FITFLEX

**1.Introduction**

The purpose of this Fitflix project is to provide users with an engaging and interactive platform to track their fitness journey access workout plans. The goal is to promote a healthier lifestyle by offering valuable resources, workout guides, and help users achieve their fitness goals**.**

**1.1 Project Title:**

**FitFlex:** Your Personal Fitness

**1.2 Team Details:**

|  |  |  |
| --- | --- | --- |
| **Team Leader** | J.Bavatharani | akibavatharani@gmail.com |

|  |  |  |  |
| --- | --- | --- | --- |
| **Team Member 1** | J.Bavatharani | akibavatharani@gmail.com |  |
| **Team Member 2** | Divyavalli.D.K | dkdivya2710@gmail.com |  |
| **Team Member 3** | Rajeshwari.S | rajeshwarishankar05@gmail.com |  |
| **Team Member 4** | Keerthiga.C | pandikeerthiga1@gmail.com |  |
| **Team member 5** | Navya Malar.K | saisridevinavi@gmail.com |  |

**2. Project Overview**

**Purpose:**

FitFlix is a comprehensive fitness platform that offers a vast library of workout methods and poses, personalized fitness plans, and expert guidance to help you achieve your fitness goals.

**Features:**

* **User Authentication**: Secure login and signup with JWT authentication.
* **Responsive Design**: Mobile-first approach ensuring optimal user experience.
* **Dynamic UI Elements**: Interactive modals, forms, and real-time data updates.
* **API Integration**: Fetching and displaying data from backend services.
* **Dark Mode Support**: Theming options for user preference.
* **Accessibility Features**: ARIA attributes, keyboard navigation, and screen reader support.

**3. Architecture**

**Component Structure:**

Provide an overview of how the React components are structured. Example:

* **App.js:** Main application wrapper.
* **Components Folder:** Reusable UI components such as buttons, modals, and forms.
* **Pages Folder:** Contains different views such as Home, Dashboard, Profile.
* **Hooks Folder:** Custom hooks to manage state and side effects.

**State Management:**

Explain how the application manages state across components. If using **Redux**, **Context API**, or third-party libraries like **Zustand**, describe how the state flows.

**Routing:**

Mention the routing library used (**React Router** or others) and how navigation is structured:

* Define public and private routes.
* Use nested routes for efficient navigation.

**4. Setup Instructions**

**Prerequisites:**

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/
* **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.

* **Create a new React app:**npx 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.

✓ **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 clone and run the application project from Google Drive:** Follow the steps below:

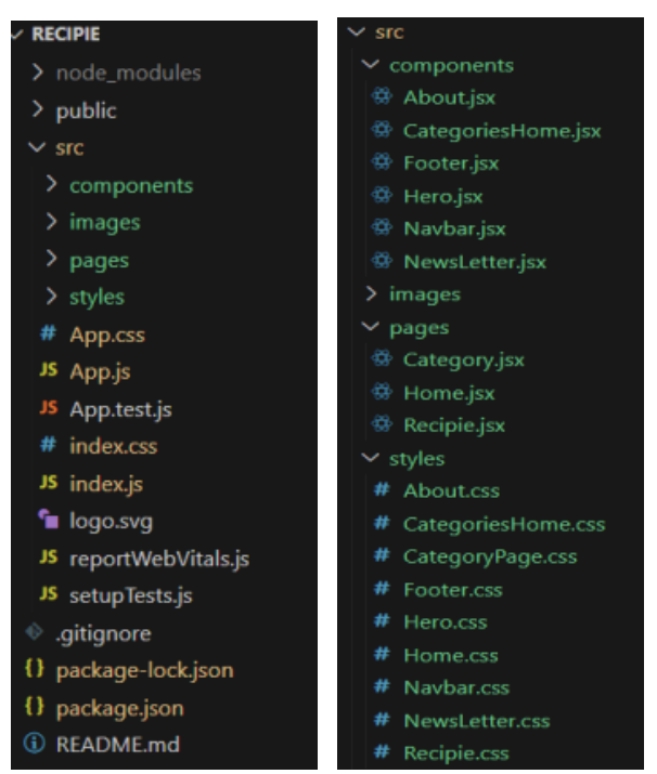
* **Get the code:**
  + Download the code from the drive link given below: https://drive.google.com/drive/folders/1u8PnV\_mE0mwKkH\_CvuNpliZtRLJZMqrO?usp=sharing
* **Install Dependencies:**

Navigate into the cloned repository directory and install libraries:  
cd recipe-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 recipe app'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 url’s 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**

**Start the frontend server locally**

npm start # Starts the development server

The application will be available at [**http://localhost:3000**](http://localhost:3000/).

**7. Component Documentation**

**Key Components:**

* **Navbar.js:** Handles site navigation.
* **Footer.js:** Displays the footer.
* **Dashboard.js:** Main user dashboard with real-time data.
* **Profile.js:** User profile management.

**Reusable Components:**

Explain commonly used UI elements:

* **Button.js:** Customizable button component.
* **Modal.js:** Generic modal component for pop-ups.
* **Loader.js:** Displays loading animations.

**8. State Management**

**Global State:**

* If using **Redux**, explain actions, reducers, and store setup.
* If using **Context API**, detail context providers and consumers.

**Local State:**

* Explain state within individual components using useState and useEffect.

