Frontend Development with React.js

Project Documentation format

1. Introduction

Project Title: **Fitflex Your Personal Companion**

Team Leader:

S.Hemalatha - hemalathashemalatha9@gmail.com

- **Team Members:**
- **V.Yuvarani - vyuvaranivyuva@gmail.com**
- **P.Malathi - malathimalathi4041@gmail.com**
- **P.Deepika - deepikapraba21@gmail.com**

2.Project Overview

- **Purpose:**

  FitFlex is a revolutionary fitness app designed to transform your workout experience. It offers an intuitive interface, dynamic search, and a vast library of exercises for all fitness levels. Join FitFlex to embark on a personalized fitness journey and achieve your wellness goals.

- **Features:**

  ✓ Exercises from Fitness API: Access a diverse array of exercises from reputable fitness APIs, covering a broad spectrum of workout categories and catering to various fitness goals.

  ✓ Visual Exercise_Exploration: Engage with workout routines through curated image galleries, allowing users to explore different exercise categories and discover new fitness challenges visually.

✓ **Intuitive and User-Friendly Design: Navigate the app seamlessly with a clean, modern interface designed for optimal user experience and clear exercise selection.**

✓ **Advanced Search Feature: Easily find specific exercises or workout plans through a powerful search feature, enhancing the app's usability for users with varied fitness preferences.**

3.Architecture

1.Component Structure: Component Structure A well organized React application typically follows a modular component structure. Components are divided based on their functionality and reusability:

Presentational Components: These are primarily concerned with how things look. They receive data via props and render UI elements accordingly.

Container Components: These handle the application's logic and state management. They fetch data, manage state, and pass necessary information to presentational components.

Organizing components by feature or route can enhance scalability and maintainability. For instance, grouping all components related to a specific feature within a dedicated directory ensures a clear and intuitive structure.

2.State Management:Managing state efficiently is crucial for the performance and scalability of a React application. Several approaches are commonly used

Local State: Utilizing React's built-in useState and useReducer hooks to manage state within components. This approach is suitable for state that doesn't need to be shared across multiple components.

Context API: For state that needs to be accessible by multiple components, React's Context API provides a way to pass data through the component tree without having to pass props down manually at every level.

Redux: A popular library for managing global state, especially in larger applications. Redux centralizes the application's state and logic, making state transitions predictable and easier to debug.

Choosing the right state management approach depends on the application's complexity and specific requirements. For instance, while local state and Context API might suffice for smaller applications, larger applications might benefit from the predictability and structure provided by Redux.

3.Routing:React doesn't include built-in routing capabilities, so developers often use external libraries like react-router-dom to handle navigation.

A clean routing architecture involves:

Defining Routes: Mapping URL paths to specific components that should be rendered.

Nested Routes: Handling complex layouts by nesting routes, allowing for shared layouts or components across different sections of the application.

Protected Routes: **Implementing access control by restricting certain routes to authenticated users.**

Organizing routes in a modular way ensures that the routing logic remains clean and maintainable.

By adhering to these best practices, you can build a React application that is both scalable and maintainable.

4.Setup Instructions

- **Prerequisites**:

  ✓ **Node.js and npm:** **Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. It provides a scalable and efficient platform for building network applications. Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.**

  ● Download: https://nodejs.org/en/download/

  ● Installation instructions: https://nodejs.org/en/download/package-manager/

  ✓ **React.js:** **React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. Install React.js, a JavaScript library for building user interfaces.**

  ● Create a new React app: npm create-react-app my-react-app Replace my-react-app with your preferred project name.

  ● Navigate to the project directory: cd my-react-app

• Running the React App: With the React app created, you can now start the development server and see your React application in action.

• Start the development server: npm start This command launches the development server, and you can access your React app at http://localhost:3000 in your web browser.

✓ HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

✓ Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

• Git: Download and installation instructions can be found at: https://git-scm.com/downloads

✓ Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• Visual Studio Code: Download from https://code.visualstudio.com/download

• Sublime Text: Download from https://www.sublimetext.com/download

• WebStorm: Download from https://www.jetbrains.com/webstorm/download To get the Application project from drive: Follow below steps:

✓ Get the code: • Download the code from the drive link given below: https://drive.google.com/drive/folders/14f9eBQ5W7VrLdPhP2W6PzOU_HCy8UMex?usp=sharing Install Dependencies:

• Navigate into the cloned repository directory and install libraries: cd fitness-app-react npm install

