Project Documentation

1. Introduction

Project Title : FitFlex: The Personal Fitness Companion
Team leader: Dinesh kumar t
Team member: Deepak k
Team member: Devasagayam v
Team member: Eswar v

2. Project Overview

Purpose : The purpose of FitFlex is to provide users with a centralized, intuitive, and responsive web application to manage their personal fitness journey. The primary goals are to allow users to track their daily workouts, log nutritional intake, monitor their progress over time through visual charts, and set personalized fitness goals, thereby promoting a healthier and more organized lifestyle.

Features : The key features of the FitFlex frontend include a user-friendly dashboard for an overview of daily activities, a workout tracker to log and review exercise routines, a meal logger to record daily calorie and macronutrient intake, a progress page that displays historical data using charts, and a user profile section for managing account settings and personal goals.

3. Architecture

Component Structure: The application is built with a component-based architecture. The top-level `App` component manages routing and global state providers. It renders layout components such as `Header` and `Sidebar` for navigation. The main content area conditionally renders page-level components like `Dashboard`, `WorkoutTracker`, `MealLogger`, `Progress`, and `Profile` based on the current route. These page components are composed of smaller, reusable presentational components like `Card`, `Button`, `Modal`, and `Chart`.

State Management : For state management, FitFlex utilizes the Context API combined with the `useReducer` hook to handle global application state, such as user authentication status, workout data, and meal history. This approach was chosen for its simplicity and built-in support within React, avoiding the overhead of additional libraries for a project of this scale. Local UI state, such as form inputs and modal visibility, is managed within individual components using the `useState` hook.

Routing : Client-side routing is implemented using the `react-router-dom` library. The routing structure defines paths for all major features, including '/dashboard' for the main dashboard, '/workouts' for the workout tracker, '/meals' for the meal logger, '/progress' for viewing statistics, and '/profile' for user settings. This

ensures a seamless, single-page application experience without full page reloads.

4. Setup Instructions

Prerequisites : To run this project, the following software must be installed on your machine: Node.js (version 16 or higher) and npm (which comes bundled with Node.js). A modern web browser like Chrome, Firefox, or Edge is also required.

Installation : To get the project running locally, begin by cloning the project repository from the source code management platform (e.g., GitHub) using the `git clone` command. Navigate into the project's root directory (`cd fitflex`) and install all necessary project dependencies by running the command `npm install`. This will download all required packages listed in the `package.json` file. Finally, configure any necessary environment variables, such as API keys for external services, in a `.env` file at the root of the project.

5. Folder Structure

Client : The React application is organized within a `src` folder. Key directories include: `components` for reusable UI components (further divided into `common` and `layout` subfolders), `pages` for top-level page components, `context` for React Context files related to state management, `assets` for static images and styles, `utils` for helper functions, and `hooks` for custom React hooks.

Utilities : The project employs several utility functions to promote code reuse and separation of concerns. This includes a `dateFormatter` function for consistent date handling across the app, a `calculateMacros` helper for nutritional calculations, and a `localStorage` utility for caching user data. A custom `useLocalStorage` hook is also used to easily synchronize state with the browser's local storage.

6. Running the Application

To start the development server and run the frontend application locally, navigate to the project's root directory in your terminal and execute the command `npm start`. This will compile the React application and automatically open it in your default web browser, typically at `http://localhost:3000`. The page will reload automatically if you make any edits to the source code, thanks to React's hot-reload feature.

7. Component Documentation

Key Components : The `WorkoutTracker` page component is responsible for rendering the form to log new workouts and displaying a list of past workouts. It receives props for the `workoutList` array and callback functions like

`onAddWorkout` and `onDeleteWorkout` to interact with the global state. The `ProgressChart` component is crucial for the Progress page; its purpose is to visualize user data (e.g., weight over time) and it receives a `data` prop containing an array of data points to be plotted.

Reusable Components : The `Modal` component is a reusable UI element used for displaying forms and messages in a overlay. It is configurable through props such as `isOpen` (boolean to control visibility), `onClose` (a callback function to close it), and `title` (a string for the modal header). The `Button` component is another reusable element that accepts props for `variant` (e.g., 'primary', 'secondary'), `size`, `onClick` handler, and `children` for its label.