**9. User Interface**

**Hero component**

This section would showcase trending workouts or fitness challenges to grab

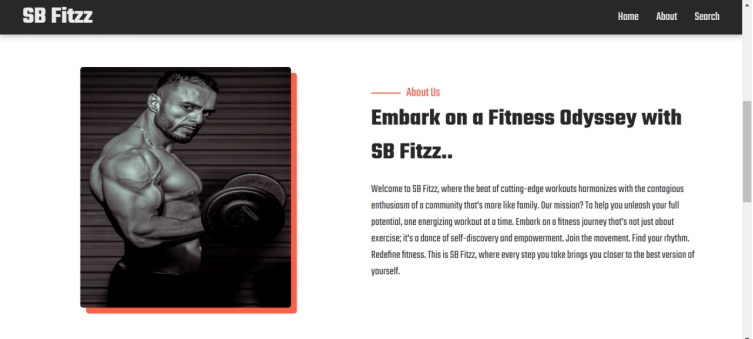
users' attention.



**About**

FitFlex isn't just another fitness app. We're meticulously designed to transform

your workout experience, no matter your fitness background or goals.



**Search**

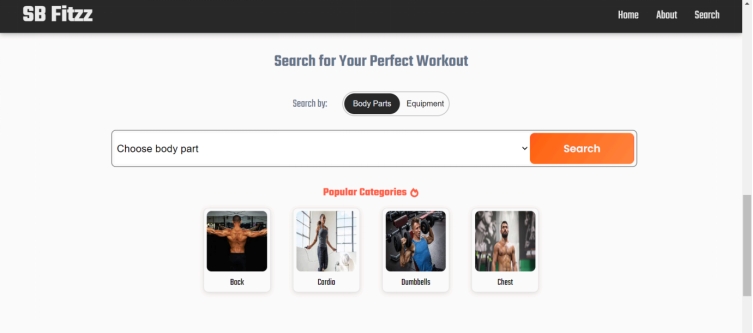
B Fitzz makes finding your perfect workout effortless. Our prominent search bar

empowers you to explore exercises by keyword, targeted muscle group, fitness

level, equipment needs, or any other relevant criteria you have in mind. Simply

type in your search term and let FitFlex guide you to the ideal workout for your

goals.



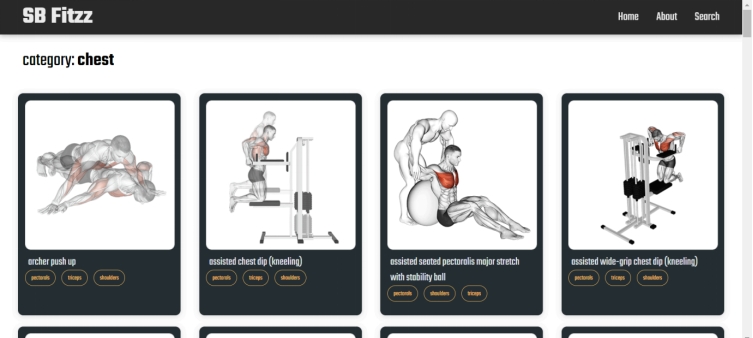
**Category page**

FitFlex would offer a dedicated section for browsing various workout categories.

This could be a grid layout with tiles showcasing different exercise types (e.g.,

cardio, strength training, yoga) with icons or short descriptions for easy

identification.



**Exercise page**

This is where the magic happens! Each exercise page on FitFlex provides a

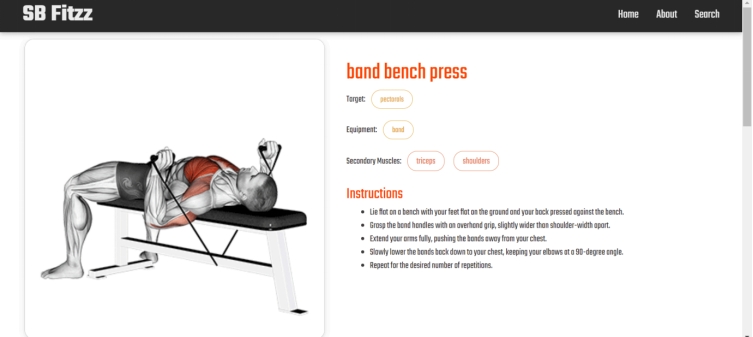
comprehensive overview of the chosen workout. Expect clear and concise

instructions, accompanied by high-quality visuals like photos or videos

demonstrating proper form. Additional details like targeted muscle groups, difficulty

level, and equipment requirements (if any) will ensure you have all the information

needed for a safe and effective workout.



**10. Styling**

**CSS Frameworks/Libraries:**

Mention styling choices such as:

* **Tailwind CSS** for utility-based styling.
* **Material-UI** for prebuilt UI components.
* **Styled-Components** for CSS-in-JS approach.

**Theming:**

Describe if dark mode, custom themes, or design tokens are implemented.

**11. Testing**

Testing ensures application reliability and smooth feature integration. It involves three main levels:

**1. Unit Testing**

Tests individual components or functions in isolation to verify correctness.

* **Focus:** Small, isolated units like functions, React components, or hooks.
* **Tools:** Jest (default React testing framework), React Testing Library.

**2. Integration Testing**

Ensures different components or services interact correctly.

* **Focus:** API calls, state updates, component communication.
* **Tools:** React Testing Library, Mock Service Worker (MSW).

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

Simulates real user interactions and tests the full application flow.

* **Focus:** UI, APIs, and database interactions.
* **Tools:** Cypress, Playwright.

**12.Demo Video**

A link to a live demo :-

https://drive.google.com/file/d/1Lu7B74vrAHbeN1fasiOWjDHthBOvNqda/view?usp=drive\_link