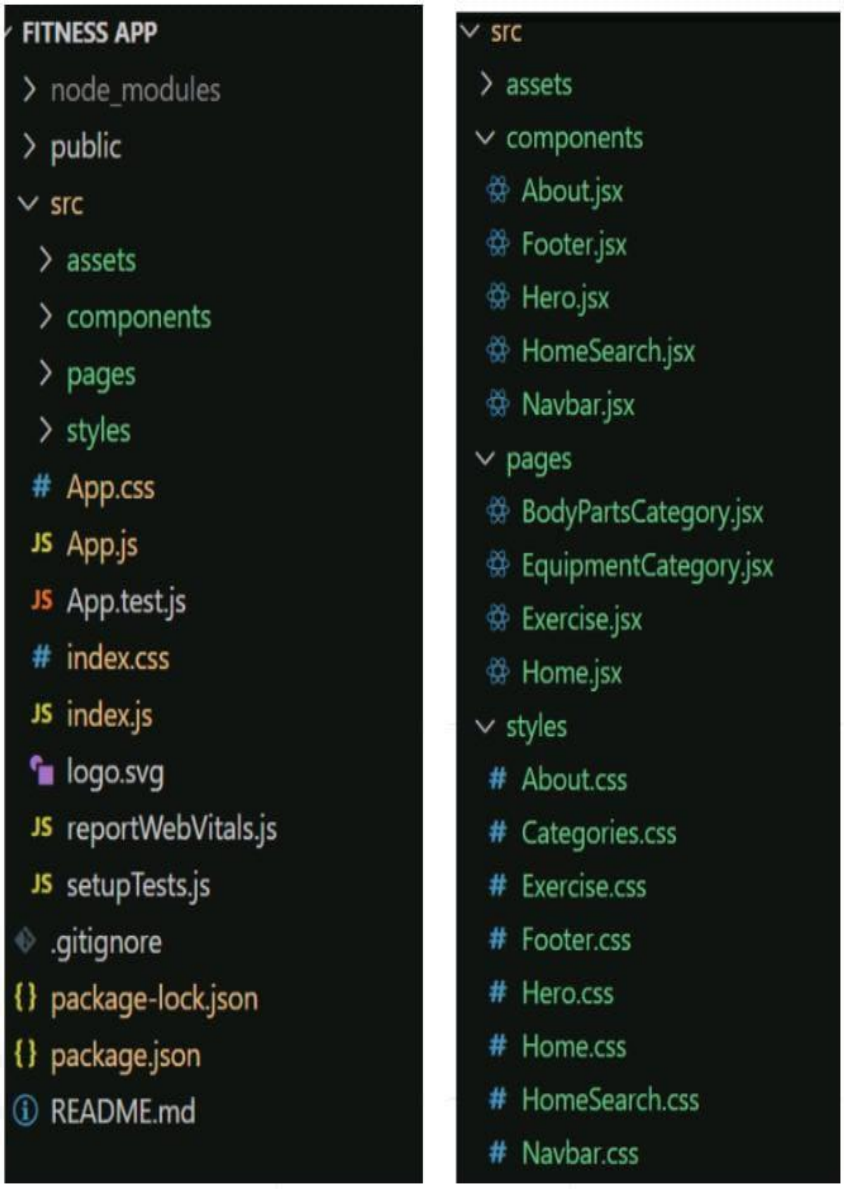
✓ Start the Development Server:

• To start the development server, execute the following command: npm start Access the App:

• Open your web browser and navigate to http://localhost:3000.

• You should see the application's homepage, indicating that the installation and setup were successful. You have successfully installed and set up the application on your local machine. You can now proceed with further customization, development, and testing as needed.

5.Folder Structure



- **In this project, we've split the files into 3 major folders, Components, Pages and Styles. In the pages folder, we store the files that acts as pages at different URLs in the application. The components folder stores all the files, that returns the small**

components in the application. All the styling css files will be stored in the styles folder.

6.Running the Application

Project Flow:

- **Project demo: Before starting to work on this project, let's see the demo.**

Demo link:https://drive.google.com/file/d/1mMqMb41RtroiFbUQ-1ZfeYfWJZ6okSNb/view?usp=sharing Use the code in: https://drive.google.com/drive/folders/14f9eBQ5W7VrLdPhP2W6PzOU_HCy8UMex?usp=sharing Milestone

1: Project setup and configuration.

● Installation of required tools: To build the FitFlex app, we'll need a developer's toolkit. We'll leverage React.js for the interactive interface, React Router Dom for seamless navigation, and Axios to fetch fitness data. To style the app, we'll choose either Bootstrap or Tailwind CSS for pre-built components and a sleek look. Open the project folder to install necessary tools. In this project, we use: o React Js o React Router Dom o React Icons o Bootstrap/tailwind css o Axios

● For further reference, use the following resources o https://react.dev/learn/installation o https://react-bootstrap-v4.netlify.app/getting-started/introduction/ o https://axios-http.com/docs/intro o https://reactrouter.com/en/main/start/tutorial Milestone

2: Project Development

❖ Setup the Routing paths Setup the clear routing paths to access various files in the application.

❖ Develop the Navbar and Hero components

❖ Code the popular search/categories components and fetch the categories from rapid Api.

❖ Additionally, we can add the component to subscribe for the newsletter and the footer.

❖ Now, develop the category page to display various exercises under the category.

❖ Finally, code the exercise page, where the instructions, other details along with related videos from the YouTube will be displayed. Important Code snips: Fetching available Equipment list & Body parts list From the Rapid API hub, we fetch available equipment and list of body parts with an API request.

7.Component Documentation

Key Components: Workout & Activity Tracking

WorkoutTracker: Logs exercises, sets, reps, and duration (Props: workoutData, onSave).

StepCounter: Tracks daily step count and activity levels (Props: stepData).

