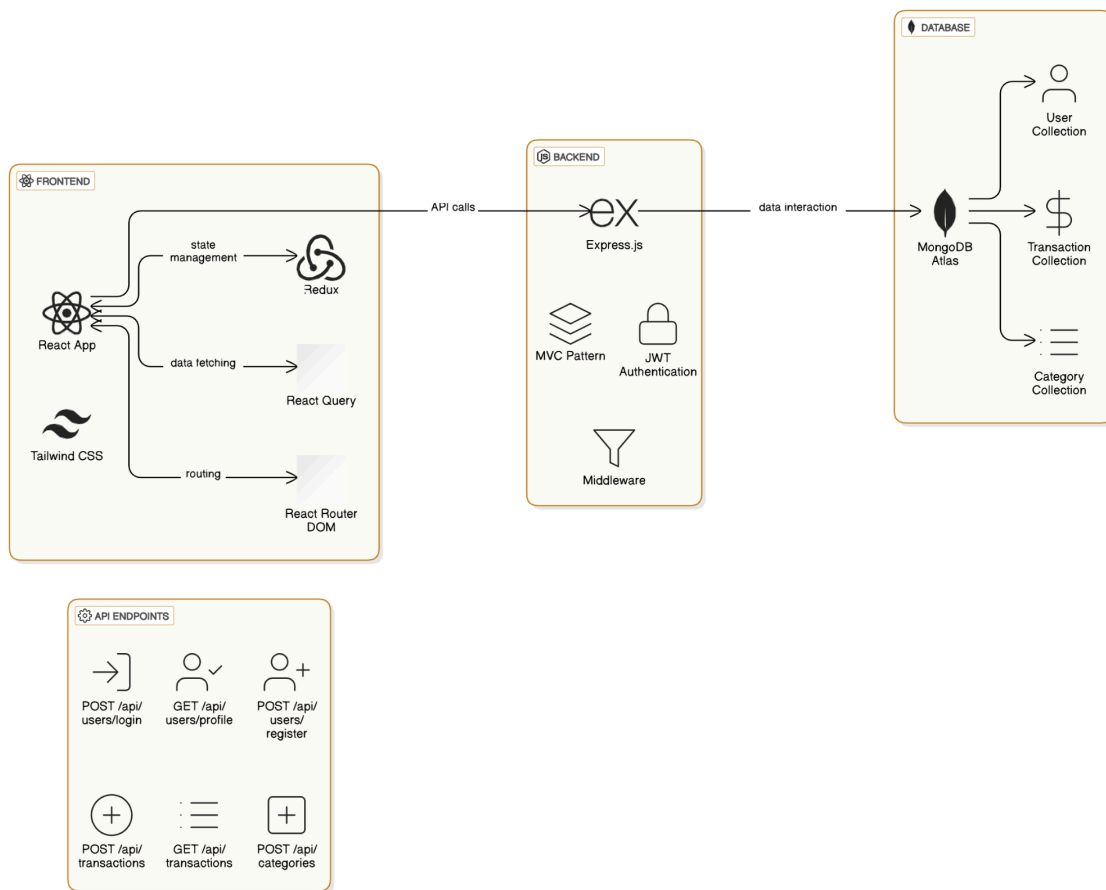
Responsive Design

- Mobile-friendly interface with dark mode.
- Built using Tailwind CSS for a clean UI.

3. Architecture

The Expense Tracker application follows a **client-server architecture** based on the **MERN stack** (MongoDB, Express.js, React.js, Node.js). It adopts the **MVC (Model-View-Controller)** design pattern on the backend and integrates **state management tools** like Redux and React Query on the frontend for better data flow and synchronization.

