FITFLEX

Introduction:

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. FitFlex is a cutting-edge fitness app that redefines how you approach exercise and wellness. In a world where maintaining a healthy lifestyle can feel overwhelming, FitFlex aims to simplify your fitness journey with an innovative and user-friendly platform designed for people of all fitness levels. Whether you're a beginner just starting to explore fitness, a seasoned gym-goer looking to take your training to the next level, or someone with specific fitness goals, FitFlex offers everything you need to succeed. FitFlex also features a dynamic search function that makes it easier than ever to find the exercises or routines that best suit your individual needs. The app houses an extensive library of exercises, from strength training and cardio to flexibility and mobility work. Each exercise is accompanied by clear instructions and helpful tips, ensuring proper technique and minimizing the risk of injury. Whether you're at home, in the gym, or on the go, FitFlex provides workouts that you can adapt to your environment. Personalization is another key component of FitFlex. The app tailors workout plans based on your fitness goals, level, and preferences. This personalized approach ensures that you're always working toward a goal that's realistic and achievable, whether that's building muscle, improving endurance, losing weight, or enhancing flexibility.

Description:

Our innovative fitness app is meticulously designed to revolutionize the way you engage with exercise routines, catering to the diverse interests of both fitness enthusiasts and seasoned workout professionals. With a focus on an intuitive user interface and a comprehensive feature set, FitFlex is set to redefine the entire fitness discovery and exercise experience. Crafted with a commitment to user-friendly aesthetics, FitFlex immerses users in an unparalleled fitness journey. Effortlessly navigate through a wide array of exercise categories with features like dynamic search, bringing you the latest and most effective workouts from the fitness world. From those embarking on their fitness journey to seasoned workout aficionados, FitFlex embraces a diverse audience, fostering a dynamic community united by a shared passion for a healthy lifestyle. Our vision is to reshape how users interact with fitness, presenting a platform that not only provides effective exercise routines but also encourages collaboration and sharing within the vibrant fitness community. Embark on this fitness adventure with us, where innovation seamlessly intertwines with established exercise principles. Every tap within FitFlex propels you closer to a realm of diverse workouts and wellness perspectives. Join us and experience the evolution of fitness engagement, where each feature is meticulously crafted to offer a glimpse into the future of a healthier you. Elevate your fitness exploration with FitFlex, where every exercise becomes a gateway to a world of wellness waiting to be discovered and embraced. Trust FitFlex to be your reliable companion on the journey to staying connected with a fit and active lifestyle.

TEAM MEMBERS & ROLES:

1) Pavithra S - (Demo Video)
2) Tabassum Khan - (Coding)
3) Kaarthiga - (Coding & Debugging)
4) M.B Jeevitha – (Documentation)

PROJECT OVERVIEW:

Scenario based Intro:

You lace up your sneakers, determined to get serious about your fitness. But where do you start? Suddenly, you remember FitFlex, the innovative app that promised to revolutionize your workouts. With a tap, you open the app. Vibrant visuals flood the screen – personalized workout plans, diverse exercise categories, and a supportive community. This isn't your typical fitness app. FitFlex feels...different. Intrigued, you select a workout and get ready to experience the future of fitness.

Project Goals and Objectives:

The overarching aim of FitFlex is to offer an accessible platform tailored for individuals passionate about fitness, exercise, and holistic well-being.

Our key objectives are as follows:

✓ **User-Friendly Experience:** Develop an intuitive interface that facilitates easy navigation, enabling users to discover, save, and share their preferred workout routines.

✓ **Comprehensive Exercise Management:** Provide robust features for organizing and managing exercise routines, incorporating advanced search options for a personalized fitness experience.

✓ **Technology Stack:** Harness contemporary web development technologies, with a focus on React.js, to ensure an efficient and enjoyable user experience.

Features of FitFlex:

✓ **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.