ActivityHistory: Displays past workouts and activities (Props: historyData).

8.State Management

Global State:

- **FitFix uses React Context API for global state management. The global state stores essential user-related data such as:**
- **User Authentication Data: Manages user login state, authentication tokens, and user details.**
- **Workout Data: Tracks exercises, routines, and progress across the app.**
- **Nutrition Data: Stores and updates meal plans or calorie intake details.**
- **Theme Preferences: Manages light/dark mode if implemented.**

State Flow

1.Context Providers: Wrap the application inside a provider component (e.g., UserContext, WorkoutContext).

2. State Access: Any component needing global data can access it via useContext().

3. Actions & Reducers: For complex state updates (e.g., workouts, authentication), state reducers may be used.

9.User Interface

<u>API Key:</u> Replace 'place your api key' with a placeholder mentioning that the user needs to replace it with their own RapidAPI key. You can mention how to acquire an API key from RapidAPI. bodyPartsOptions and equipmentOptions: These variables hold configuration options for fetching data from the RapidAPI exercise database.

● method: The HTTP method used in the request. In this case, it's set to GET as the code is fetching data from the API.

● url: The URL of the API endpoint to fetch data from. Here, it's set to https://exercisedb.p.rapidapi.com/exercises/bodyPartList for fetching a list of body parts and https://exercisedb.p.rapidapi.com/exercises/equipmentList for fetching a list of equipment.

● headers: This section contains headers required for making the API request. Here it includes the X-RapidAPI-Key header to provide your API key and the X-RapidAPI-Host header specifying the host of the API. fetchData function: This function is responsible for fetching data from the API. It makes use of async/await syntax to handle asynchronous operations. First it fetches data for body parts using axios.request(bodyPartsOptions). Then it stores the fetched data in the bodyParts state variable using setBodyParts. Similarly, it fetches data for equipment using axios.request(equipmentOptions). Then it stores the fetched data in the equipment state variable using

setEquipment. In case of any errors during the API request, the catch block logs the error to the console using console.error. useEffect Hook: The useEffect hook is used to call the fetchData function whenever the component mounts. This ensures that the data is fetched as soon as the component loads. Overall, the code snippet demonstrates how to fetch data from a RapidAPI exercise database using JavaScript's Axios library. Fetching exercises under particular category To fetch the exercises under a particular category, we use the below code. It defines a function called fetchData that fetches data from an exercise database API. Here's a breakdown of the code: const options = {...}: This line creates a constant variable named options and assigns it an object literal. The object literal contains properties that configure the API request, including:

● method: Set to 'GET', indicating that the API request is a GET request to retrieve data from the server.

● url: Set to https://exercisedb.p.rapidapi.com/exercises/equipment/${id}, which is the URL of the API endpoint for fetching exercise equipment data. The ${id} placeholder will likely be replaced with a specific equipment ID when the function is called.

● params: An object literal with a property limit: '50'. This specifies that you want to retrieve a maximum of 50 exercise equipment results.

● headers: An object literal containing two headers required for making the API request:

● 'X-RapidAPI-Key': Your RapidAPI key, which is used for authentication. You should replace 'your api key' with a placeholder instructing users to replace it with their own API key.

● 'X-RapidAPI-Host': The host of the API, which is 'exercisedb.p.rapidapi.com' in this case. const fetchData = async (id)

=> {...}: This line defines an asynchronous function named fetchData that takes an id parameter. This id parameter is likely used to specify the equipment ID for which data needs to be fetched from the API. try...catch block:

● The try...catch block is used to handle the API request.

● The try block contains the code that attempts to fetch data from the API using axios.request(options).

● The await keyword is used before axios.request(options) because the function is asynchronous and waits for the API request to complete before proceeding.

● If the API request is successful, the response data is stored in the response constant variable.

● The console.log(response.data) line logs the fetched data to the console.

● The .then method (not shown in the image) is likely used to process the fetched data after a successful API request.

● The catch block handles any errors that might occur during the API request. If there's an error, it's logged to the console using console.error(error). Fetching Exercise details Now, with the help of the Exercise ID, we fetch the details of a particular exercise with API request. The code snippet demonstrates how to fetch exercise data from an exercise database API using JavaScript's fetch API. Here's a breakdown of the code: API Endpoint and Key:

● Replace 'https://example.com/exercise' with the actual URL of the API endpoint you want to use.

● Replace 'YOUR_API_KEY' with a placeholder instructing users to replace it with their own API key obtained from the API provider. async function: The code defines an asynchronous function named fetchData that likely takes an id parameter as input. This id

parameter might be used to specify the ID of a particular exercise or category of exercises to fetch. fetch request: Inside the fetchData function, the fetch API is used to make an HTTP GET request to the API endpoint. The function creates a fetch request with the following details:

● Method: GET (to retrieve data from the server)

● URL: The API endpoint URL where exercise data resides. Handling the Response:

● The then method is used to handle the response from the API request. If the request is successful (i.e., status code is 200), the response is converted to JSON format using response.json().