Architecture Diagram



Frontend (React.js + Redux + React Query)

- Built with **React.js** and styled using **Tailwind CSS**
- **React Redux** handles global state such as user login status and theme
- **React Query** manages API data (server state), caching and background refetching

- Implements **React Router DOM** for routing and **Authenticated Routes** for page protection

Backend (Node.js + Express.js)

- Organized using **MVC pattern**
- Uses **Express.js** to define RESTful API routes
- Implements **JWT-based authentication**
- Applies **middleware** for error handling, validation, and route protection
- Interacts with MongoDB using **Mongoose models**

Database (MongoDB)

- Cloud-hosted **MongoDB Atlas**
 - Stores:
 - User data (credentials, profile)
 - Transaction data (type, amount, category, date, description)
 - Category data (Income/Expense categories)
-

4. Setup Instructions

Prerequisites:

- Node.js (v18 or higher)
- MongoDB (local or use MongoDB Atlas)
- Git
- Code Editor (e.g., VS Code)

Installation:

1. Clone the Repository

```
https://github.com/JV456/SmartBridge-Fullstack-Development-MongoDB-Externs  
hip.git
```

```
cd PersonalExpenseTrackerApp
```

2. Install Backend Dependencies

```
cd backend
```

```
"bcryptjs": "^3.0.2",  
"cors": "^2.8.5",  
"express": "^4.21.2",  
"express-async-handler": "^1.2.0",  
"jsonwebtoken": "^9.0.2",  
"mongoose": "^8.13.1"
```

3. Install Frontend Dependencies

```
cd frontend
```

```
"@headlessui/react": "^2.2.1",  
"@heroicons/react": "^2.2.0",  
"@reduxjs/toolkit": "^2.6.1",
```

```
"@tailwindcss/vite": "^4.0.17",
"@tanstack/react-query": "^5.70.0",
"@tanstack/react-query-devtools": "^5.70.0",
"aos": "^2.3.4",
"axios": "^1.8.4",
"chart.js": "^4.4.8",
"formik": "^2.4.6",
"framer-motion": "^12.6.3",
"react": "^19.0.0",
"react-chartjs-2": "^5.3.0",
"react-dom": "^19.0.0",
"react-icons": "^5.5.0",
"react-redux": "^9.2.0",
"react-router-dom": "^7.4.1",
"tailwindcss": "^4.0.17",
"yup": "^1.6.1"
```

Environment Setup:

Create a .env file inside the /backend/ directory and add the following variables:

```
mongodb+srv://openjai456:hNbi4pn7xZ0eN8dT@mern-expenses-cluster.dczyx8c.mongodb.net/?retryWrites=true&w=majority&appName=mern-expenses-cluster
```

5. Folder Structure

frontend/

```
|
|
|— node_modules/
|
|— public/
|
|
|— src/
|   |— assets/
|   |
|   |— components/
|   |   |— Alert/
|   |   |   |— AlertMessage.jsx
|   |   |— Auth/
|   |   |   |— AuthRoute.jsx
|   |   |— Category/
|   |   |   |— AddCategory.jsx
|   |   |   |— CategoriesList.jsx
|   |   |   |— UpdateCategory.jsx
|   |   |— Home/
|   |   |   |— HomePage.jsx
|   |   |— Navbar/
|   |   |   |— PrivateNavbar.jsx
|   |   |   |— PublicNavbar.jsx
|   |   |— Transactions/
|   |   |   |— TransactionChart.jsx
|   |   |   |— TransactionForm.jsx
```

```
| | | └─ TransactionList.jsx
| | └─ Users/
| |   └─ Dashboard.jsx
| |   └─ Login.jsx
| |   └─ Register.jsx
| |   └─ UpdatePassword.jsx
| |   └─ UserProfile.jsx
|
| └─ redux/
| | └─ slice/
| | | └─ authSlice.js
| | └─ store.js
|
| └─ services/
| | └─ category/
| | | └─ categoryService.js
| | └─ transactions/
| | | └─ transactionService.js
| | └─ users/
| |   └─ userService.js
|
| └─ utils/
| | └─ getUserFromStorage.js
| | └─ url.js
|
| └─ App.jsx
```



```
| |— App.css
| |— index.js
| |— index.css
|
|— .gitignore
|— vite.config.js
|— index.html
|— package.json
└— package-lock.json
```

Explanation:

1. components/

- Organized feature-wise — this improves scalability and maintainability.
- For example, components/Transactions includes everything for creating, listing, and visualizing transactions.
- AuthRoute.jsx helps protect routes that require login.

