**FIT FLEX**

**(Fitness Tracker)**

**Team Details:**

|  |  |
| --- | --- |
| **TEAM LEADER** | **EMAIL ID** |
| NISHA. S | nishasathya20@gmail.com |

|  |  |
| --- | --- |
| **TEAM MEMBERS** | **EMAIL ID** |
| DIVYA BHARATHI. A | divyabharathi67262@gmail.com |
| HASMATH HASEENA. S | hasenahasmath@gmail.com |
| SHAMILI. S | shamilinive21@gmail.com |

**INDEX**

|  |  |
| --- | --- |
| **S.NO** | **TOPIC** |
| 1. | Introduction |
| 2. | Technical Architecture |
| 3. | Project Goals and Objectives |
| 4. | Features of FitFlex |
| 5. | Pre-requisites |
| 6. | Project Structure |
| 7. | Project Flow |
| 8. | API Integration |
| 9. | Project Execution |
| 10. | Conclusion |

**FITFLEX:**

**Introduction:**

A fitness tracker is a mobile application designed to monitor and track health-related metrics such as steps taken, heart rate, calories burned, and sleep patterns. The primary objective of a fitness tracker project is to help users improve their fitness levels by providing real-time data and personalized insights.

This project involves developing a wearable device or a mobile application that collects user data using sensors like an accelerometer, heart rate sensor, and GPS. The collected data is processed and stored in a cloud database, allowing users to access their fitness reports anytime. Key features include step counting, heart rate monitoring, activity tracking, calorie estimation, and sleep analysis.

The system architecture includes hardware components (sensors and microcontrollers) and software applications (mobile/web apps) that present fitness data in a user-friendly interface. Advanced features like AI-based health insights, goal setting, and social integration make the tracker more engaging.

**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 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.

**Current trends in Fitness Tracker:**

As of March 2025, fitness trackers have evolved significantly, integrating advanced health monitoring features, personalized coaching, and seamless connectivity with other devices. Here are some current trends shaping the fitness tracker landscape:

**1. Comprehensive Health Monitoring:**

Modern fitness trackers now offer a wide array of health metrics beyond basic activity tracking. Devices like the Fitbit Charge 6 monitor heart rate variability, blood oxygen levels, and skin temperature, providing users with a holistic view of their health.

**2. Personalized Coaching and Stress Management:**

There's a growing emphasis on personalized coaching to help users achieve their fitness goals. For instance, WHOOP introduced the "WHOOP Coach" feature powered by OpenAI, delivering individualized performance coaching.

**3. Integration with Broader Ecosystems:**

Fitness trackers are increasingly integrating with larger digital ecosystems to enhance user experience. This trend reflects a move towards multifunctional devices that serve both fitness and everyday needs.

**4. Diverse Form Factors:**

Beyond traditional wrist-worn devices, fitness trackers are adopting new form factors. The Oura Ring 4, for example, offers comprehensive health tracking in a discreet ring design, appealing to users seeking non-wrist options.

**5. Focus on Recovery and Holistic Health:**

There's a shift towards emphasizing recovery and overall health rather than just activity metrics. Devices like the WHOOP 4.0 prioritize monitoring sleep and recovery, helping users balance exertion and rest for optimal health outcomes.

**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.

● 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.

● Start the development server:

npm start

This command launches the development server, and access our React app

at <http://localhost:3000> in our 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 the code editor or Integrated Development Environment(IDE) that suits for our preferences, we use Visual Studio Code.

Visual Studio Code: <https://code.visualstudio.com/download>.

**Install Dependencies:**

* Navigate into the cloned repository and install libraries:

cd fitness-app-react

npm install

Start the Development Server:

To start the development server, execute the following command:

npm run dev (vite) or npm start.