8. State Management

Global State : The global state is managed using a combination of React Context and useReducer. A primary `AppStateContext` holds critical data such as the authenticated user's profile, their array of logged workouts, and their meal history. This state is accessible from any component in the tree, and updates are performed by dispatching actions (e.g., `ADD_MEAL`, `DELETE_WORKOUT`) to a central reducer function, which then updates the state and triggers a re-render of all consuming components.

Local State : Local component state is extensively used for UI interactions and form handling. For instance, the `MealLogger` component uses the `useState` hook to manage the current input values for the meal form (food name, calories, etc.) before the user submits it and the data is dispatched to the global state. Similarly, a `isLoading` state boolean might be used to control the display of a loading spinner while data is being fetched.

9. User Interface

The user interface of FitFlex is designed to be clean, modern, and user-friendly. It features a cohesive dashboard with summary cards, intuitive forms for data entry on the workout and meal logging pages, and interactive charts on the progress page. The design emphasizes clarity and ease of use to ensure a positive user experience.

10. Styling

CSS Frameworks/Libraries : The application's styling is primarily implemented using CSS Modules for component-scoped styles, ensuring that styles do not leak between components. This is complemented by the Styled-Components library for building more complex, dynamic styled elements that require props to change their appearance.

Theming : A basic custom design system is implemented using CSS custom properties (CSS variables) for consistent theming. This includes variables for a

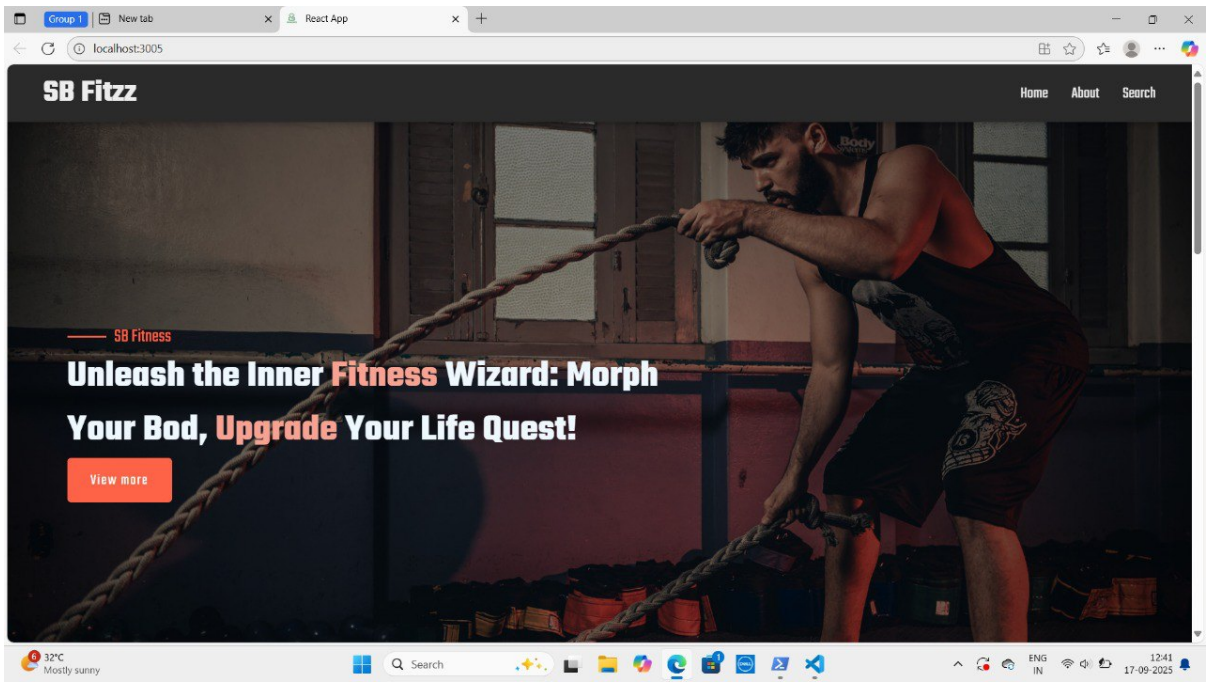
primary color palette, font stack, spacing scale, and border-radius. This allows for easy future modifications to the overall look and feel of the application from a single source.