● The .then method then likely processes the fetched exercise data, which might involve storing it in a state variable or using it to populate a user interface. Error Handling: The .catch method is used to handle any errors that might occur during the API request. If there's an error, it's logged to the console using console.error. Fetching related videos from YouTube Now, with the API, we also fetch the videos related to a particular exercise with code given below. The code snippet shows a function called fetchRelatedVideos that fetches data from YouTube using the RapidAPI service. Here's a breakdown of the code: fetchRelatedVideos function: This function takes a name parameter as input, which is likely the name of a video or a search query. API configuration: The code creates a constant variable named options and assigns it an object literal containing configuration details for the API request:

● method: Set to 'GET', indicating a GET request to retrieve data from the server.

● url: Set to 'https://youtube-search-and-download.p.rapidapi.com/search', which is the base URL of the RapidAPI endpoint for YouTube search.

● params: An object literal containing parameters for the YouTube search query:

● query: Set to \${name}, a template literal that likely gets replaced with the actual name argument passed to the function at runtime. This specifies the search query for YouTube videos.

● Other parameters like hl (language), sort (sorting criteria), and type (video type) are included but their values are not shown in the snippet.

● headers: An object literal containing headers required for making the API request:

● 'X-RapidAPI-Key': Your RapidAPI key, which is used for authentication. You should replace 'YOUR_API_KEY' with a placeholder instructing users to replace it with their own API key.

● 'X-RapidAPI-Host': The host of the API, 'youtube-search-and-download.p.rapidapi.com' in this case. Fetching Data (try...catch block):

● The try...catch block is used to handle the API request. which is

● The try block contains the code that attempts to fetch data from the API using axios.request(options).

● axios is an external JavaScript library for making HTTP requests. If you don't already use Axios in your project, you'll need to install it using a package manager like npm or yarn.

● The .then method (not shown in the code snippet) is likely used to process the fetched data after a successful API request.

● The catch block handles any errors that might occur during the API request. If there's an error, it's logged to the console using console.error(error).

10.Styling

About.css

```
.about-container{
    height: 90vh;
    display: grid;
    align-items: center;
    grid-template-columns: 50% 50%;
}
.about-image{
  display: flex;
    justify-content: center;
    align-items: center
}
.about-image img{
    height: 70vh;
    width: 28vw;
    box-shadow: 10px 10px rgb(71, 203, 255);
```

```
    border-radius: 0.3rem;
}
.about-text{
    width: 90%;
}
.about-text span{
    display: flex;
    align-items: center;
    gap: 10px;
}
.about-text span .about-line{
    height: 2px;
    width: 50px;
    background-color: rgb(71, 181, 255);
    margin: 0;
}

.about-text span h5{
    margin: 0;
    font-size: 1.4rem;
    font-weight: 400;
```

```
    color: rgb(71, 206, 255),
}
.about-text h3{
    font-size: 2.5rem;
    margin: 0;
    color: rgb(42, 42, 42);
}
.about-text p{
    font-size: 1.2rem;
    color: rgb(62, 62, 62);
}
```

Categories.css

```
.category-exercises-page{
    min-height: 80vh;
    padding-top: 10vh;
    padding-bottom: 10vh;
}

.category-exercises-page h1{
```

```
    font-size: 2rem;
    font-weight: 400;
    margin-bottom: 2rem;
    margin-left: 3vw;
}

.category-exercises-page h1 span{
    font-weight: 600;
}

.exercises{
    display: grid;
    grid-template-columns: 22% 22% 22% 22%;
    justify-content: space-evenly;
    row-gap: 3vh;
}

.exercise{
    border: 1px solid #e0e0e0;
    box-shadow: rgba(99, 99, 99, 0.2) 0px 2px 8px 0px;
    padding: 2vh 2vh 4vh 2vh;
```

```
    border-radius: 0.6rem;
    cursor: pointer;
    background-color: #020e13de;
}

.exercise:hover{
    transform: scale(1.05);
    transition: all 0.2s ease-in-out;
}

.exercise img{
    width: 100%;
    object-fit: cover;
    border: 1px solid #e0e0e0;
    border-radius: 0.6rem;
}

.exercise h3{
    margin: 0;
    font-weight: 400;
    width: 96%;
```

```
    margin-left: 2%;
    color: rgb(229, 232, 235);
}

.exercise ul{
    display: flex;
    flex-wrap: wrap;
    gap: 10px;
    padding: 0;
    margin: 0;
}

.exercise ul li{
    list-style: none;
    border: 1px solid rgb(254, 191, 96);
    color: rgb(254, 191, 96);
    padding: 5px 10px;
    font-size: 0.7rem;
    border-radius: 2rem;
}
```

11.Testing

Testing Strategy for FitFix App

- **To ensure the reliability, functionality, and performance of the FitFix app, a structured testing strategy is implemented. This includes unit testing, integration testing, and end-to-end (E2E) testing using tools like Jest, React Testing Library, and Cypress.**

1. Unit Testing

- **Objective: Verify that individual components function as expected.**
- **Tools: Jest, React Testing Library**
- **Approach:**

Test each React component in isolation.

Validate component rendering, state changes, and event handling.

Mock API calls and test asynchronous functions using Jest mocks.