**OUR GITHUB REPOSITORY LINK:**

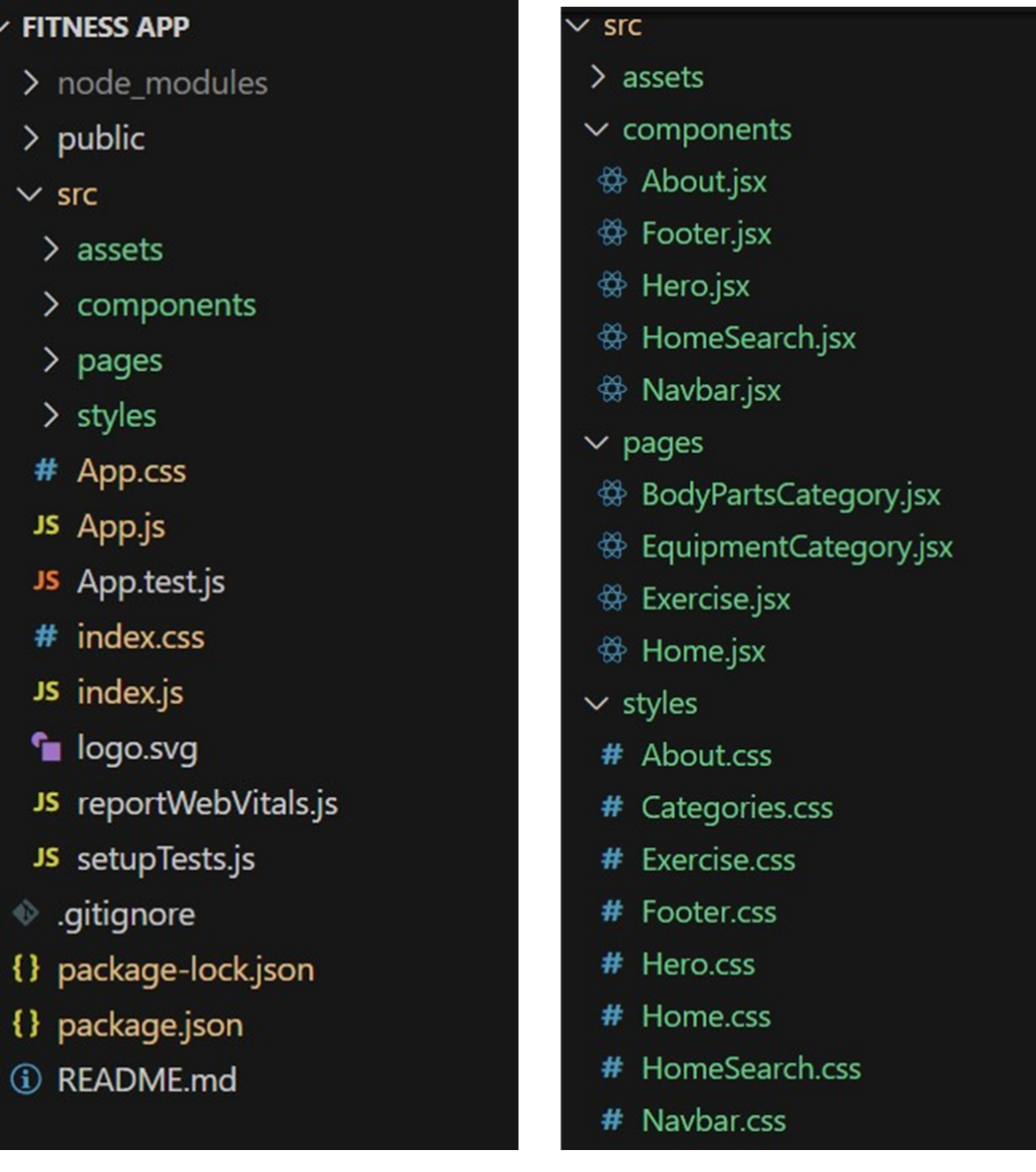
<https://github.com/Nishas-20/fitness_track.git>

**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.

Now we have successfully installed and set up the application on our local machine. We can now proceed with further customization, development, and testing as needed.

**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:**

**Project demo:**

Before starting to work on this project, let’s see the demo.