2. redux/

- Central state management.
- authSlice.js: Handles login/logout state.
- store.js: Combines all slices and integrates Redux Toolkit.

3. services/

- API layer using Axios or Fetch.

- Each domain (category, transactions, users) has its own service file for CRUD operations.

4. utils/

- Reusable logic like getting the logged-in user from local storage and defining the API base URL.

Root Files

- App.jsx: Main layout and route switching happens here.
- index.js: Initializes React app by rendering App.jsx.
- App.css, index.css: Styling layers.
- vite.config.js: Configuration for fast Vite development server.

backend/

```
|
|
|— controllers/
|  |— categoryCtrl.js
|  |— transactionCtrl.js
|  └— usersCtrl.js
|
|
|— middlewares/
|  |— errorHandlerMiddleware.js
|  └— isAuth.js
|
|
|— model/
|  |— Category.js
```

```
|   ├── Transaction.js
|   └── User.js
|
|── routes/
|   ├── categoryRouter.js
|   ├── transactionRouter.js
|   └── userRouter.js
|
|── utils/
|
|── .env
|── .gitignore
|── app.js
|── package.json
└── package-lock.json
```

Explanation:

1. controllers/

This is where we define the *business logic*. It receives HTTP requests and uses models to interact with the database. For example:

- usersCtrl.js: handles user registration, login, and profile retrieval.
- transactionCtrl.js: logic to add, list, or delete transactions.
- categoryCtrl.js: logic to manage user-defined categories.

2. middlewares/

Middlewares in Express are functions that execute *before* the route handler:

- `isAuth.js`: protects routes using JWT validation.
- `errorHandlerMiddleware.js`: catches and formats errors in a consistent way.

3. model/

Defines **Mongoose schemas** to interact with MongoDB. Each file defines the structure and rules for a specific type of data:

- `User.js`, `Transaction.js`, `Category.js`

4. routes/

Each router connects HTTP methods (GET, POST, PUT, DELETE) with controller functions. It's the "entry point" for clients:

- Example: `POST /api/users/register` → goes to `usersCtrl.registerUser`.

5. app.js

The root of my app: it initialises Express, connects to MongoDB, loads middlewares and routers, and starts the server.

6. Running the Application

Frontend:

```
cd PersonalExpenseTrackerApp
```

```
cd frontend
```

```
npm run dev
```

Backend:

```
cd PersonalExpenseTrackerApp
```

```
cd backend
```

```
node --watch app
```

7. API Documentation

Authentication Routes

Method	Endpoint	Description	Access	Body Parameters
POST	http://localhost:8000/api/v1/users/register	Register a new user	Public	username, email, password
POST	http://localhost:8000/api/v1/users/login	Log in an existing user	Public	email, password
GET	http://localhost:8000/api/v1/users/profile	To get the profile	Private	—
PUT	http://localhost:8000/api/v1/users/change-password	To change the password	Private	newPassword
PUT	http://localhost:8000/api/v1/users/update-profile	To update the profile	Private	username, email

Transaction Routes

Method	Endpoint	Description	Access	Body / Params
GET	http://localhost:8000/api/v1/transactions/lists	Get all transactions	Private	—

POST	http://localhost:8000/api/v1/transactions/create	Add a new transaction	Private	type, amount, category, date, description
PUT	http://localhost:8000/api/v1/transactions/update	Update a transaction	Private	id param, updated fields in body
DELETE	http://localhost:8000/api/v1/transactions/delete	Delete a transaction	Private	id param

Category Routes

Method	Endpoint	Description	Access	Body/Params
POST	http://localhost:8000/api/v1/categories/create	add a new category	Private	name, type
PUT	http://localhost:8000/api/v1/categories/update	update a category	Private	name
GET	http://localhost:8000/api/v1/categories/lists	get all categories	Private	—
Delete	http://localhost:8000/api/v1/categories/delete	Delete a category	Private	—

8. Authentication

In this Personal Expense Tracker Web App, **authentication and authorization** are implemented securely using **JWT (JSON Web Tokens)**. These mechanisms ensure that only authenticated users can access protected routes and perform actions like adding transactions, creating categories, or updating their profiles.

