# Investment Portfolio Management System

## 1. Overview

This project is a web-based investment portfolio management system built using Vue.js and Vuex. It allows tracking of clients, their investment portfolios, and individual holdings. Users can perform CRUD operations on portfolios, holdings, and clients. The system features:

- Interactive dashboards and stats cards.
- Portfolio and user management views.
- Modular Vuex stores for state management.
- Async actions simulating API calls.
- Tailwind CSS for styling.

## 2. Project Structure

**Folder structure:**

```
src/
├── components/        # Reusable Vue components (NavigationBar, Footer, etc.)
├── views/             # Pages for portfolios and users
│   ├── Home.vue
│   ├── portfolios/
│   │   ├── PortfolioList.vue
│   │   ├── PortfolioForm.vue
│   │   └── PortfolioDetail.vue
│   └── users/
│       ├── UserList.vue
│       └── UserForm.vue
├── store/             # Vuex modules
│   ├── portfolio.js
│   └── user.js
├── App.vue
└── main.js
```

## 3. Components & Views

### 3.1 Portfolio Views

- **PortfolioList.vue**: Lists all portfolios with status filtering (Active, Upcoming, Closed).
- **PortfolioForm.vue**: Form for adding or updating portfolio details.
- **PortfolioDetail.vue**: Shows detailed holdings, total value, and returns for a specific portfolio.

### 3.2 User Views

- **UserList.vue**: Displays all clients with basic details.
- **UserForm.vue**: Form to add or update user information.

### 3.3 Navigation

- **NavigationBar.vue**: Handles navigation between Home, Portfolios, and Users pages.

## 4. State Management (Vuex)

The system uses two namespaced Vuex modules:

### 4.1 Portfolio Module

**State**: portfolios array, loading, error.

**Mutations**: CRUD operations for portfolios and holdings, set loading, set error.

**Actions**: Async operations for adding/updating/deleting portfolios and holdings.

**Getters**: Filter portfolios by status, client, ID, and provide summary counts.

### 4.2 User Module

**State**: users array, nextId, loading, error.

**Mutations**: CRUD operations for users, set loading, set error.

**Actions**: Async operations for adding/updating/deleting users.

**Getters**: Fetch all users, by ID, total count, loading, and error state.

## 5. Data Flow

1. User interaction (e.g., adding a portfolio) triggers a component event.
2. The component calls a Vuex action (async).
3. Action commits a mutation to update the state.
4. Getters compute derived data for UI display.
5. UI components automatically reactively update based on state changes.

## 6. Example Workflows

### 6.1 Adding a Portfolio

- User fills PortfolioForm.
- Form emits submit event → action addPortfolio → mutation ADD_PORTFOLIO.
- State updates → Dashboard stats update → PortfolioList refreshes.

### 6.2 Updating a Holding

- User edits a holding in PortfolioDetail.vue.
- Component dispatches updateHolding → mutation UPDATE_HOLDING.
- Portfolio's total value and returns are recomputed via getters.

### 6.3 Managing Users

- UserForm used for adding/updating users.
- Vuex handles ADD_USER, UPDATE_USER, DELETE_USER mutations.
- UserList automatically reflects changes.

## 7. Computed Data (Getters)

- allPortfolios, portfolioById, activePortfolios, upcomingPortfolios, closedPortfolios, portfoliosCount.
- allUsers, userById, usersCount.
- Ensures consistent reactive updates across components.

## 8. Styling

- Uses Tailwind CSS for layout, cards, buttons, and responsive grid design.
- Portfolio status colors are dynamically assigned using helper functions.

## 9. Summary

The system is modular, reactive, and scalable:

- Components handle UI and emit events.
- Vuex stores maintain centralized state and business logic.
- Getters provide filtered views and summaries.
- Tailwind CSS ensures consistent styling across views.

**GitHub Repo** - https://github.com/Rithika-Mamilla/investment-portfolio-management

**Instructions to Run the Project**

**1. Prerequisites**

Before running the project, ensure the following are installed on your system:

- **Node.js** (v16 or higher recommended)
- **npm** (comes with Node.js)
- **Git** (for cloning the repository)

**2. Clone the Repository**

Clone the project repository to your local system:

git clone https://github.com/Rithika-Mamilla/investment-portfolio-management.git

Navigate into the project folder:

cd investment-portfolio-management

**3. Install Dependencies**

Install all required Node.js dependencies using npm: npm install

This will install all packages including Vue, Vuex, Vite, TailwindCSS, and other dependencies.

**4. Run the Development Server**

To run the project in development mode: npm run dev

- This will start a local development server.
- Open a browser and go to the URL displayed in the terminal (usually http://localhost:5173).

**5. Run Tests**

To run unit tests (using Jest): npm run test

- This will execute all test cases and display results in the terminal.

**6. Notes**

- The project uses Vue 3 + Vite + Vuex + Tailwind CSS.
- All data is currently mocked in Vuex stores (portfolio.js and user.js). No backend is required.