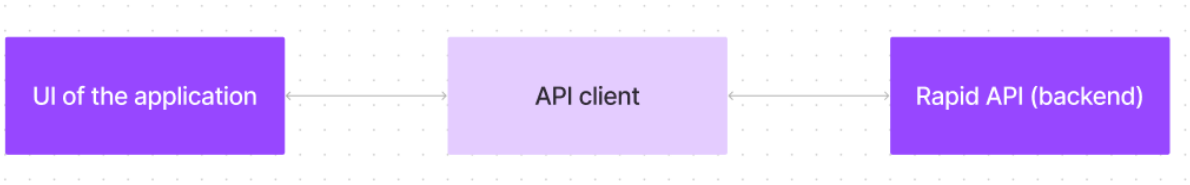
✓ **Visual Exercise Exploration:** Engage with workout routines through curated image galleries, allowing users to explore different exercise categories and discover new fitness challenges visually.

✓ **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.

ARCHITECTURE:

Technical Architecture:



FitFlex prioritizes a user-centric approach from the ground up. The engaging user interface (UI), likely built with a framework like React Native, keeps interaction smooth and intuitive. An API client specifically designed for FitFlex communicates with the backend, but with a twist: it leverages Rapid API. This platform grants access to various external APIs, allowing FitFlex to potentially integrate features like fitness trackers, nutrition data, or workout tracking functionalities without building everything from scratch. This approach ensures a feature-rich experience while focusing development efforts on the core FitFlex functionalities.

PRE-REQUISITES:

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.

Install React.js, a JavaScript library for building user interfaces.

- 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.

✓ **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.

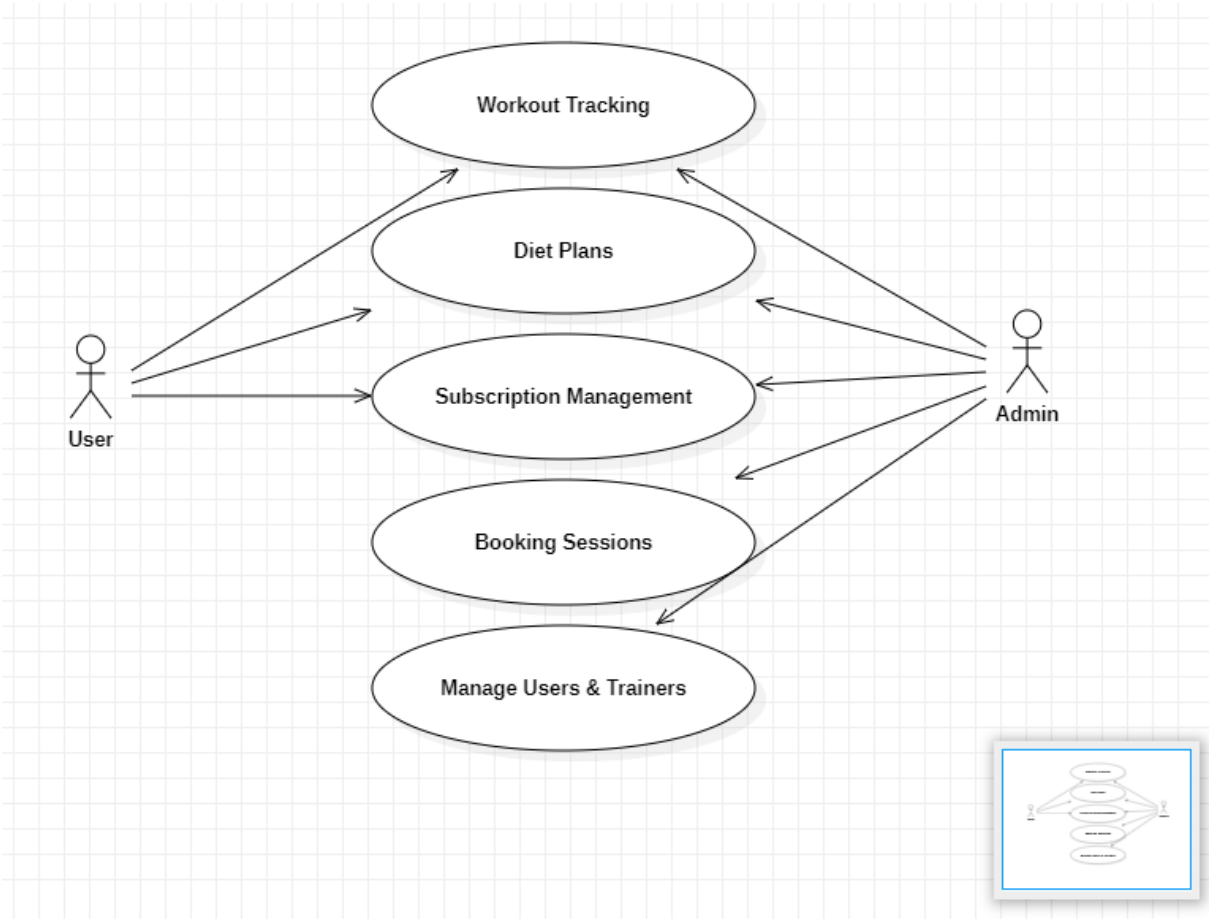
- Git: Download and installation instructions can be found at: https://git-scm.com/downloads