Demo link:

<https://github.com/Nishas-20/fitness_track.git>

**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:

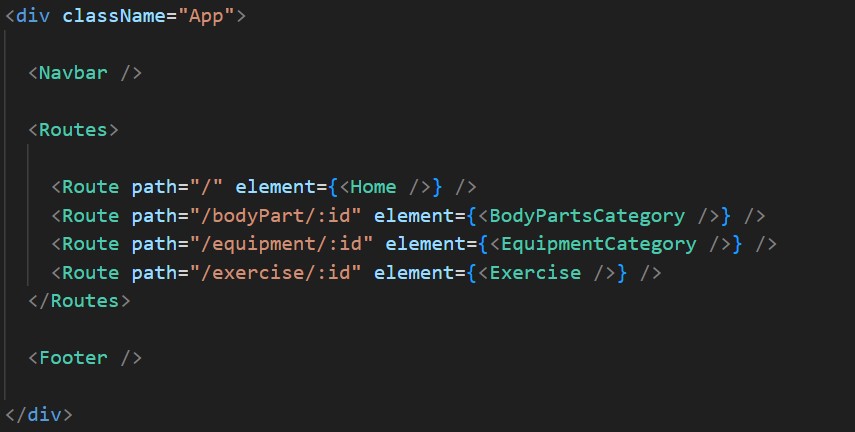
* + - * + React Js
        + React Router Dom
        + React Icons
        + Bootstrap/tailwind CSS
        + Axios
* For further reference, use the following resources

* + <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 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.



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.

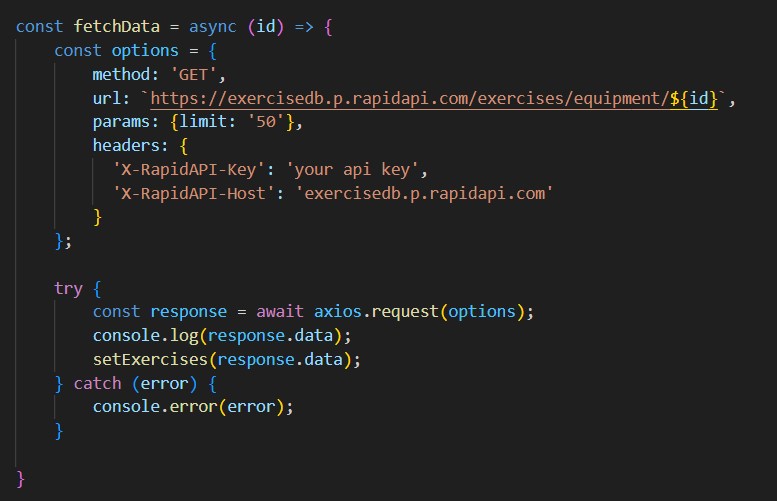
*useEffect Hook:*

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.



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: 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:*

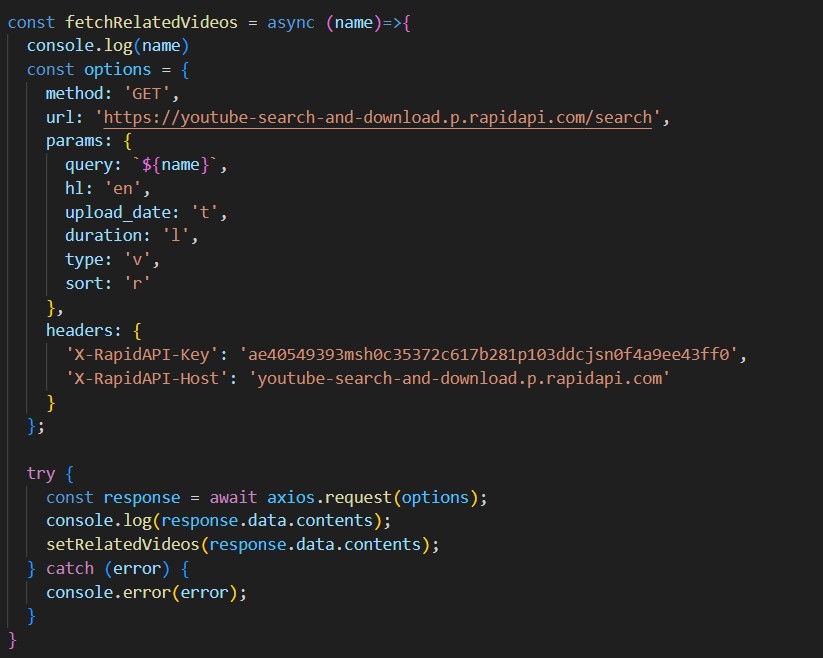
* The then method is used to handle the response from the API request. If the request is successful (i.e., status code is 200), the response is converted to JSON format using response.json().
* The then method then likely processes the fetched exercise data, which might involve storing it in a state 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:

'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.

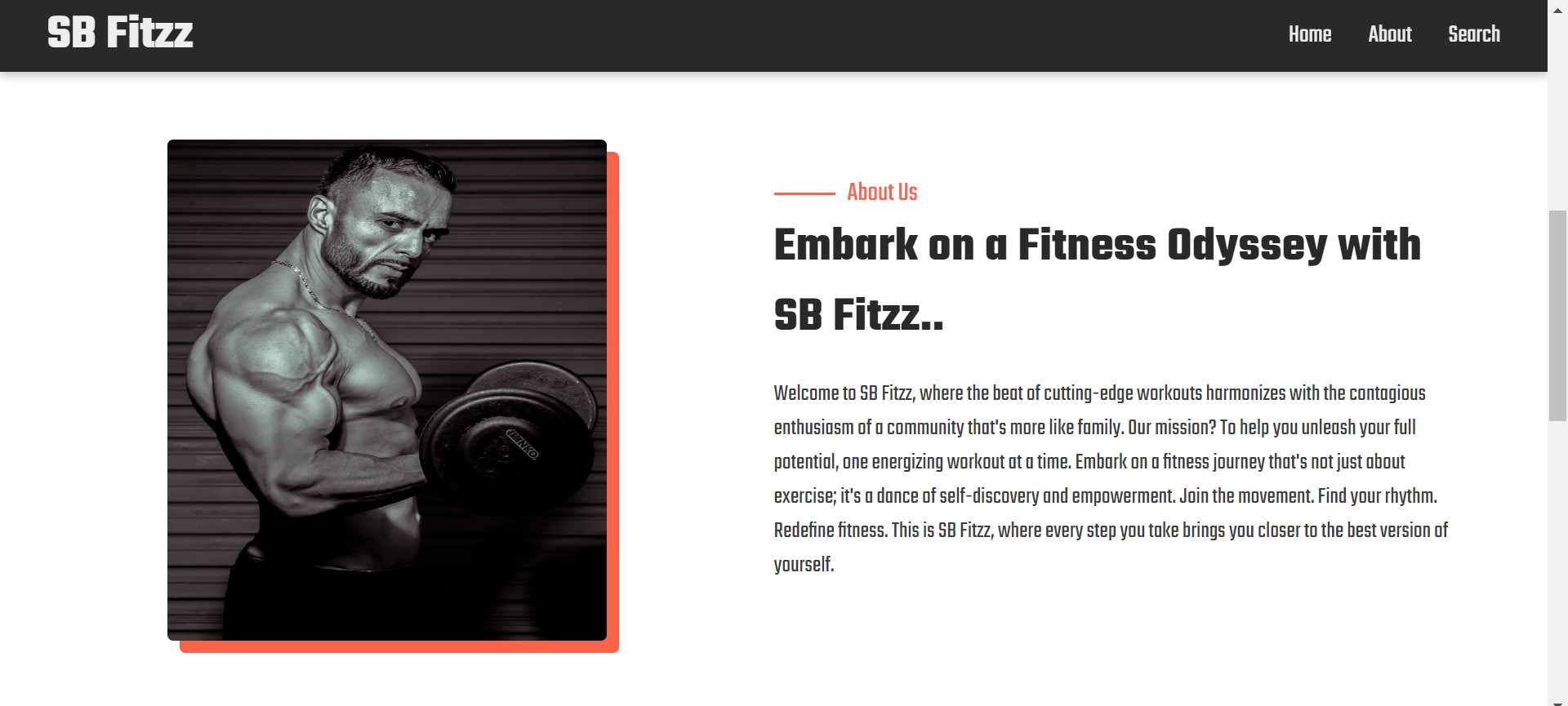
* **Hero component**

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