2. Integration Testing

- **Objective: Verify interactions between components and API services.**
- **Tools: Jest, React Testing Library**
- **Approach:**

Test how multiple components interact within a feature.

Mock API requests using msw (Mock Service Worker).

Ensure UI updates correctly based on API responses.

3. End-to-End (E2E) Testing

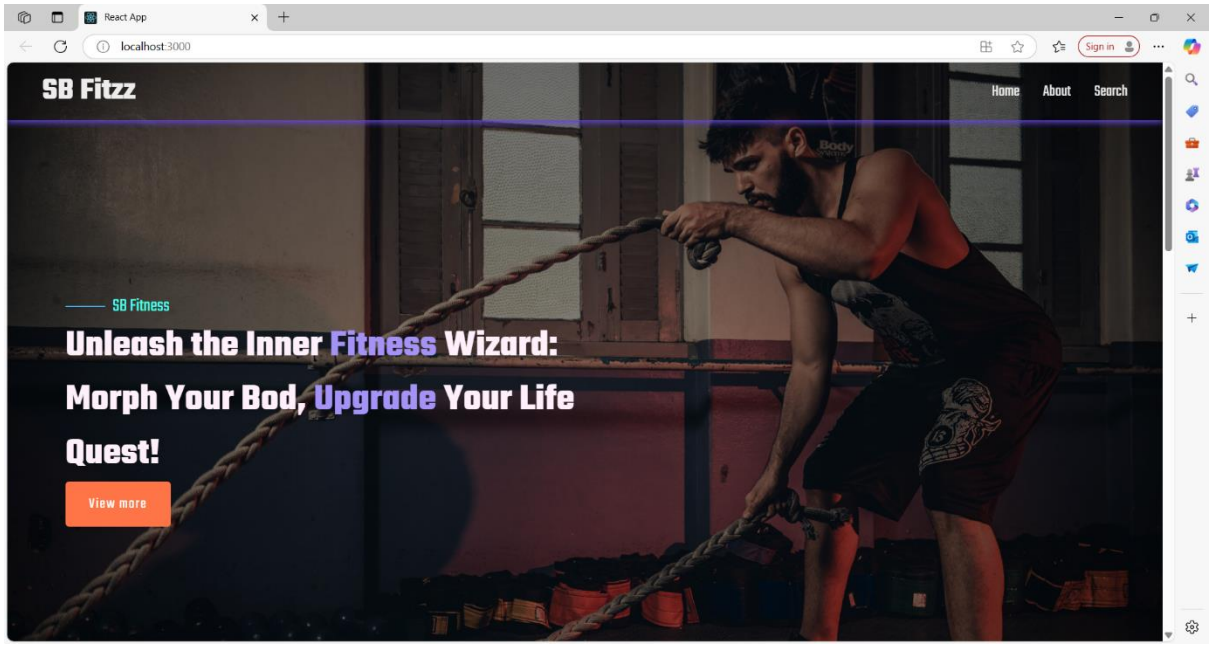
- **Objective: Test the entire user journey from UI interactions to backend responses.**
- **Tools: Cypress**
- **Approach:**

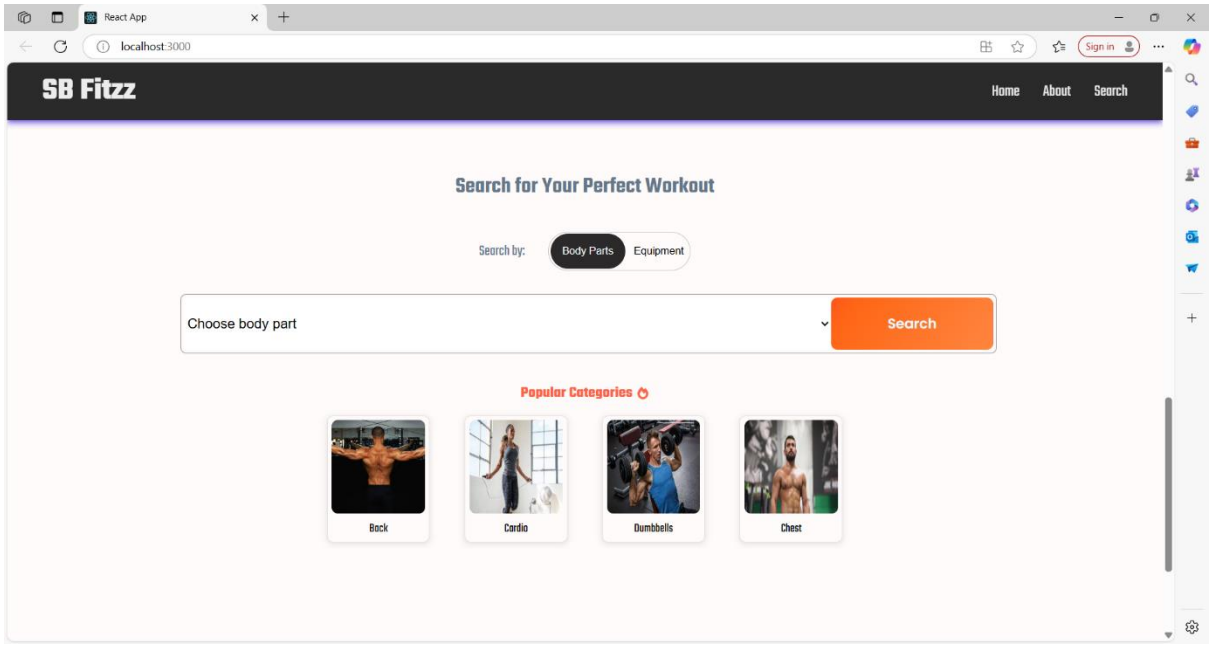
Simulate real user interactions like form submissions and navigation.