How Authentication Works

1. User Login:

When a user logs in using their email and password, the server (Node.js backend) verifies the credentials.

If valid, the server generates a **JWT token** and sends it back to the client.

2. Token Storage:

The token is stored on the **client-side** (in localStorage) and is attached to all future API requests in the Authorization header using the Bearer scheme.

3. Protecting Routes:

The backend uses a middleware function (isAuth.js) to **verify the token** on each protected route.

If the token is valid, the request proceeds; otherwise, an error is returned indicating unauthorized access.

4. Authorization:

Once authenticated, the user's ID (embedded in the token) is extracted and used to authorize access to user-specific data (like personal transactions or categories).

This ensures that users can **only access or modify their own data**.

5. Sessionless Authentication:

JWT provides **stateless authentication**, meaning the server doesn't store session data in memory. This makes the app scalable and simplifies session management.

Key Files Involved

→ Backend:

- ◆ **usersCtrl.js:** Handles login and registration logic.
- ◆ **isAuth.js:** Middleware that validates JWT tokens.

→ Frontend:


- ◆ **authSlice.js:** Stores the logged-in user info in Redux state.
- ◆ **getUserFromStorage.js:** Retrieves the token/user data from local storage.
- ◆ **AuthRoute.jsx:** A route guard component that restricts access to authenticated routes only.

9. User Interface

UI includes:

- Responsive layout with dark mode
- Sidebar navigation
- Expense forms
- Interactive charts
- Mobile-friendly UI

Screenshots:



Walleto

Add Transaction

Add Category

Categories

Profile

Dashboard

Logout

Add New Category

Fill in the details below

Type

Select transaction type

Name

Enter category name

Add Category

Walleto

Add Transaction

Add Category

Categories

Profile

Dashboard

Logout

Categories

Manage your transaction categories

+ Add New

<div>salary</div>	<div>Income</div>	<div><div></div></div> <div><div></div></div>
<div>waterpark</div>	<div>Expense</div>	<div><div></div></div> <div><div></div></div>
<div>household</div>	<div>Expense</div>	<div><div></div></div> <div><div></div></div>

Add New Transaction

Track your income and expenses with ease

Transaction Type

Select transaction type

\$ Amount

\$ 0.00

Category

Select a category

Date

dd-mm-yyyy

Description (Optional)

