

# Frontend Development with React.js

## Project Documentation format

### 1. Introduction

- **Project Title:** Fit Flex: Your Personal Fitness Companion
- **Team Leader:** Karthigadevi R
- **Team Members:**
  1. KeerthanaPriyaa A
  2. Kowshika S
  3. Gayathri S
  4. Akshaya Devi S

### 2. Project Overview

- **Purpose:**

Fit Flex is a revolutionary fitness app designed to transform the workout experience by providing an intuitive interface, dynamic search, and a vast library of exercises for all fitness levels. The goal is to create an accessible platform for individuals passionate about fitness, exercise, and holistic well-being. Fit Flex aims to reshape how users engage with fitness by offering personalized workout plans, fostering a supportive community, and integrating advanced search and recommendation features

- **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

- **Component Structure:**

The project is structured into three major folders:

**Components:** Contains reusable UI elements like Footer, Search Bar, and Exercise Cards.

**Pages:** Stores files that act as pages at different URLs in the application (e.g., Home, Exercise Details, and Categories).

**Styles:** Contains CSS files for styling the application.

- **State Management:**

Fit Flex uses **React's use State and use Effect hooks** for managing local state.

- API data (like exercise categories and details) is fetched using **Axios** and stored in state variables.
- The **use Effect** hook ensures data is fetched on component mount.
- For global state management, **Context API** or **Redux** can be integrated if needed.

- **Routing:**

The application uses **React Router Dom** for navigation.

- Users can navigate between different pages seamlessly.
- `/` → Home Page
- `/categories` → Displays different exercise categories
- `/exercise/:id` → Detailed view of a specific exercise
- The **Route Parameters** help fetch and display specific exercise details dynamically.

### 4. Setup Instructions

- **Prerequisite:**

Here are the key prerequisites for developing a frontend application using React.js:

✓ **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/>

✓ **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: `npx create-react-app my-react-app`

- 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.

✓ **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.

- Visual Studio Code: [Download from https://code.visualstudio.com/download](https://code.visualstudio.com/download)

- **Installation**

- 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
```

- 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.

## 5. Folder Structure

- **Client:**

The FitFlex React application is organized into three major folders:

```
/fitness-app-react
├── /src
│   ├── /components          # Reusable UI elements
│   │   ├── Navbar.js
│   │   ├── Footer.js
│   │   ├── SearchBar.js
│   │   ├── ExerciseCard.js
│   │   └── CategoryList.js
│   ├── /pages               # Main pages
│   │   ├── Home.js
│   │   ├── Categories.js
│   │   └── ExerciseDetails.js
│   ├── /styles              # CSS styles
│   │   ├── global.css
│   │   ├── navbar.css
│   │   └── footer.css
│   ├── /assets              # Images, icons, fonts
│   ├── /utils               # Helper functions and API calls
│   ├── App.js               # Main application component
│   ├── index.js             # Entry point
│   └── package.json         # Project dependencies
├── .gitignore
└── README.md
```

- **Utilities:**

The `/utils` folder contains helper functions, utility classes, and custom hooks for optimized functionality.

✓ **API Requests (`api.js`)**

Handles all API calls using Axios:

```
1. import axios from "axios";
2. const API_URL = "https://exercisedb.p.rapidapi.com/exercises";
3.
4. export const fetchExercises = async (category) => {
5.   try {
6.     const response = await
       axios.get(`${API_URL}/category/${category}`, {
7.       headers: {
8.         "X-RapidAPI-Key": process.env.REACT_APP_RAPIDAPI_KEY,
9.         "X-RapidAPI-Host": "exercisedb.p.rapidapi.com",
10.      },
11.    });
12.    return response.data;
13.  } catch (error) {
```

```
14.         console.error("Error fetching exercises:", error);
15.     }
```

### ✓ Custom Hook (useFetch.js)

A reusable hook for fetching API data:

```
2. import { useState, useEffect } from "react";
3. import axios from "axios";
4.
5. const useFetch = (url, options) => {
6.   const [data, setData] = useState(null);
7.   const [loading, setLoading] = useState(true);
8.   const [error, setError] = useState(null);
9.
10.  useEffect(() => {
11.    const fetchData = async () => {
12.      try {
13.        const response = await axios.get(url, options);
14.        setData(response.data);
15.      } catch (err) {
16.        setError(err);
17.      } finally {
18.        setLoading(false);
19.      }
20.    };
21.    fetchData();
22.  }, [url]);
23.
24.  return { data, loading, error };
25. };
26.
27. export default useFetch;
```

## 6. Running the Application

To start the **FitFlex** frontend server locally, follow these steps:

### 1. Navigate to the project directory:

