Investment Portfolio Management System — Code Logic Documentation

---

1. Project Overview

The **Investment Portfolio Management System (IPMS)** is a front-end Single Page Application that helps advisors and investors manage portfolios and clients entirely in the browser. It supports login & signup flows, portfolio list/detail/create-edit, client list/create-edit, demo data seeding, and INR-format reporting.

The application follows a clean, production-style layered SPA architecture (**Views/Components → Vuex Store → Router → Mock Persistence**), ensuring maintainability, scalability, and a clear separation between presentation logic, state management, and data persistence.

Architecture: Views/Components → Store (Vuex) → Router → Mock (localStorage)

---

2. System Architecture and Design

Layer	Folder	Responsibility	Key Files
Presentation (Views/Components)	`src/views`, `src/components/ui`	Screen layouts, forms, tables, alerts, headers	`PortfolioList.vue`, `PortfolioForm.vue`, `PortfolioDetail.vue`, `UserList.vue`, `UserForm.vue`, `LoginPage.vue`, `SignUpPage.vue`, `AppHeader.vue`, `AppTable.vue`
State Management (Vuex)	`src/store/modules`	Centralized state, actions, mutations, derived getters	`auth.js`, `portfolios.js`, `users.js`
Routing	`src/router`	Page navigation and (optional) auth gating	`index.js`
Mock Persistence	`src/mock`	In-browser data store abstraction and first-run seed	`db.js`, `seed.js`

Assets & Styling	`src/assets`, Tailwind configs	Tailwind styles and utility classes	`assets/tailwind.css`, `tailwind.config.js`

Core Technologies

- **Framework:** Vue 3
- **State:** Vuex 4
- **Routing:** Vue Router 4
- **Styling:** Tailwind CSS
- **Persistence:** Browser `localStorage` (mock DB)
- **Testing:** Jest + Vue Test Utils
- **Tooling:** Vue CLI / Vite compatible

---

3. Login & Registration Logic

Workflow Summary
This module provides lightweight authentication for demo purposes. Users are stored in `localStorage.users`. The active session is stored in `localStorage.activeUser`.

Registration Flow

1. User opens **SignUpPage.vue** and provides *name, email, password*.
2. The **auth** store checks for unique email and validates inputs.
3. On success, a new user record is added to `users`, the session is created, and the user is redirected to the portfolio list.

Login Flow

1. User opens **LoginPage.vue** and submits credentials.
2. The **auth** store validates against `users`.
3. On success, `activeUser` is set and the user is redirected to `/portfolios`.

4. On failure, an inline error is displayed.

> **Security Note:** Credentials are stored in plain text for demonstration. For production, integrate a secure backend, hash passwords (e.g., bcrypt/Argon2), and use JWT-based sessions.

---

4. Portfolio Management Logic

The Portfolio module is the core of the app, handling creation, updates, deletion, and viewing of portfolios with INR-aware outputs.

a. Add / Edit Portfolio

1. The user opens **PortfolioForm.vue** to create or edit a portfolio.
2. Inputs include: *name, description, initialInvestment, currentValue, expectedReturnRate, status (ACTIVE/CLOSED/UPCOMING), clientId*.
3. The **portfolios** store validates and persists via `db.insert`/`db.update`, which write to `localStorage.portfolios`.
4. On success, the UI confirms and routes back to `/portfolios`.

b. View Portfolio List

- **PortfolioList.vue** fetches portfolios from the **portfolios** store.
- The list supports quick filtering by **status**.
- Each row shows INR-formatted values and computed return percentage.
- Row click navigates to `/portfolios/:id`.

c. Portfolio Detail

- **PortfolioDetail.vue** displays a friendly breakdown:
    - Name and **Status**
    - **Initial Investment (₹)** and **Current Value (₹)**

- **Return (₹)** and **Return %** (derived)
- **Expected Return Rate**
- **Client** info (if `clientId` is linked)
- Optional holdings table (future enhancement)

d. Delete Portfolio

- Only permitted for existing portfolios by ID.
- The **portfolios** store removes the record and updates the list, then routes back with a confirmation alert.