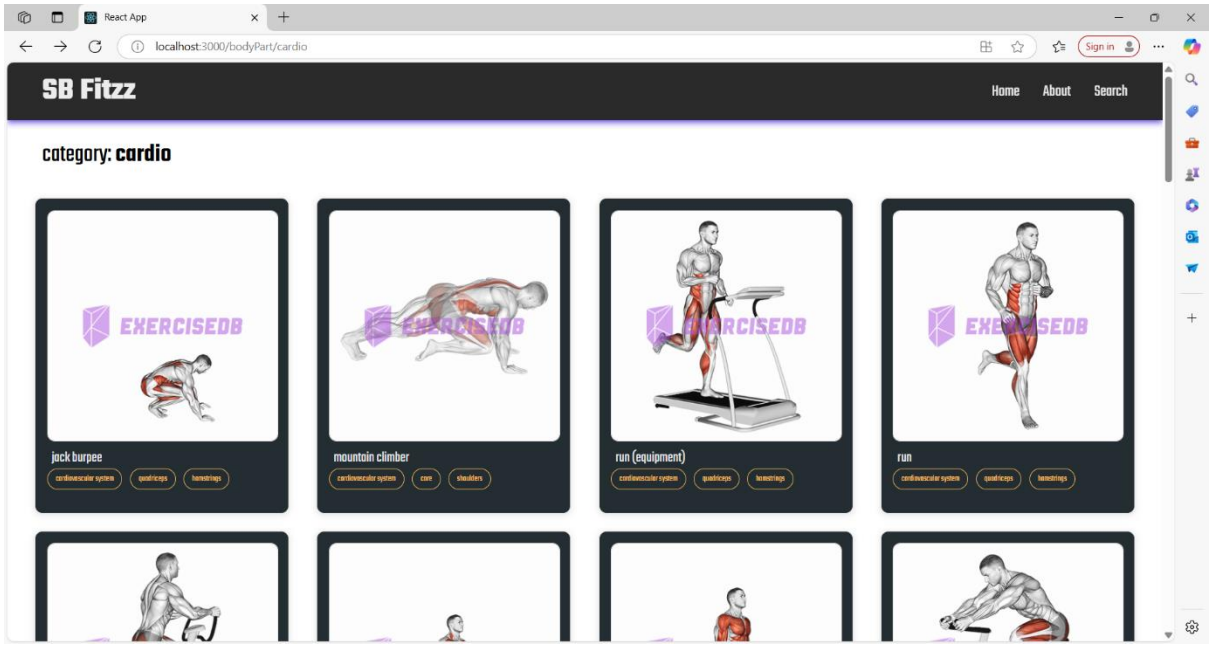
Validate that data is stored correctly and retrieved accurately.

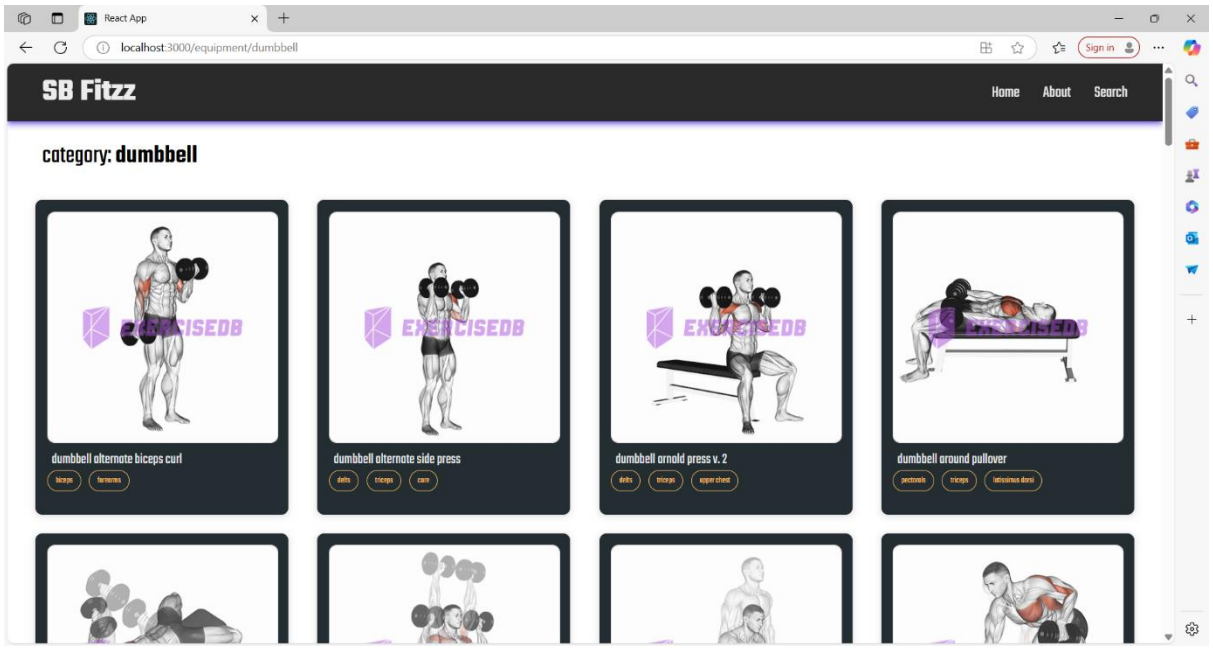
Ensure error handling works properly.