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

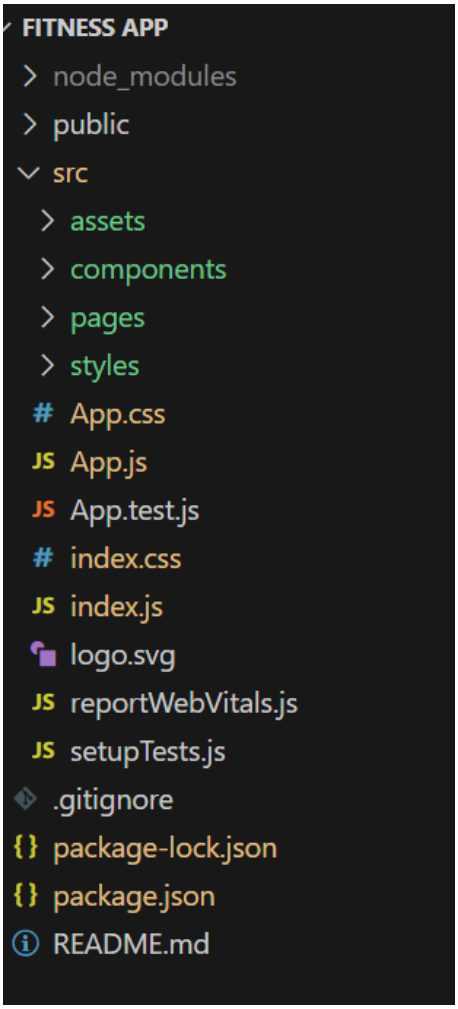
- Visual Studio Code: Download from https://code.visualstudio.com/download
- Sublime Text: Download from https://www.sublimetext.com/download
- WebStorm: Download from https://www.jetbrains.com/webstorm/download