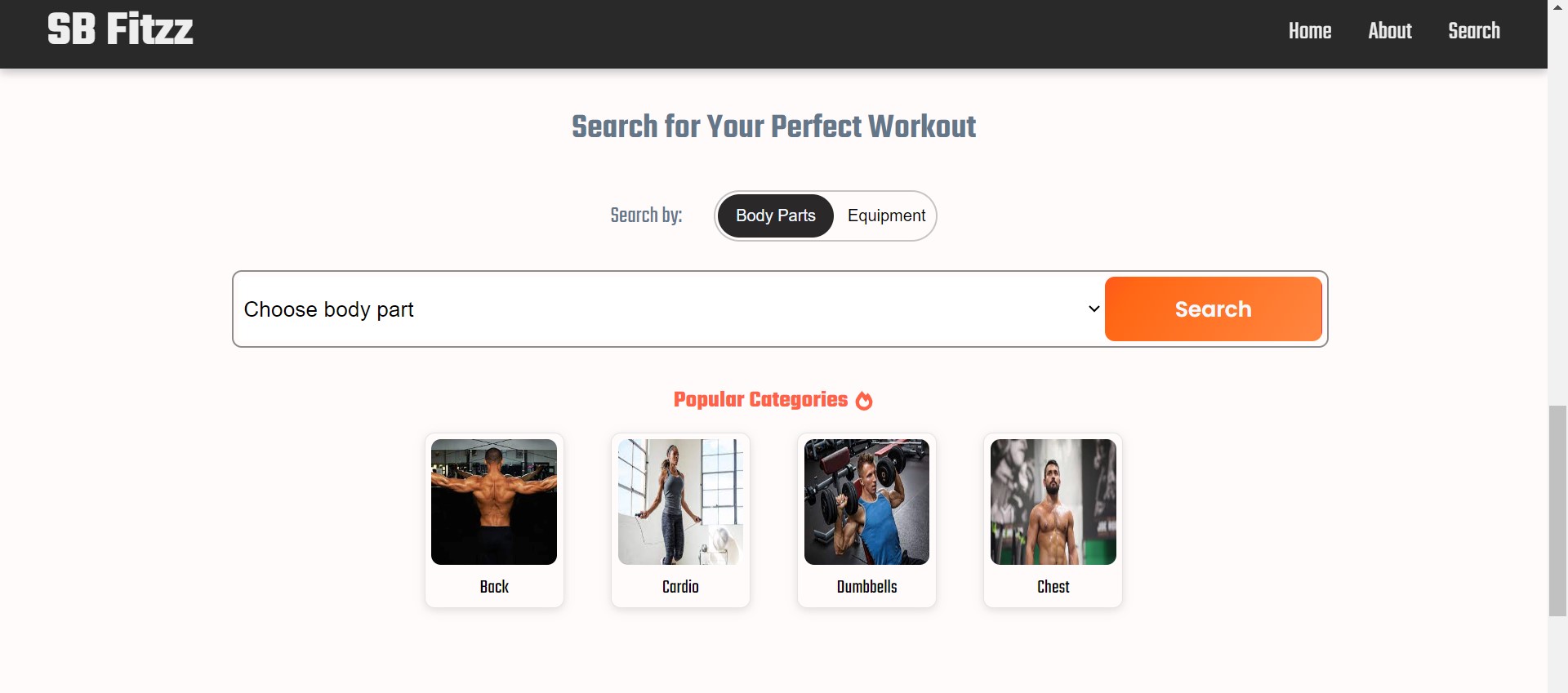
* **About**

FitFlex isn't just another fitness app. We're meticulously designed to transform your workout experience, no matter your fitness background or goals.