```
cd fitness-app-react
```

### 2. Install dependencies:

```
npm install
```

• **Frontend:** `npm start`

### Access the application:

Open your browser and go to <http://localhost:3000>.

If everything is set up correctly, the FitFlex homepage should load.

If using **Vite.js**, use the command:

npm run dev

## 7. Component Documentation

### ○ Key Components:

Component	Purpose	Props
<b>Navbar.js</b>	Displays the navigation menu across all pages.	<code>links</code> (array) – List of navigation links.
<b>Footer.js</b>	Shows footer content with links and social media.	None
<b>SearchBar.js</b>	Allows users to search for exercises.	<code>onSearch</code> (function) – Callback function to handle search input.
<b>ExerciseCard.js</b>	Displays a single exercise with its details.	<code>exercise</code> (object) – Contains exercise name, image, and description.
<b>CategoryList.js</b>	Renders a list of workout categories.	<code>categories</code> (array) – List of exercise categories.
<b>ExerciseDetails.js</b>	Shows detailed information about a selected exercise.	<code>exerciseId</code> (string) – ID of the selected exercise.

### ○ Reusable Components:

These components are used multiple times across different parts of the application.

#### ✓ **Button.js**

A customizable button component.

- `label` (string): Button text.
- `onClick` (function): Click event handler.
- `className` (string): Additional styling classes.

#### ✓ **Loader.js**

Displays a loading spinner while fetching data.

#### ✓ **Modal.js**

A generic modal popup component.

- `isOpen` (boolean): Controls modal visibility.
- `onClose` (function): Closes the modal.
- `children` (JSX): Content inside the modal.

## 8. State Management

- **Global State:**

For **global state**, **React Context API** or **Redux** could be used depending on the scale of the app. In the current implementation, the **Context API** is primarily used to manage and share global state between components that do not have a direct parent-child relationship.

- **User Authentication Context:** Stores user information such as login status or user preferences across different pages.
- **Exercise Data Context:** Shares fetched exercise categories, workout lists, and other data globally so that components like Search, Categories, and Exercise Cards can access the information without re-fetching it.
- **Settings Context:** Handles global settings, such as dark/light mode or language preferences.

- **Local State:**

Local state is used within individual components to manage the UI's internal behavior. This state is not shared globally but instead scoped to the component that defines it.

- **Search Input State:** Manages the current value of the search bar for filtering exercises.
- **Loading State:** Tracks whether the app is currently fetching data or whether an error has occurred during data retrieval.
- **Modal Visibility:** Determines whether a modal is open or closed.

