

# Fitness tracker

**FITFLEX:YOUR PERSONAL FITNESS  
COMPANION**

→ **TEAM DETAILS**

<b>TEAMLEADER</b>	<b>EMAIL ID</b>
GOKUL.T	gokulharish765@gmail.com

<b>TEAM MEMBERS</b>	<b>EMAIL ID</b>
RAHUL.P	raghul.p2904@gmail.com
VIGNESHWARAN.S	vigneshwaranvignesh505@gmail.com

## 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.

## 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.

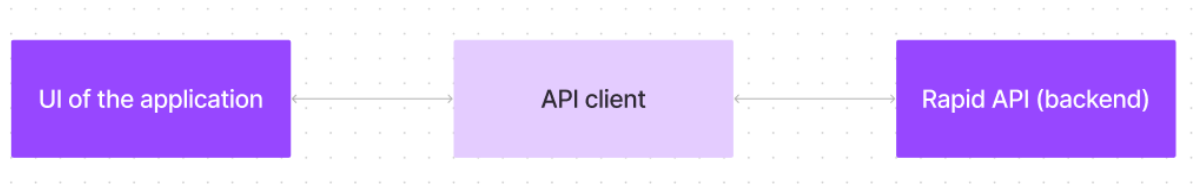
Our key objectives are as follows:

- ✓ **User-Friendly Experience:** Develop an intuitive interface that facilitates easy navigation, enabling users to effortlessly 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.

## 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/>

### ✓ 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:

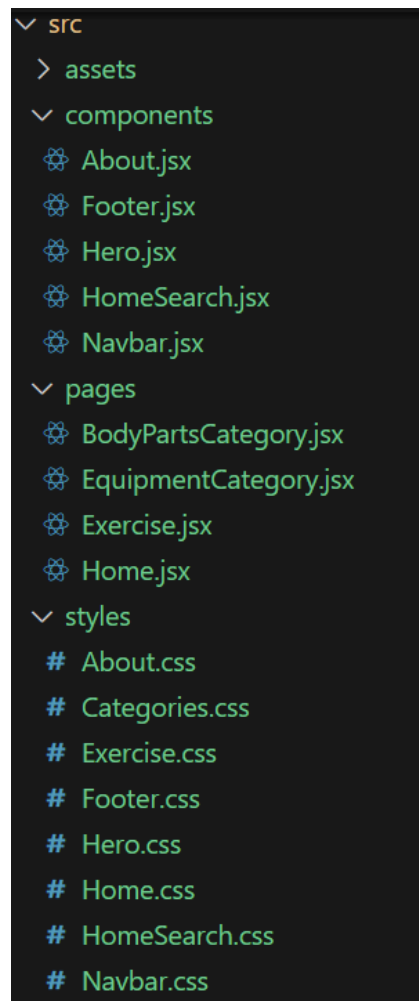
```
npm start
```

- 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.

## Project structure:



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.
- ✓ **Development Environment:** Choose a code editor or Integrated Development

Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/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](https://drive.google.com/drive/folders/14f9eBQ5W7VrLdPhP2W6PzOU_HCy8UMex?usp=sharing)

**Install Dependencies:**

- Navigate into the cloned repository directory and install libraries:

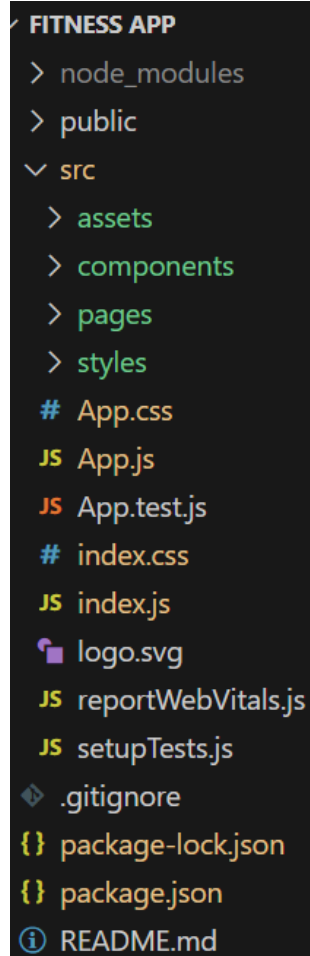
```
cd fitness-app-react  
npm install
```

✓ **Start the Development Server:**

- To start the development server, execute the following command:

```
npm start
```

**Access the App:**



```
✓ FITNESS APP
  > node_modules
  > public
  ✓ src
    > assets
    > components
    > pages
    > styles
  # App.css
  JS App.js
  JS App.test.js
  # index.css
  JS index.js
  logo.svg
  JS reportWebVitals.js
  JS setupTests.js
  .gitignore
  {} package-lock.json
  {} package.json
  ⓘ README.md
```

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

### 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](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:

- 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>

## **Milestone 2: Project Development**

- ❖ **Setup the Routing paths**

Setup the clear routing paths to access various files in the app



```

<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>

```

lication.