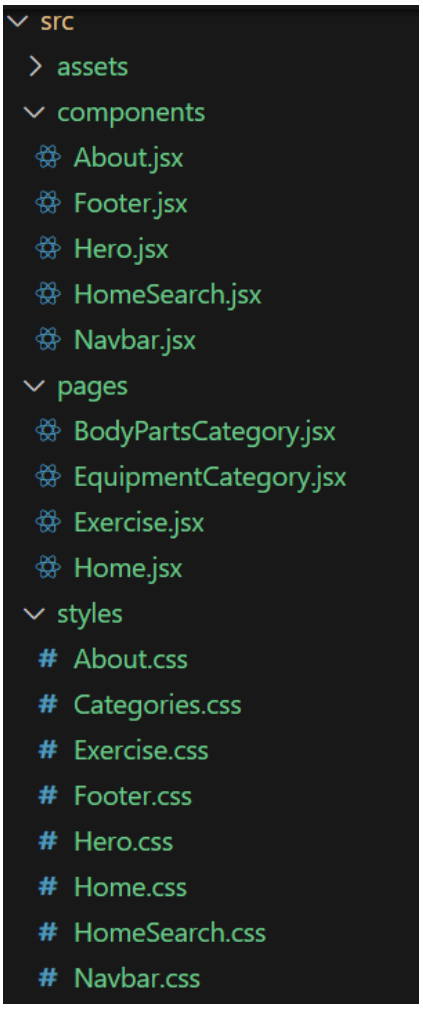
USE CASE DIAGRAM:

A Use Case defines a specific interaction between a user (or system) and a software application or system, describing how the system responds to certain actions or requests in order to achieve a goal. It provides a detailed description of a user's objectives, the steps they take to achieve those objectives, and the system's responses throughout the process. Use cases are typically employed in system design and software development to ensure that the functionality and behavior of the system align with user requirements.



Project 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.

Project flow:

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:

- React Js
- React Router Dom
- React Icons
- Bootstrap/tailwind css
- Axios

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

2: Project Development

- Setup the Routing paths

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

```
<div className="App">

  <Navbar />

  <Routes>

    <Route path="/" element={<Home />} />
    <Route path="/bodyPart/:id" element={<BodyPartsCategory />} />
    <Route path="/equipment/:id" element={<EquipmentCategory />} />
    <Route path="/exercise/:id" element={<Exercise />} />
  </Routes>

  <Footer />
</div>
```

- 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.

API Key:

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.

```
const fetchData = async (id) => {
    const options = {
        method: 'GET',
        url: `https://exercisedb.p.rapidapi.com/exercises/equipment/${id}`,
        params: {limit: '50'},
        headers: {
          'X-RapidAPI-Key': 'your api key',
          'X-RapidAPI-Host': 'exercisedb.p.rapidapi.com'
        }
    };

    try {
        const response = await axios.request(options);
        console.log(response.data);
        setExercises(response.data);
    } catch (error) {
        console.error(error);
    }

}
```

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 This line defines an asynchronous function named fetchData that takes

```
useEffect(()=>{
    if (id){
        fetchData(id)
    }
},[])

const fetchData = async (id) => {
    const options = {
      method: 'GET',
      url: `https://exercisedb.p.rapidapi.com/exercises/exercise/${id}`,
      headers: {
        'X-RapidAPI-Key': 'ae40549393msh0c35372c617b281p103ddcjsn0f4a9ee43ff0',
        'X-RapidAPI-Host': 'exercisedb.p.rapidapi.com'
      }
    };

    try {
        const response = await axios.request(options);
        console.log(response.data);
        setExercise(response.data);

        fetchRelatedVideos(response.data.name)
    } catch (error) {
        console.error(error);
    }

}
```

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).the Exercise ID, we fetch the details of a particular exercise.

- **Fetching Exercise details**

Now, with the help of the Exercise ID, we fetch the details of a particular exercise with API request.

```
useEffect(()=>{
    if (id){
        fetchData(id)
    }
},[])

const fetchData = async (id) => {
    const options = {
      method: 'GET',
      url: `https://exercisedb.p.rapidapi.com/exercises/exercise/${id}`,
      headers: {
        'X-RapidAPI-Key': 'ae40549393msh0c35372c617b281p103ddcjsn0f4a9ee43ff0',
        'X-RapidAPI-Host': 'exercisedb.p.rapidapi.com'
    }
  };

    try {
        const response = await axios.request(options);
        console.log(response.data);
        setExercise(response.data);

        fetchRelatedVideos(response.data.name)
    } catch (error) {
        console.error(error);
    }

}
```