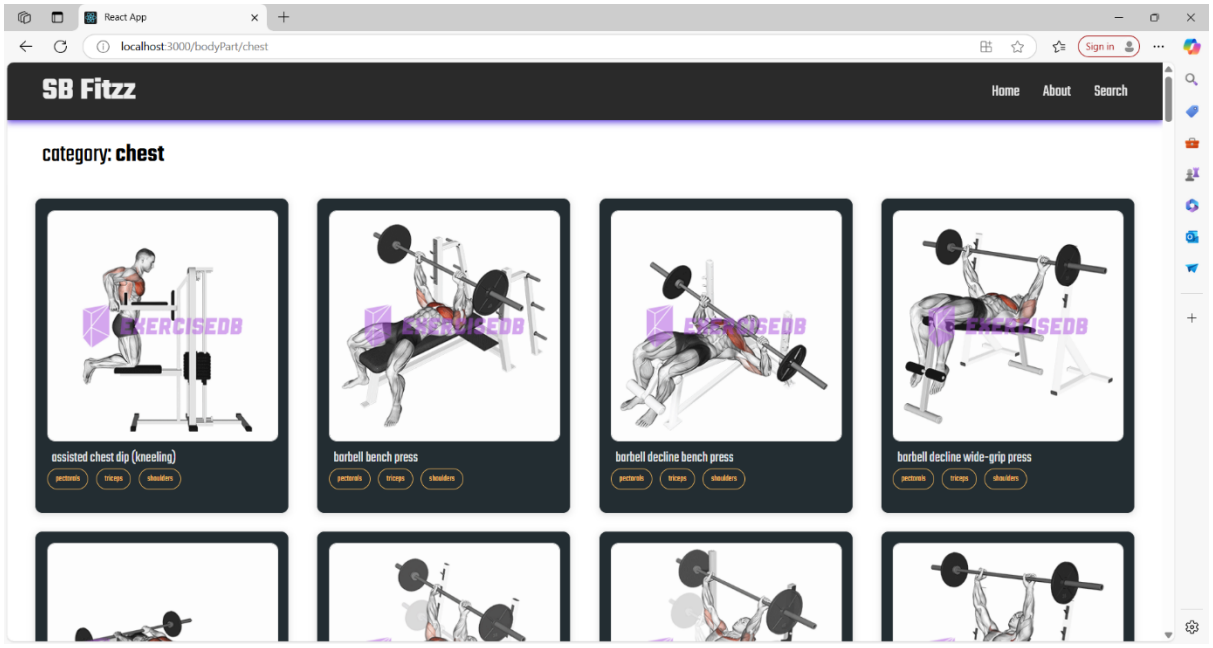
12.Screenshot or Demo

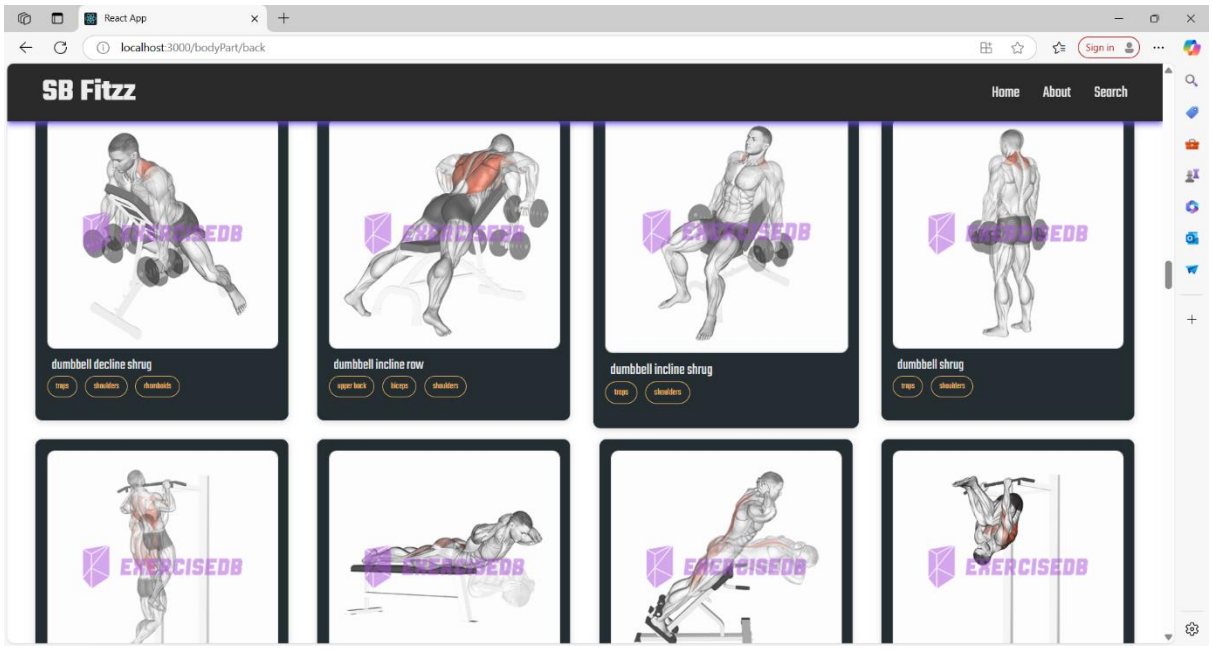

React App
localhost:3000
Sign in
SB Fitzz
Home
About
Search
SB Fitness
Unleash the Inner Fitness Wizard:
Morph Your Bod, Upgrade Your Life
Quest!
View more


React App
localhost:3000
Sign in
SB Fitzz
Home
About
Search
Search for Your Perfect Workout
Search by:
Body Parts
Equipment
Choose body part
Search
Popular Categories
Back
Cardio
Dumbbells
Chest


React App
localhost:3000/bodyPart/cardio
Sign in
SB Fitzz
Home
About
Search
category: cardio
jack burpee
mountain climber
run (equipment)
run


React App
localhost:3000/equipment/dumbbell
Sign in
SB Fitzz
Home
About
Search
category: dumbbell
dumbbell alternate biceps curl
dumbbell alternate side press
dumbbell arnold press v. 2
dumbbell around pullover





13. Known Issue

1. UI/UX Issues

Issue: Some buttons do not respond on the first click.

Cause: Possible event listener delay in React components.

Workaround: Click again or refresh the page.

2. API & Backend Issues

Issue: Workout data sometimes does not sync properly.

Cause: Possible delay in Node.js backend processing or database queries.

Workaround: Refresh the app or check network connectivity.

3. Performance Issues

Issue: Slow loading times on the dashboard page.

Cause: Large API responses and inefficient state management.

Workaround: Optimize API requests and implement lazy loading.

4. Compatibility Issues

Issue: App does not work properly on older mobile devices.

Cause: Some modern JavaScript features are not supported on older browsers.

Workaround: Use a modern browser like Chrome or Firefox.

5. Data Persistence Issues

Issue: User progress resets after logging out in some cases.

Cause: Local storage not updating correctly or session issues.

Workaround: Ensure the database saves progress before logging out.

14.Future Enhancements

1. New Features & Functionalities

✅ **AI-Powered Workout Recommendations**

Implement machine learning algorithms to provide personalized workout plans based on user progress, preferences, and fitness goals.

✅ **Diet & Nutrition Tracking**