---

5. Demo Data Seeding & Mock Persistence

Overview
The app uses a local, mock persistence layer to emulate backend behavior without external services. This simplifies demos and testing.

Seed Workflow

1. On first application load, **seed.js** checks for a `seeded` flag in localStorage.
2. If missing, demo users, clients, and portfolios are created and saved to:
   - `localStorage.users`
   - `localStorage.portfolios`
3. `localStorage.seeded` is set to prevent reseeding.

Persistence Layer (`db.js`)

- Minimal CRUD helpers (e.g., `getAll`, `saveAll`, `insert`, `update`, `remove`) wrap `localStorage` access.
- All Vuex actions call these helpers to keep persistence concerns centralized and testable.

Why Mock Persistence?
It keeps the project fully front-end, portable, and IDE-friendly while mirroring realistic CRUD flows that can later be swapped for real APIs.

---

6. INR Reporting & Computed Returns

Formatting

- All monetary outputs use the **Indian Rupee** format (₹).

A small helper (or inline usage) ensures consistent formatting across views:

```
new Intl.NumberFormat('en-IN', { style: 'currency', currency: 'INR' }).format(amount)
```

-

Derived Metrics

- **Return (₹)** = `currentValue - initialInvestment`
- **Return (%)** = `(Return / initialInvestment) * 100` (guard against division by zero)

These values are computed in the **Portfolio List** and **Detail** views to present clear, human-readable performance summaries.

---

7. Database Schema

This project does **not** use a server-side database. Instead, it models data via `localStorage` collections for the browser.

Collections & Shapes (sample)

`users`

```
{
  "id": 1,
  "name": "Advisor Admin",
  "email": "admin@example.com",
  "password": "demo",
  "role": "ADVISOR"
```

```
}
```

-

`portfolios`

```
{
  "id": 101,
  "name": "Alpha",
  "description": "Large-cap equity portfolio",
  "initialInvestment": 100000,
  "currentValue": 150000,
  "expectedReturnRate": 10,
  "status": "ACTIVE",
  "clientId": 201,
  "createdAt": "2025-10-10T10:00:00Z",
  "updatedAt": "2025-10-10T10:00:00Z"
}
```

- When migrating to a backend, these shapes translate directly to API DTOs and database tables (e.g., Postgres/Mongo) with minimal changes to the UI layer.

---

8. Utilities and Helpers

Utility	Purpose
Currency helper	Consistent INR formatting across list/detail views
Validators	Email uniqueness, required fields, numeric checks
ID generator	Simple incremental/UUID for new records
Date helpers	Timestamps for `createdAt` / `updatedAt`
Storage helpers (`db.js`)	Unified CRUD over localStorage

These small, pure helpers keep logic reusable and unit-test friendly.

---

9. Setup and Execution Guide

Prerequisites

- Node.js 18+
- npm (or yarn)
- Vue CLI / Vite (project is CLI-ready; Vite compatible)

Steps to Run

Install dependencies:

```
cd investment-portfolio
npm install
```

1. Start the dev server:

   ```
   npm run serve
   ```

2. Open the app at:

   http://localhost:8080/

3. **Testing**

   Run unit tests:

   ```
   npm run test:unit
   ```

   (Optional) Auto-fix lint issues:

   ```
   npm run lint -- --fix
   ```

---

10. Repository Link
📎 GitHub Repository:
https://github.com/Faiq0602/InvestmentPortfolioManagementSystem

---

11. Future Enhancements

- Replace localStorage with a real backend (Node/Express or Spring Boot)
- Secure authentication (hashed passwords, JWT, role-based access)
- Interactive charts for performance over time (Chart.js/ECharts)
- CSV/PDF export of portfolio and client summaries
- Holdings & Transactions module with imports (CSV)
- Switch to Pinia (optional) or add IndexedDB for richer offline support

---

12. Summary

The Investment Portfolio Management System is a modular, production-style **Vue 3** SPA that demonstrates clear architectural layering, business-centric workflows, and human-friendly reporting — all without a backend. It's ideal for rapid advisor demos and forms a solid foundation that can be upgraded to full API-backed persistence with minimal changes to the presentation and store layers.