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.

```
const fetchRelatedVideos = async (name)=>{
  console.log(name)
  const options = {
    method: 'GET',
    url: 'https://youtube-search-and-download.p.rapidapi.com/search',
    params: {
      query: `${name}`,
      hl: 'en',
      upload_date: 't',
      duration: 'l',
      type: 'v',
      sort: 'r'
    },
    headers: {
      'X-RapidAPI-Key': 'ae40549393msh0c35372c617b281p103ddcjsn0f4a9ee43ff0',
      'X-RapidAPI-Host': 'youtube-search-and-download.p.rapidapi.com'
    }
  };

  try {
    const response = await axios.request(options);
    console.log(response.data.contents);
    setRelatedVideos(response.data.contents);
  } catch (error) {
    console.error(error);
  }
}
```

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, which is '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.
- 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).

USER INTERFACE:

After completing the code, run the react application by using the command “npm start” or “npm run dev” if you are using vite.js

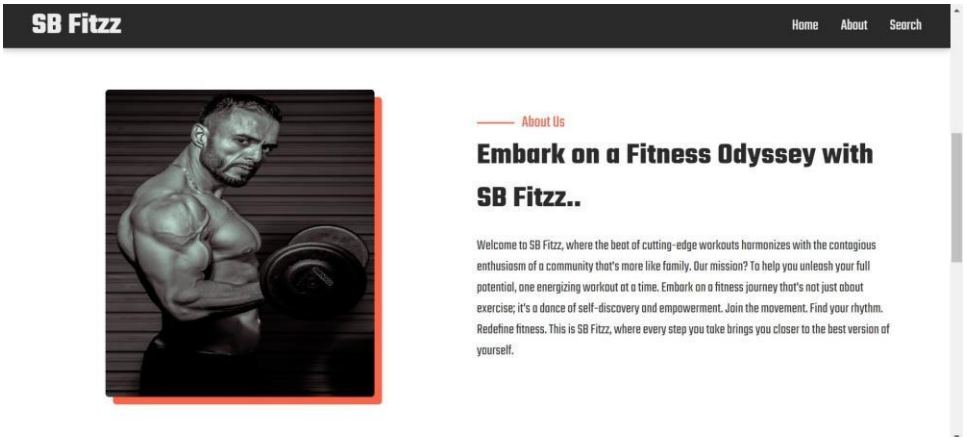
Here are some of the screenshots of the application.

- **Hero component**

  this section would showcase trending workouts or fitness challenges to grab users' attention.

- **About**

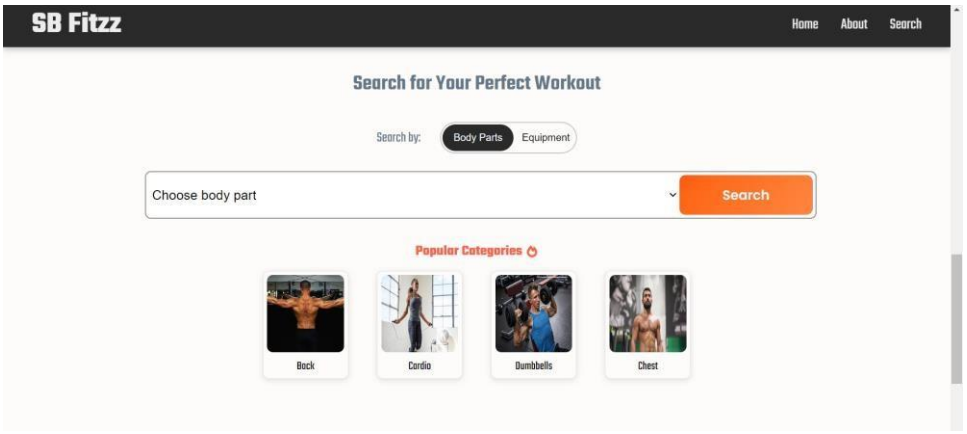
FitFlex isn't just another fitness app. We're meticulously designed to transform



meticulously designed to transform your workout experience, no matter your fitness background or goals.