## 9. User Interface

The **FitFlex** application provides an intuitive and engaging **User Interface (UI)** that prioritizes ease of use, clarity, and visual appeal. The interface is designed to provide users with a seamless experience when exploring exercises, managing workout routines, or searching for new fitness challenges.

### Key UI Elements:

- **Navigation Bar (Navbar):**  
A responsive navbar that includes links to key sections of the app (e.g., Home, Categories, Workout Plans).
- **Search Bar:**  
A prominent search bar allowing users to easily find exercises by name, category, or difficulty level.
- **Exercise Cards:**  
Cards displaying exercise details such as name, type, and an image to help users visually explore and engage with workouts.

- **Category Filters:**  
Categories such as strength training, cardio, flexibility, etc., are organized in filterable buttons or tabs for easy navigation.
- **Modal Windows:**  
Modals are used for showing detailed exercise information or confirming user actions (e.g., saving or removing an exercise).
- **Loading Indicators:**  
An elegant spinner or progress bar to indicate loading or data-fetching states.

## 10. Styling

- **CSS Frameworks/Libraries:**

FitFlex uses a combination of **modern CSS frameworks and libraries** to ensure a clean, responsive, and visually appealing design.

1. **Tailwind CSS:**  
Tailwind CSS is used to style components with utility-first classes. This allows for faster design iterations, easier customization, and better maintainability..
2. **Bootstrap (optional):**  
If needed, **Bootstrap** can be used for quick prototyping or as a fallback. It provides a set of pre-styled UI components like buttons, cards, and forms.
3. **CSS Grid/Flexbox:**  
CSS Grid and Flexbox are used to create flexible and adaptive layouts. This ensures that components like cards, buttons, and text align appropriately across different screen sizes.

- **Theming:**

FitFlex provides theming capabilities to allow users to toggle between light and dark modes.

1. **Light Mode:**
  - A clean, bright interface with a light background color and dark text for high contrast and easy readability.
2. **Dark Mode:**
  - A darker interface, primarily using shades of gray and black for the background, with light-colored text. This reduces eye strain in low-light environments.
3. **Customizable Themes:**  
Users can adjust the primary color scheme, font sizes, and interface elements according to their preferences, ensuring accessibility and personalization.



## 11. Testing

- **Testing Strategy:**

The **FitFlex** application employs a comprehensive testing strategy to ensure reliability, maintainability, and a smooth user experience. The testing covers the following levels:

1. **Unit Testing:**

Individual components (e.g., buttons, cards, modals) and helper functions are tested to ensure they work as expected. **Jest** and **React Testing Library** are used to test isolated logic and component rendering.

