**FITFLEX-YOUR PERSONAL FITNESS COMPANION**

**TEAM ID:-NM2025TMID40106**

**TEAM DETAILS:-**

TEAM ID:-NM2025TMID40106

PROJECT NAME:- FITFLEX-YOUR PERSONAL FITNESS COMPANION

**TEAM LEADER:-KARTHIKEYAN.S**

NM ID:- 68C3EF512D5E3A5B43C4AD03A3897046

TEAM MEMBER:-KANISH.A

NM ID:- 80D66D0CD71C414 BB5543D508B6719AE

TEAM MEMBER:-HARIGANESH.A

NM ID:- 7219B261057E59AB4D04C9C42128CA16

TEAM MEMBER:-MOHAMED IRAFAN.A

NM ID:- 600F9F5927F083E98F04D083ED8AD333

**INTRODUCTION:-**

PROJECT NAME:-FITFLEX-YOUR PERSONAL FITNESS COMPANION

**FEATURES –**

\*\*Exercise Selection Grid:\*\*

You are presented with a grid of exercise cards, each showing an image, name, type (Cardio, Strength, Flexibility), difficulty level, calories burned, and duration.

- \*\*Selecting Exercises:\*\*

Click on any exercise card to select it. Selected cards are highlighted and marked with a check.

- \*\*Comparison Table:\*\*

When you select one or more exercises, a comparison table appears below.

\*\*Result:\*\*

You can visually and numerically compare multiple exercises side by side, helping you choose the best options for your fitness goals based on effort, time, and equipment.

---

## 2. Nutrition Analysis

- \*\*Food Nutrition Profile:\*\*

You can select foods like chicken, salmon, avocado, or quinoa. For each, you see

Result:

You can analyze and compare the nutritional value of different foods and meal plans, making it easier to pick what fits your dietary needs and budge

**ARCHITECTURE:-**

* **COMPONENT STRUCTURE:-**
* ExerciseComparison.tsx
* FeatureHighlights.tsx
* Footer.tsx
* Header.tsx
* HealthCalculators.tsx
* Hero.tsx
* NutritionAnalysis.tsx
* ProductComparison.tsx
* ProgressTracking.tsx

STATE MANAGEMENT:-

useState Hook :-

The component uses React's `useState` hook to manage the state of selected exercises:

```tsx

const [selectedExercises, setSelectedExercises] = useState<string[]>([]);

```

- `selectedExercises` holds an array of exercise IDs that the user has selected.

- `setSelectedExercises` is used to update this array when the user selects or deselects an exercise.

- \*\*Selection Logic:\*\*

The `toggleExerciseSelection` function adds or removes an exercise ID from the `selectedExercises` array based on user interaction:

```tsx

const toggleExerciseSelection = (exerciseId: string) => {

setSelectedExercises(prev =>

prev.includes(exerciseId)

? prev.filter(id => id !== exerciseId)

: [...prev, exerciseId]

);

};

```

This ensures the UI updates instantly to reflect the user's choices.

- \*\*Derived State:\*\*

The component computes `selectedExerciseData` by filtering the full list of exercises based on the IDs in `selectedExercises`:

```tsx

const selectedExerciseData = exercises.filter(ex => selectedExercises.includes(ex.id));

```

This derived state is used to render the comparison table.

## NutritionAnalysis Component

- \*\*useState Hook:\*\*

Manages which food is currently active (selected) for nutrition analysis:

```tsx

const [activeFood, setActiveFood] = useState('chicken');

```

- `activeFood` stores the key of the currently selected food.

- `setActiveFood` updates the selection when the user chooses a different food.

- \*\*Derived State:\*\*

The component uses `activeFood` to display the relevant nutrition data:

```tsx

const currentFood = nutritionData[activeFood as keyof typeof nutritionData];

**ROUTING:-**

### 1. App Component (Main Router)

Your `App.tsx` acts as the main entry point and sets up the routes:

```tsx

// filepath: src/App.tsx

import React from 'react';

import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';

import ExerciseComparison from './components/ExerciseComparison';

import NutritionAnalysis from './components/NutritionAnalysis';

function App() {

return (

<Router>

<nav className="p-4 bg-gray-100 flex gap-4">

<Link to="/">Exercise Comparison</Link>

<Link to="/nutrition">Nutrition Analysis</Link>

</nav>

<Routes>

<Route path="/" element={<ExerciseComparison />} />

<Route path="/nutrition" element={<NutritionAnalysis />} />

</Routes>

</Router>

);