- **Search**

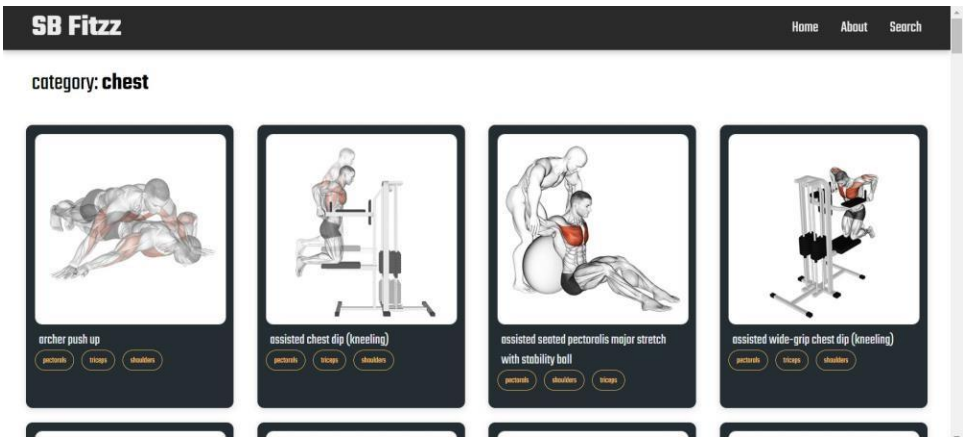
B Fitzz makes finding your perfect workout effortless. Our prominent search bar empowers you to explore exercises by keyword, targeted muscle group, fitness level, equipment needs, or any other relevant criteria you have in mind. Simply



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

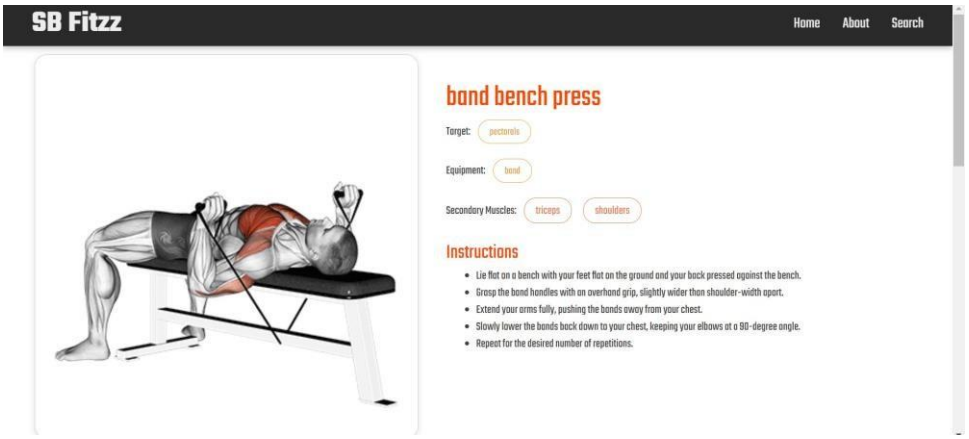
- **Category page**

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.



STYLING

- **CSS Frameworks/Libraries**: FitFlex uses **Tailwind CSS** for a streamlined and responsive design, ensuring a modern and mobile-friendly experience. Additionally, **Styled-Components** is utilized for scoped component styling in React.
- **Theming**: The website supports **dark and light mode** using CSS variables, allowing users to customize their experience. A **custom design system** with reusable UI components ensures brand consistency across the platform.