2. **Integration Testing:**

Ensures that different components interact as expected. For example, testing if the **Search Bar** and **Exercise List** properly update when a user submits a search query.

3. **End-to-End (E2E) Testing:**

**Cypress** or **Puppeteer** could be used to simulate real user behavior, ensuring that the entire application, from login to exercise tracking, works correctly.

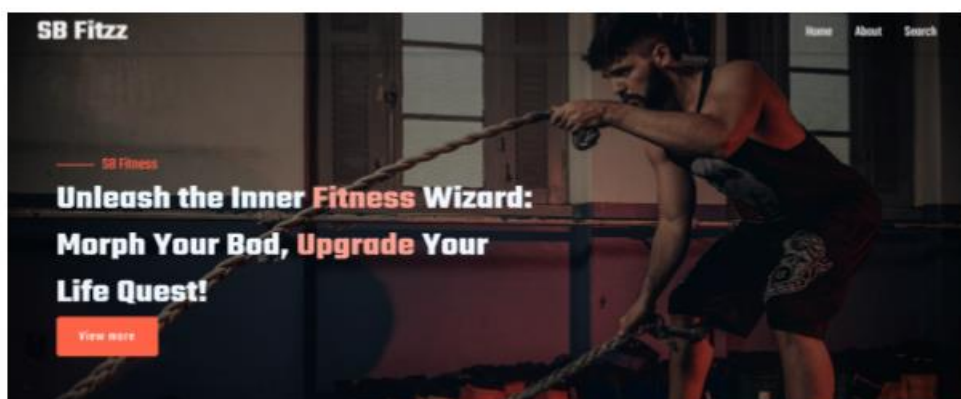
4. **Visual Regression Testing:**

Tools like **Storybook** and **Chromatic** can be used to track UI changes and ensure visual consistency across different screen sizes.

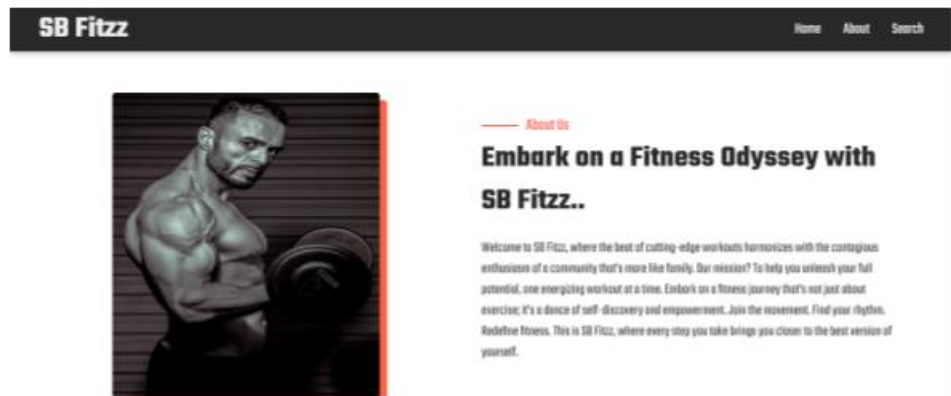
- **Code Coverage:**

The application uses **Jest** to calculate code coverage, ensuring that the critical paths of the app are well-tested. Code coverage tools help identify areas of the codebase that lack sufficient test coverage.

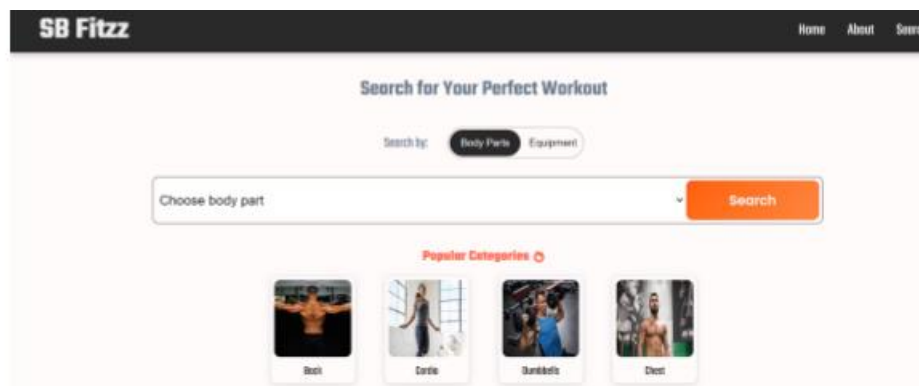
## 12. Screenshots or Demo



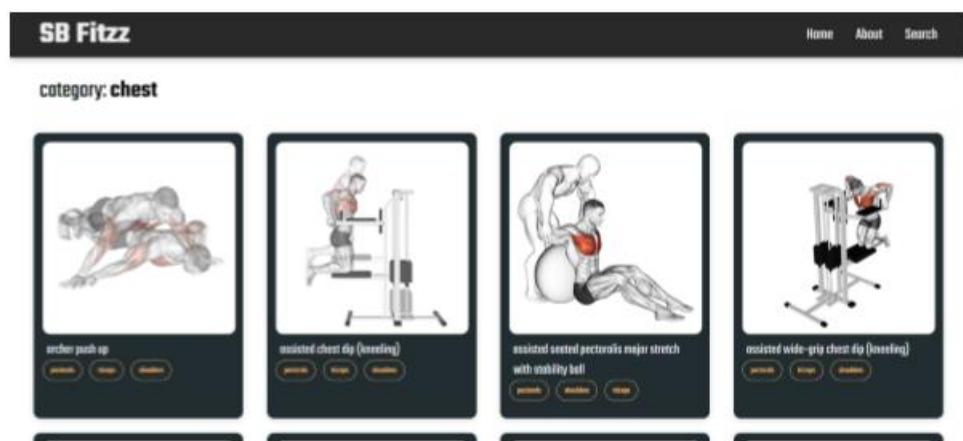
## About



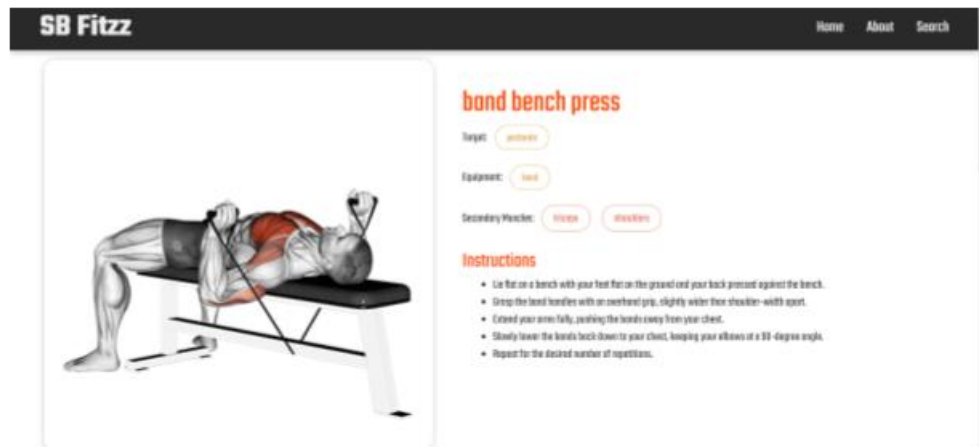
## Search



## Category Page



## Exercise Page



### 13. Known Issues

Some known issues or limitations include:

- **Limited Browser Support:**  
The app may have minor styling inconsistencies on older browsers (e.g., Internet Explorer). The application is primarily optimized for modern browsers (Chrome, Firefox, Safari, Edge).
- **API Rate Limits:**  
Depending on the API usage plan (e.g., Rapid API), the number of requests per day may be limited. This can impact the number of exercises available for fetching in a given session.
- **Dark Mode Bug:**  
Occasionally, when toggling dark mode, some components may not adjust their colors correctly, causing readability issues. This is under investigation and will be resolved in a future update.
- **Performance on Mobile:**  
While the app is responsive, some users may experience slower performance on older mobile devices when loading large datasets or images.

### 14. Future Enhancements

Several enhancements are planned to improve user experience and functionality:

1. **Enhanced Search and Filtering:**  
More granular filters, such as filtering by muscle group, equipment type, difficulty, and workout duration.
2. **User Profiles and Progress Tracking:**  
Allow users to create profiles where they can track their workout progress, save favorite exercises, and set fitness goals.

3. **Workout Plans Integration:**  
Offer personalized workout plans based on user fitness goals (strength, endurance, weight loss) and track progress over time.
4. **Gamification Features:**  
Introduce features like badges, achievements, and workout challenges to increase user engagement.
5. **Offline Mode:**  
Allow users to access and save workout plans offline to continue their exercises without an active internet connection.