Introduce a feature that allows users to log their meals and track macronutrients (proteins, carbs, fats) for a comprehensive fitness experience.

✅ **Wearable Device Integration**

Sync with fitness wearables (e.g., Apple Watch, Fitbit, Garmin) to track heart rate, calories burned, and steps in real-time.

✅ **Social & Community Features**

Allow users to connect with friends, share achievements, and join fitness challenges.

Create a leaderboard for motivation and competition.

✅ **Gamification Elements**

Introduce rewards, badges, and achievement levels to keep users engaged and motivated.

✅ **Voice Command & AI Assistant**

Enable hands-free interaction using voice commands for starting workouts, setting timers, and tracking progress.

✅ **Water & Sleep Tracking**

Add features to log water intake and sleep patterns to provide holistic health tracking.

2. UI/UX Enhancements

✅ **Dark Mode & Theming**

Allow users to switch between dark and light themes for a better visual experience.

✅ **Enhanced Animations & Transitions**

Smooth page transitions, animated progress bars, and interactive buttons for a more engaging UI.

✅ **Customizable Dashboard**

Let users personalize their dashboard with widgets (calorie tracker, step counter, workout stats).

✅ **AR-Based Workout Demonstrations**

Augmented Reality (AR) feature to guide users through exercises with 3D models and real-time form correction.

3. Backend & Performance Improvements

✅ **Real-Time Data Syncing**

Improve synchronization between devices, so users can access their data from any device seamlessly.

✅ **Offline Mode**

Allow users to log workouts and view past data without an internet connection, syncing when online.

✅ **Push Notifications & Reminders**

Smart reminders based on user activity, like workout reminders or hydration alerts.

✅ **Enhanced Security & Data Privacy**

Implement OAuth and biometric authentication for secure logins.

Strengthen data encryption to protect user information.

✅ **Scalability & Cloud Storage**

Upgrade backend to handle a larger user base efficiently using cloud solutions like AWS or Firebase.