TESTING

We want FitFlex to run smoothly and be as bug-free as possible, so we follow a structured testing approach. Here's how we ensure everything works as expected:

Testing Strategy:

- **Unit Testing** – We use **Jest and React Testing Library** to test individual components, making sure buttons, forms, and UI elements function correctly.
- **Integration Testing** – This ensures that different components work together as expected, such as verifying that the workout tracker updates progress when a user logs an exercise.

- **End-to-End (E2E) Testing** – Using **Cypress**, we simulate real user interactions like signing up, starting a workout, and tracking progress to catch any issues in the complete user experience.

Code Coverage

- We track how much of the code is covered by tests using Jest's **coverage reports** to ensure all critical features are well-tested.
- **GitHub Actions** runs automated tests every time new code is pushed to detect bugs before they go live.
- Our goal is to maintain high test coverage so FitFlex remains reliable and user-friendly.

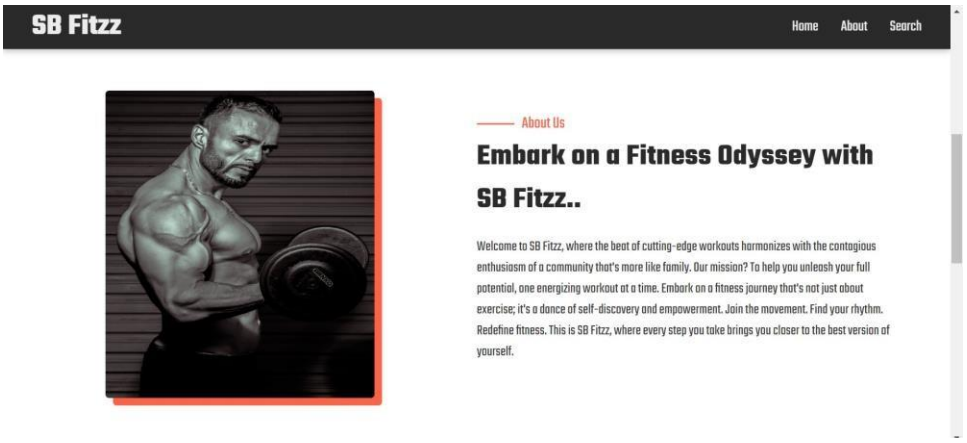
By following this testing approach, we ensure that FitFlex is stable and performs well across different devices and user scenarios.

SCREENSHOTS OR DEMO:

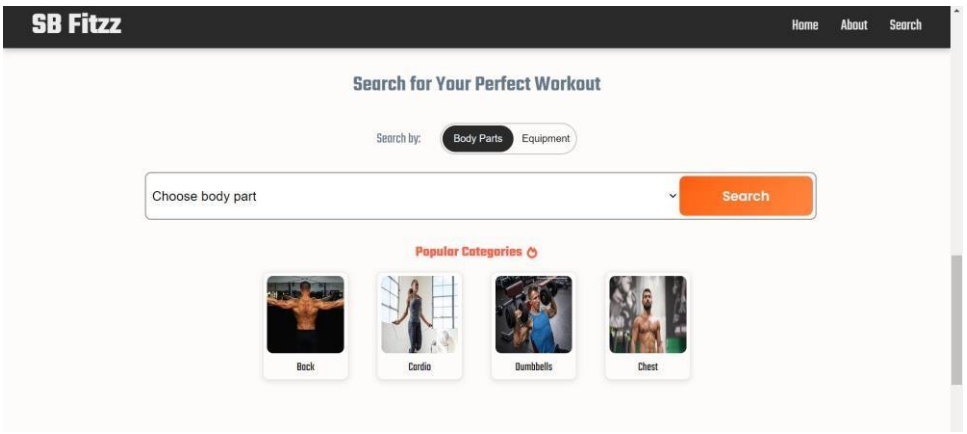
- A live demo of the FitFlex website can be accessed at: https://github.com/03252005/FitnessApp-react-
- Below are some screenshots showcasing key features:
    - Home Page with interactive workout plans
    - Personalized dashboard for users
    - Mobile-friendly workout tracking system
    - Nutrition and meal plan recommendations