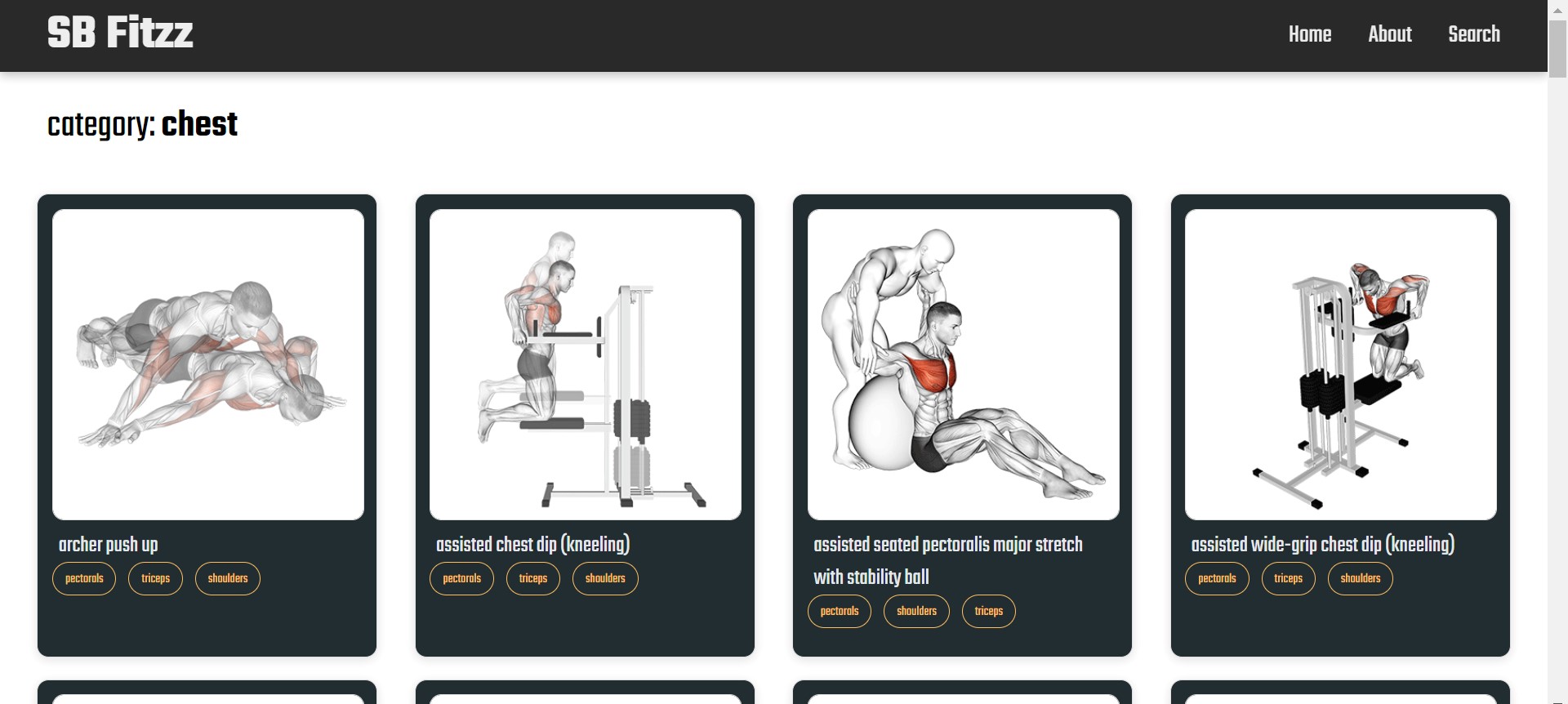
* **Search**

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**

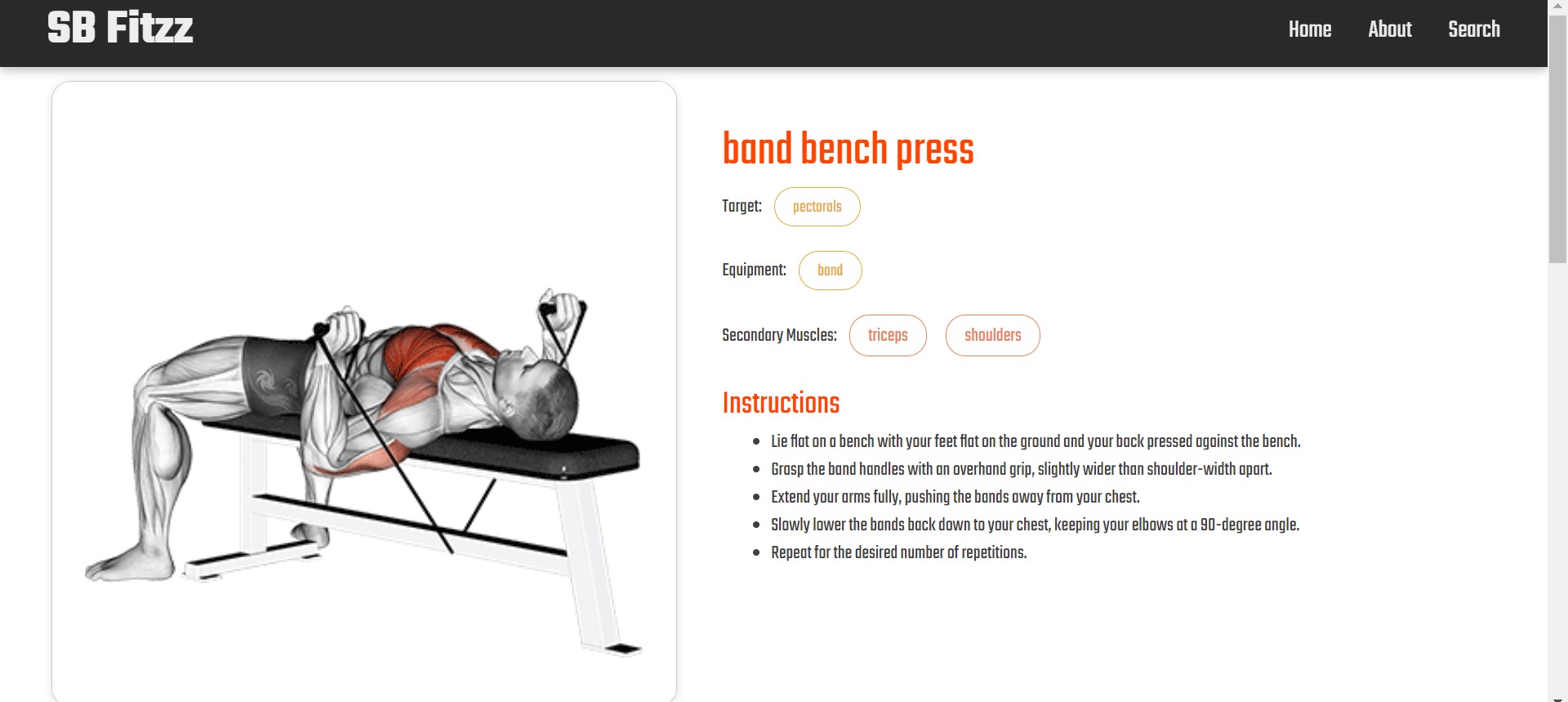
FitFlex would offer a dedicated section for browsing various workout categories. This could be a grid layout with tiles showcasing different exercise types (e.g., cardio, strength training, yoga) with icons or short descriptions for easy identification.



* **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.



**CONCLUSION:**

The evolution of fitness trackers reflects a growing emphasis on holistic health management, personalized insights, and seamless integration with digital ecosystems. Modern devices go beyond basic step counting, incorporating advanced health metrics such as heart rate variability, stress levels, and sleep tracking.

With the rise of AI-powered coaching, diverse form factors, and stylish designs, fitness trackers are now more than just activity monitors—they are comprehensive health companions. Their integration with smartphones, wearables, and digital payment systems enhances user convenience, making them indispensable in daily life.

As technology continues to advance, fitness trackers will likely become even more personalized, predictive, and interconnected, further empowering individuals to take control of their health and well-being.