Full-Stack Developer Intern Task

Table of Contents

1. Overview

The Exercise Program Manager is a web-based application that enables physiotherapists to create and manage customized exercise programs for patients. The application features a user-friendly interface for selecting exercises, setting parameters (e.g., sets, reps, and hold time), and organizing exercises using drag-and-drop functionality. The backend is implemented with Node.js and serves a RESTful API, utilizing a data.json file as a simulated database.

2. UI Components

The UI for this project is structured within a client folder, built using Vite + React. The src directory contains the following key folders and files, each playing a unique role in the app's functionality:

1. **Utils/**
   - Contains reusable utility functions, including customFetch.jsx, which standardizes API requests across the app for consistency and error handling.
2. **Ui /**
   - This folder includes individual components that represent various parts of the user interface. Each component is organized by its functionality within the app, ensuring that the code is modular, organized, and easy to maintain.
3. **services/**
   - Stores files such as apiExercise.js and apiPrograms.js, which define API service functions to interact with backend endpoints. By centralizing API interactions here, we achieve a more consistent and maintainable way of handling data requests and responses.

4. **features/**

   - Houses custom React hooks with file names prefixed by “use.” These hooks encapsulate reusable logic and simplify the component code. Leveraging React Query for data fetching, caching, and managing loading/error states, the folder also includes React Query DevTools for debugging and monitoring API request states directly within the browser.

5. **Global Notifications with React Hot Toast**

   - Toast notifications are configured in App.js for global use, making it easy to trigger notifications across the app. For user feedback tied to specific component actions, React Hot Toast is also imported directly when user call any api.

6. **Styling with Tailwind CSS**

The app is styled using Tailwind CSS, allowing for responsive, utility-first CSS.

3. Backend API Structure

1. data.json Structure

This JSON file simulates the database, holding categories and exercises, as well as saved programs. Example structure:


```
{
"categories": [
{
"id": 1,
"name": "Lower Body",
"exercises": [
{
"id": 1,
"name": "Squats",
"sets": 3,
"reps": 10,
"holdTime": 30,
"side": "Left"
},
{
"id": 2,
"name": "Lunges",
"sets": 3,
"reps": 12,
"holdTime": 0,
"side": "Right"
```

```
},
{
"id": 3,
"name": "Leg Press",
"sets": 4,
"reps": 10,
"holdTime": 0,
"side": "Both"
}
]
},
{
"id": 2,
"name": "Upper Body",
"exercises": [
{
"id": 4,
"name": "Push-ups",
"sets": 3,
"reps": 15,
"holdTime": 0,
"side": "Both"
},
{
"id": 5,
"name": "Pull-ups",
"sets": 3,
"reps": 8,
"holdTime": 0,
"side": "Both"
},
{
"id": 6,
"name": "Shoulder Press",
"sets": 3,
"reps": 12,
"holdTime": 0,
"side": "Both"
}
]
},
{
"id": 3,
"name": "Core",
"exercises": [
{
"id": 7,
"name": "Crunches",
"sets": 3,
"reps": 20,
```

```
"holdTime": 0,
"side": "Both"
},
{
"id": 8,
"name": "Plank",
"sets": 3,
"reps": 0,
"holdTime": 60,
"side": "Both"
},
{
"id": 9,
"name": "Russian Twists",
"sets": 3,
"reps": 15,
"holdTime": 0,
"side": "Both"
}
]
}
],
"programs": [
{
"id": 1,
"name": "Knee Rehab Program",
"exercises": [
{
"id": 221,
"name": "Lunges",
"sets": 3,
"reps": 12,
"holdTime": 0,
"side": "Right",
"daysOfWeek": [
"M",
"W",
"F"
],
"sessionsPerDay": 2,
"therapistNotes": "Patient should focus on form."
}
]
},
{
"id": 2,
"name": "Upper Body Strengthening",
"exercises": [
{
"id": 441,
```

```
"name": "Shoulder Press",
"sets": 3,
"reps": 12,
"holdTime": 0,
"side": "Right",
"daysOfWeek": [
"T",
"Th"
],
"sessionsPerDay": 1,
"therapistNotes": "Focus on controlled movements."
}
]
}
]
}
```


2. API Endpoints

1. GET /api/categories

Description: Fetches all exercise categories and associated exercises.

Response: Array of categories with exercises.

2. POST /api/programs

Description: Saves a new exercise program.

Request Body: JSON with program details, including exercises and parameters.

Response: Confirmation message with program ID.

3. GET /api/programs

Description: Retrieves all saved programs.

Response: Array of saved programs.

4. DELETE /api/programs/:exerciseId

Description: Deletes an exercise from a program by exercise ID.

Response: Success message if deletion is successful.

3. Additional Backend Components

- **Global Handler**: A centralized error handling function that manages all errors and returns structured JSON error messages. By using Global Handler, the backend maintains consistency in handling exceptions and providing meaningful error messages across all endpoints.

- **Async Middleware (catchAsync)**: Wraps asynchronous route handlers, allowing concise handling of async/await calls without repetitive try-catch blocks.

- **Modularized Service Layer**: The API logic is separated into services, making it easy to handle business logic independently of route definitions. This structure keeps the API codebase organized and easily maintainable.

4. Project Setup Instructions

Prerequisites

- Node.js (v14+)
- npm (v6+)
- Vite + React

Installation Steps

1. Backend Setup
    - Navigate to the server folder.
    - Install dependencies:
        - npm install
        - npm start
    - Server runs at http://localhost:8000

2. Frontend Setup
    - Navigate to the client folder.
    - Install dependencies:
        - npm install
        - npm run dev
    - Server runs at http://localhost:5173

3. Testing the API
    - Use Postman or Curl for API endpoint testing.
    - Ensure the server is running on the correct port.