Home Assignment: Full-Stack User Management App

Yuval Mandler

Tech Stack

Frontend:

- **Vue 3** – Core framework for building reactive UI
- **TypeScript** – For type safety and better tooling
- **Pinia** – State management for centralized data control
- **Vite** – Development build tool for fast and efficient bundling
- **Axios** – For clean and promise-based HTTP requests
- **Vue Toastification** – To display user-friendly toast notifications
- **Vue Router** – For navigation and route handling

Backend:

- **Node.js** + **Express** – API server and routing
- **TypeScript** – Strongly typed backend
- **MongoDB** + **Mongoose** – Flexible and scalable NoSQL database

---

Project Goals & Implementation Details

Frontend

1. **Readable and Maintainable Code Structure**
    - Components and logic are clearly separated into folders (e.g., components, views, stores, services).
    - TypeScript interfaces are used to define consistent data models.

2. **Reusable Components**
    - Props are used to make UI components (like user cards) flexible and reusable across views.
    - Pinia is used for global state management, helping to track user data, app states, and history without prop drilling.
3. **Decoupled API Calls**
    - All HTTP requests are abstracted into separate API service files (`api.ts`), keeping the view components focused on rendering and interaction logic.
4. **Smart Data Fetching**
    - `computed` properties are used to determine when to call the API for better performance and reactivity.
5. **Toasts for User Feedback**
    - `vue-toastification` is used to provide real-time user feedback on success or failure of actions.
6. **Error Handling and Logging**
    - Global error handling is connected to the toast system to inform users of issues.
    - Console logs track key API responses and errors for easier debugging.

Backend

1. **Scalable Database Architecture**
    - MongoDB is used to support horizontal scaling in the future, with Mongoose as the ORM for schema modeling.
2. **Consistent and Clean Code Structure**
    - The server code is modular, with separate folders for **routes**, **controllers**, **models**, and **helpers**, ensuring clear separation of concerns.

3. **Robust Error Handling**
    - Try-catch blocks and middleware are used to gracefully handle and forward errors to the client.
    - Errors are logged and displayed on the client via the toast system for transparency.
4. **Helper for ObjectId Validation**
    - A utility function ensures user IDs are converted into MongoDB `ObjectId` format and are validated for uniqueness before database operations.

Summary

This assignment focused on clean code architecture, reusability, scalability, and user experience. Using TypeScript on both the frontend and backend ensured a consistent and robust development workflow, while libraries like Pinia and Toastification enhanced state and UX clarity. The backend was built with future scaling and maintainability in mind.