HOME PAGE:

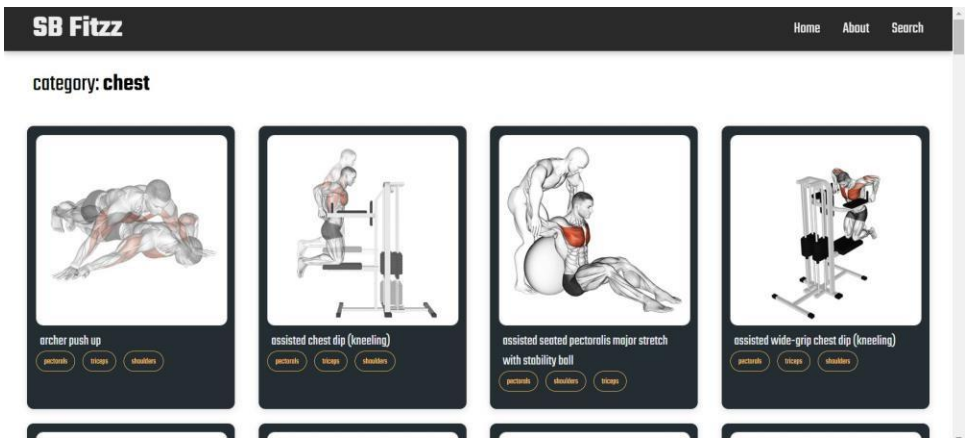
ABOUT US :



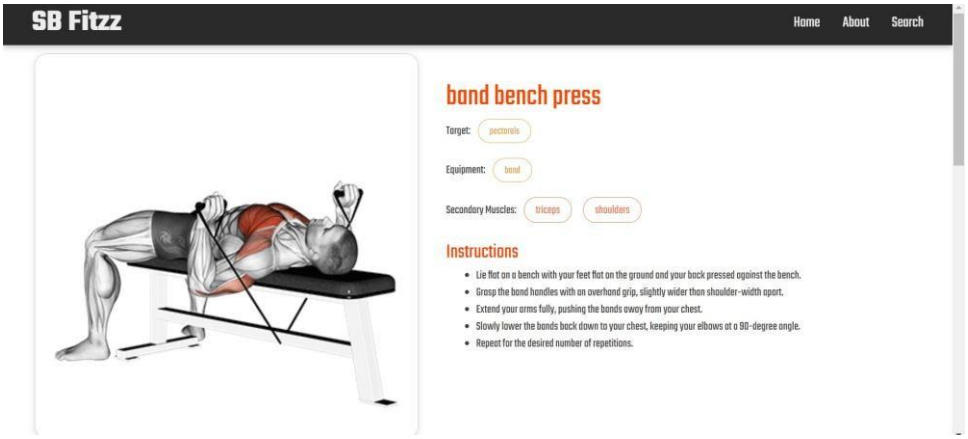
SEARCH PAGE:



CATEGORY PAGE:



EXERCISE PAGE:



KNOWN ISSUES

- Some exercises may not have video demonstrations available yet.
- Dark mode styling inconsistencies in some components.
- Minor layout shifts on mobile screens when resizing.
- Performance optimization required for loading high-resolution images.

FUTURE ENHANCEMENTS

- **AI-Powered Personal Trainer**: Implement a chatbot that suggests workout routines based on user preferences.
- **Gamification Features**: Add badges, leaderboards, and challenges to keep users motivated.
- **Community Forum**: A space where users can share fitness progress and interact.
- **Integration with Wearable Devices**: Sync with Apple Watch, Fitbit, and Google Fit for real-time tracking.
- **Enhanced UI Animations**: Implement smooth transitions and micro-interactions for an engaging experience.