Add notes about this transaction

+ Add Transaction

Welcome

Update Profile

Username

Your username

Email

Your email

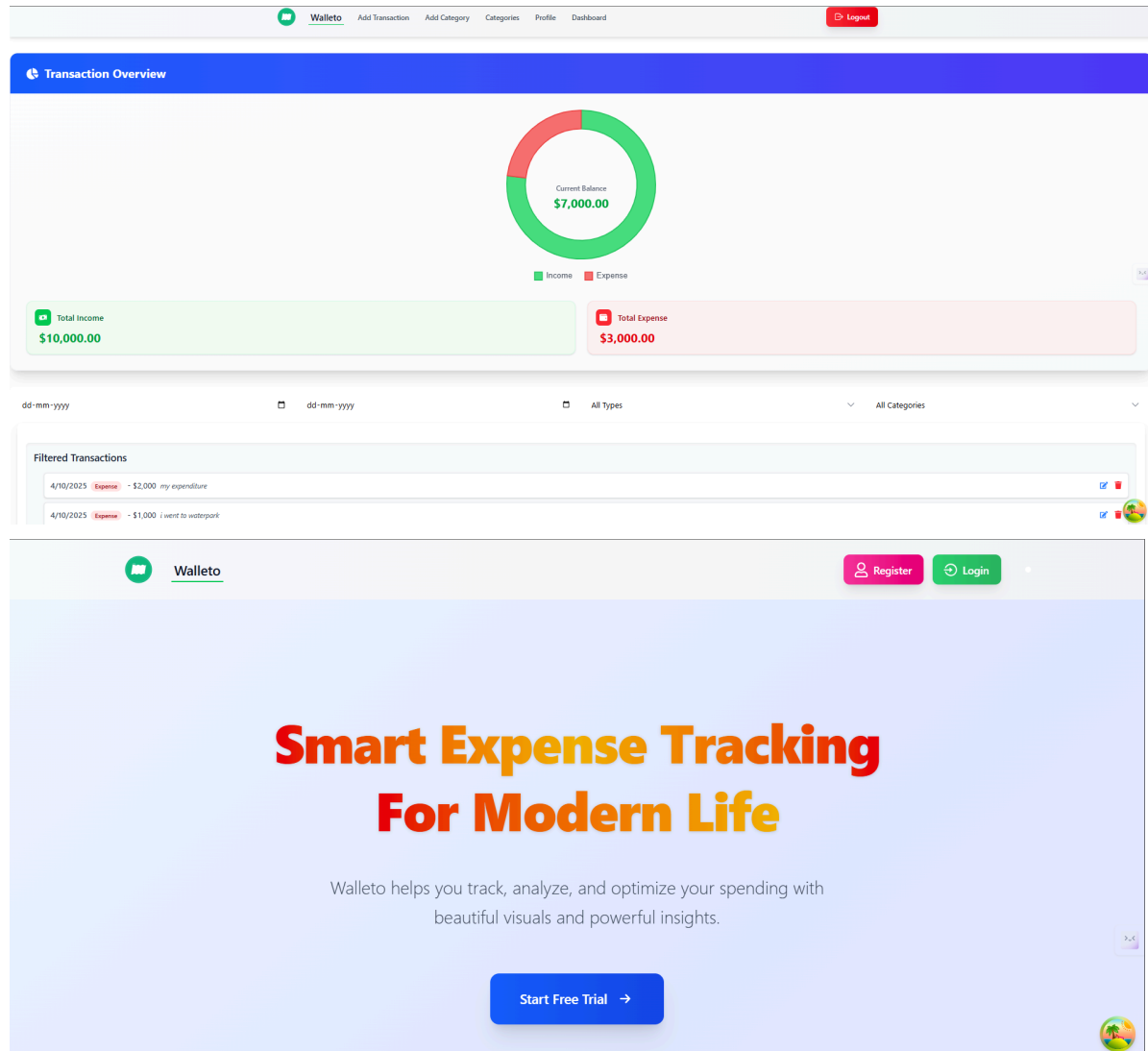
Save Changes

Change Your Password

New Password

Enter new password

Update Password



10. Testing

Frontend tested manually in browsers.

Backend API tested with Postman.

Future enhancements include adding unit tests using Jest and Mocha.

11. Screenshots or Demo

GitHub Repo:

<https://github.com/JV456/SmartBridge-Fullstack-Development-MongoDB-Externship>

Live Demo:  demoVideo.mp4

12. Known Issues

When my app starts:

- **CategoriesList.jsx** (or any protected component) mounts immediately.
- It calls **listCategoriesAPI**, which internally:
 - Uses `getUserFromStorage()` to get the JWT token from `localStorage`
 - Sends it in the Authorization header

BUT...

- At that moment, my Redux **auth.user** might still be null because:
 - Redux is not yet fully hydrated
 - My login info is in `localStorage`, but Redux is not initialized from it *yet*
 - So the request either:
 - Goes without token
 - Or with an invalid/expired token
- Result: API throws **401 Unauthorized** → **show the "Token expired" alert**

Then when i **refresh**, everything loads in sync:

- Redux reads from localStorage
- Token is available before the API call
- Everything works fine

13. Future Enhancements

- **Two-Factor Authentication (2FA)** : Add an extra layer of security during login.
- **Recurring Transactions** : Support auto-logging of monthly bills or subscriptions.
- **Currency Conversion** : Enable users to track expenses in multiple currencies.
- **In-App Notifications** : Alert users about budget limits, spending trends, and account activity