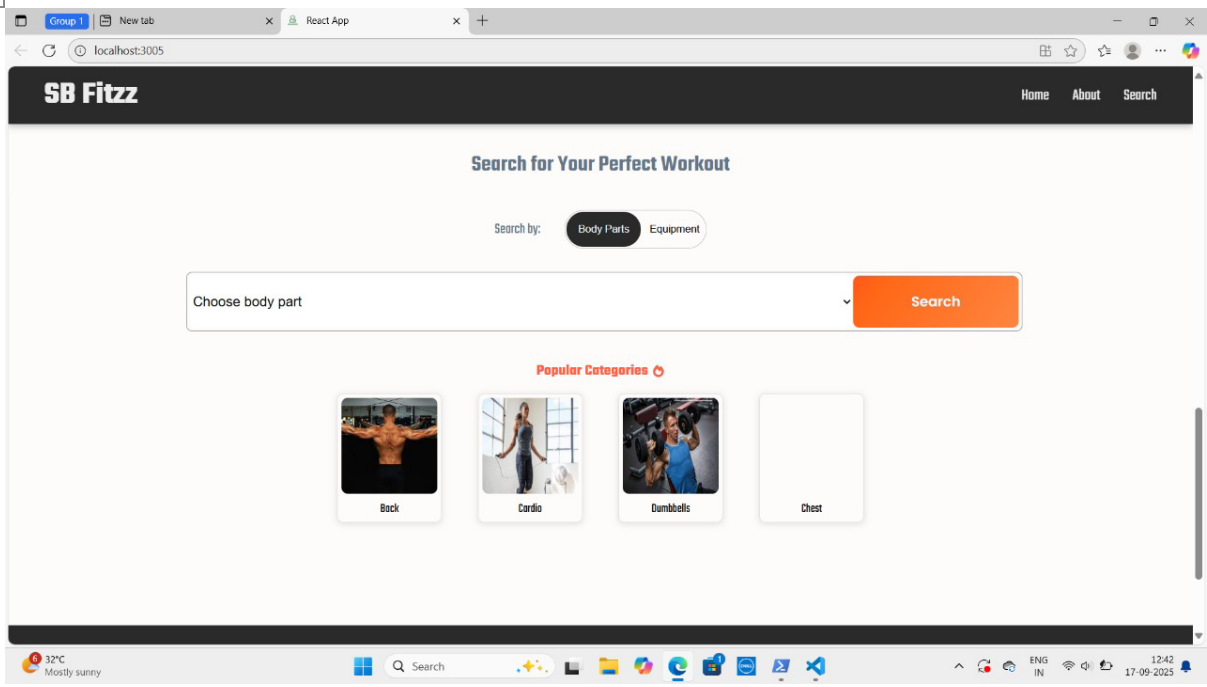
11. Testing

Testing Strategy : The testing strategy for FitFlex focuses on writing unit tests for individual React components and utility functions using Jest and React Testing Library. Tests are written to ensure that components render correctly, respond to user interactions (like button clicks and form submissions), and integrate properly with Context for state management.

Code Coverage : The Jest test runner is configured to generate code coverage reports. The goal is to maintain a high level of test coverage, particularly for critical utility functions, complex components, and state management logic, to ensure application reliability and facilitate future refactoring.

12. Screenshots





13. Known Issues

Currently, there is a known issue where the chart on the Progress page does not automatically re-render immediately after a new data point is added; a manual refresh of the page is required to update the visualization. Additionally, the application's performance may slightly degrade when a very large history of workouts and meals is loaded into the state.

14. Future Enhancements

Potential future enhancements for FitFlex include implementing user authentication and backend integration for data persistence, adding social features like sharing progress with friends, incorporating a library for advanced animations to improve UX, developing a mobile app version using React Native, and expanding the meal logger with a food database API for automatic nutritional lookup.