- ❖ 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.

```

const bodyPartsOptions = {
  method: 'GET',
  url: 'https://exercisedb.p.rapidapi.com/exercises/bodyPartList',
  headers: {
    'X-RapidAPI-Key': 'place your api key',
    'X-RapidAPI-Host': 'exercisedb.p.rapidapi.com'
  }
};

const equipmentOptions = {
  method: 'GET',
  url: 'https://exercisedb.p.rapidapi.com/exercises/equipmentList',
  headers: {
    'X-RapidAPI-Key': 'place your api key',
    'X-RapidAPI-Host': 'exercisedb.p.rapidapi.com'
  }
};

useEffect(() => {
  fetchData();
}, [])

const fetchData = async () =>{
  try {
    const bodyPartsData = await axios.request(bodyPartsOptions);
    setBodyParts(bodyPartsData.data);

    const equipmentData = await axios.request(equipmentOptions);
    setEquipment(equipmentData.data);
  } catch (error) {
    console.error(error);
  }
}

```

Here's a breakdown of the code:

#### *Dependencies:*

The code utilizes the following libraries:

Axios: A popular promise-based HTTP client for JavaScript. You can add a link to the official documentation for Axios <https://axios-http.com/>

*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 the Exercise ID, we fetch the details of a particular exercise