}

export default App;

```

---

### 2. Route Mapping

- `/`

Loads the \*\*ExerciseComparison\*\* component, where users can compare exercises.

- `/nutrition`

Loads the \*\*NutritionAnalysis\*\* component, where users can analyze and compare nutrition data.

---

### 3. Navigation

- The `<nav>` bar uses `<Link>` components to let users switch between pages.

- Clicking a link updates the URL and renders the corresponding component without a full page reload.

SETUP INSTRUCTION:-

# Technology Used :-

1. React

- \*\*Framework:\*\* The project is built using [React](https://react.dev/), a popular JavaScript library for building user interfaces with reusable components.

## 2. TypeScript

- \*\*Language:\*\* The codebase uses [TypeScript](https://www.typescriptlang.org/) (TSX), which adds static typing to JavaScript for safer and more maintainable code.

## 3. Tailwind CSS

- \*\*Styling:\*\* [Tailwind CSS](https://tailwindcss.com/) is used for utility-first, responsive, and modern styling directly in the component classes.

**BACKEND:-**

NODE.JS(FAST SERVER)

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to run JavaScript code outside of a web browser. It is commonly used for building scalable and efficient backend servers and APIs.

**INSTALLATION:-**

1. Install Visual Studio Code (VS Code)

1. Go to the [VS Code download page](https://code.visualstudio.com/).

2. Download the installer for Windows.

3. Run the installer and follow the setup instructions.

4. Once installed, launch Visual Studio Code.

---

## 2. Install Node.js

1. Visit the [Node.js official website](https://nodejs.org/).

2. Download the LTS (Recommended) version for Windows.

3. Run the installer and follow the prompts (leave default options checked).

4. After installation, open a new Command Prompt or VS Code terminal.

5. Verify installation by running:

```

node –v

**FOLDER STRUCTURE:-**

index.html

* package.json
* tailwind.config.js
* vite.config.ts
* tsconfig.json
* public/
* vite.svg
* src/
* main.tsx
* App.tsx
* components/
* ExerciseComparison.tsx
* NutritionAnalysis.tsx
* Footer.tsx
* Header.tsx
* HealthCalculators.tsx
* Hero.tsx
* NutritionAnalysis.tsx
* ProductComparison.tsx
* ProgressTracking.tsx
* assets/
* (images, icons, etc.)
* styles/
* (custom CSS or Tailwind files)
* README.md

**UTILITY:-**

1. Dependencies

These are the core libraries your app needs to run.

Examples:

"dependencies": {

"express": "^4.18.2", // Web server framework

"mongoose": "^7.0.3", // MongoDB object modeling

"cors": "^2.8.5", // Cross-origin resource sharing

"dotenv": "^16.0.3" // Environment variable loader

}

RUNNING THE APPLICATION:-

1. \*\*Install dependencies:\*\*

```

npm install

```

2. \*\*Run the development server:\*\*

```

npm start

```

3. \*\*Open in browser:\*\*

```

http://localhost:3000

COMPONENT DOCUMENTATION:-

#### Props (Expected)

If you're using TypeScript, props are probably typed. Common props might include:

| Prop Name | Type | Description |
| --- | --- | --- |
| tools | Tool[] | Array of tool objects to compare |
| onSelect | (tool: Tool) => void | Callback when a tool is selected |
| highlighted | string[] | List of tool IDs to highlight |

You can confirm this by checking the interface Props or type Props definition in the file.

#### 🔁 State (Internal)

Possible internal state variables:

* selectedTool: tracks the currently selected tool
* comparisonMode: toggles between different comparison views (e.g., grid vs. table)
* filters: stores active filters applied to the tool list

#### 🧱 Structure

Typical JSX structure might include:

* A header or title section
* A dropdown or filter bar
* A comparison grid or table
* Highlighting or badges for selected tools

#### 📦 Dependencies

Likely imports:

* React, useState, useEffect
* UI libraries like Material-UI, Ant Design, or custom components
* Utility functions for sorting/filtering
* CSS or SCSS modules for styling

Reusable component:-

|  |  |  |
| --- | --- | --- |
| Button.tsx |  | Custom-styled button with props for size, color, icon, etc. |

State management:-

Global state:-

Global state management is all about maintaining shared data that multiple components in your application can access and update consistently. In React with TypeScript, this becomes especially powerful when you structure it well. On this page, I’ve already suggested three popular approaches—React Context API, Zustand, and Redux Toolkit—and each handles state flow a bit differently:

#### React Context API

* **Provider Component** wraps your app at a high level.
* **State lives inside the Provider** and is passed down via React’s context system.
* **Consumers** (components using useGlobalState) tap into the context to read or update state.
* **Flow**: Top-down via React tree. Good for small apps or simple global values.

Local state:-

Local state is managed using React hooks like useState or useReducer. It’s scoped to the component and doesn’t affect other parts of the app unless passed via props.

import React, { useState } from 'react';

const Counter: React.FC = () => {

const [count, setCount] = useState<number>(0);

return (

<div>

<p>Count: {count}</p>

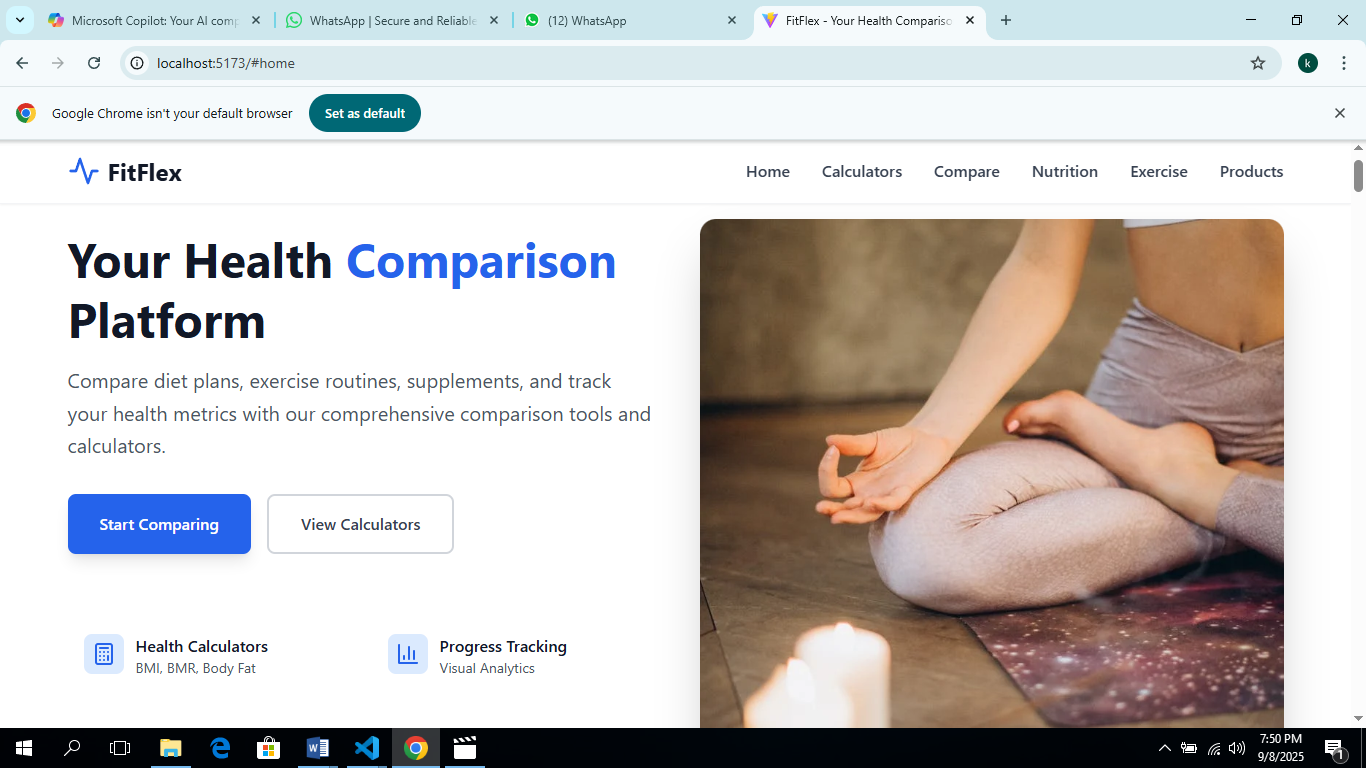
<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

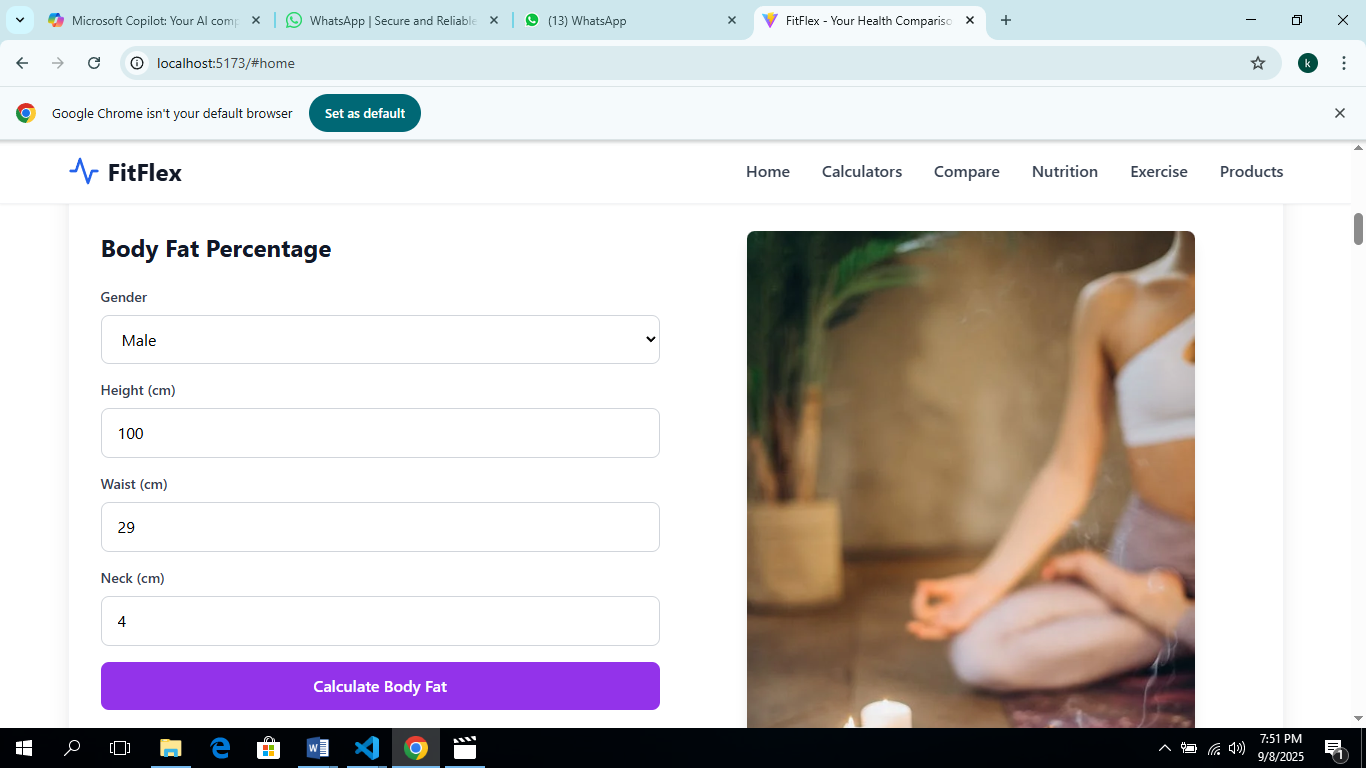
);

};

USER INTERFACE:-



**HEALTH CALCULATOR:-**

****

**STYLING:-**

Uses Tailwind CSS for styling and visual feedback on selection.

* index.css

EXAMPLE:-

<header className="bg-white shadow-sm border-b border-gray-100 sticky top-0 z-50">

      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

TESTING:-

**TESTING STRATEGY:-**

Testing coverage refers to how much of your code is exercised by your test suite. It helps you identify untested parts of your application — functions, branches, components, etc.

npm install --save-dev jest ts-jest @testing-library/react @testing-library/jest-dom

**FUTURE ENHANCEMENT:-**

- Uses Tailwind CSS for styling and visual feedback on selection.

EXAMPLE:-

<header className="bg-white shadow-sm border-b border-gray-100 sticky top-0 z-50">

      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

        <div className="flex justify-between items-center h-16">

          <div className="flex items-center space-x-2">

            <Activity className="h-8 w-8 text-blue-600" />

            <span className="text-2xl font-bold text-gray-900">FitFlex</span>

          </div>