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

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.



```
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).

## **Project Execution:**

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

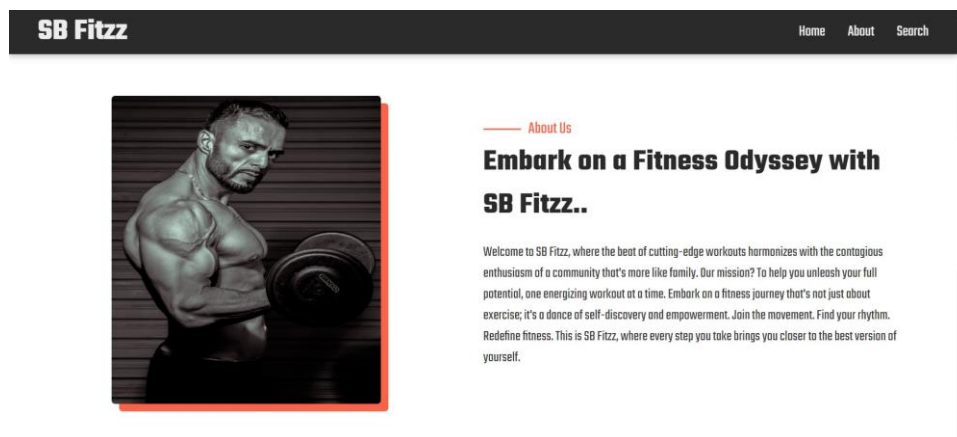
Here are some of the screenshots of the application.

## ? Hero component

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

## ? About

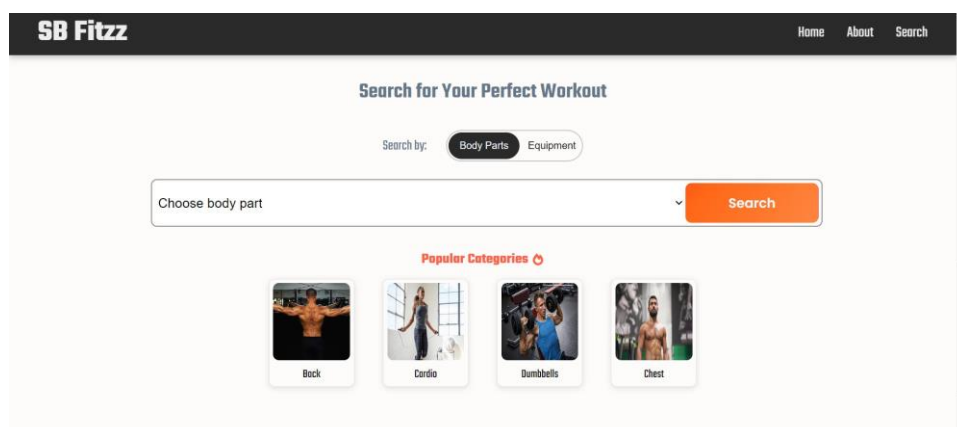
FitFlex isn't just another fitness app. We're meticulously designed to transform



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

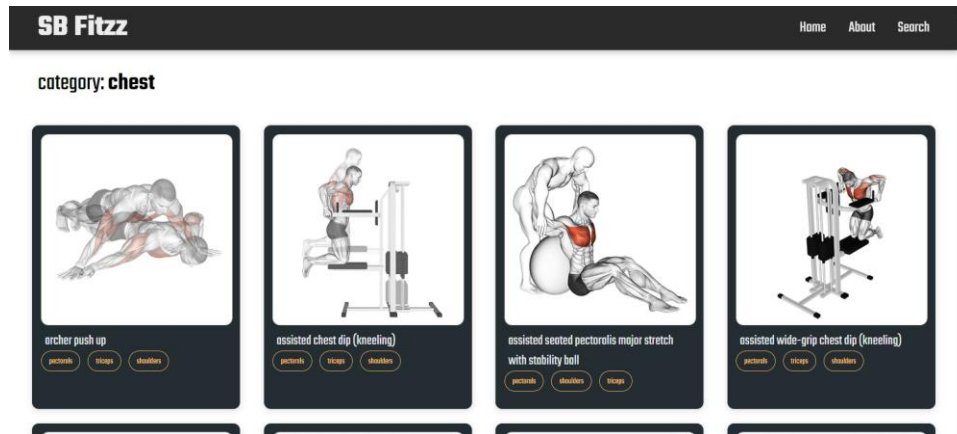


type in your search term and let FitFlex guide you to the ideal workout for your goals.

## ? Category page

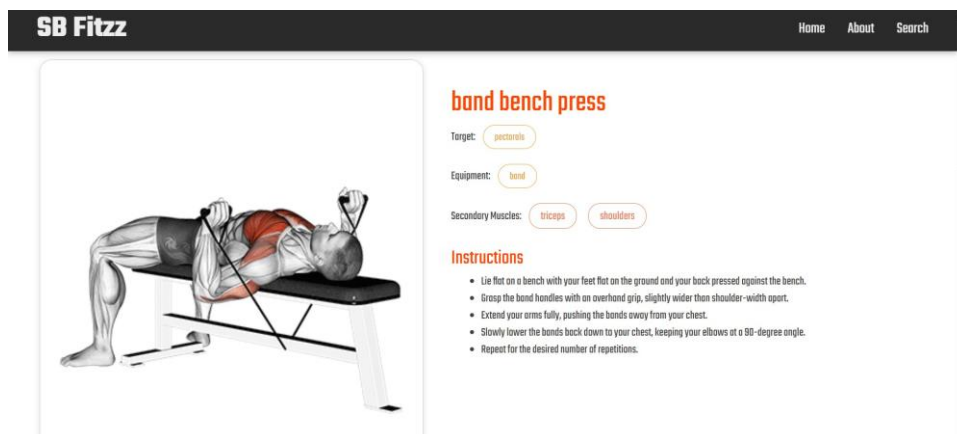
FitFlex would offer a dedicated section for browsing various workout categories. This could be a grid layout with tiles showcasing different exercise types (e.g., cardio, strength training, yoga) with icons or short descriptions for easy

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.



Demo link: <https://drive.google.com/file/d/1mMqMb41RtroiFbUQ-1ZfeYfWJZ6okSNb/view?usp=sharing>

## **CONCLUSION**

In conclusion, fitness trackers have become invaluable tools for monitoring and improving overall health and fitness. They offer a wealth of features that can help users track their physical activity, sleep patterns, heart rate, and even stress levels. By providing real-time data and insights, fitness trackers can motivate individuals to set and achieve personal fitness goals, leading to improved physical well-being. However, it's important to remember that while fitness trackers can be a helpful resource, they should be used in conjunction with professional medical advice and a balanced lifestyle for optimal health results.

**THANKYOU...!!**