Project Document

Fit Flex Your Personal Fitness Companion

# Introduction

## Project Title: Fit Flex

* + **Team ID:NM2025TMID47763**
  + **Team Leader:** Divya.K & [dhivyadhivya1932007@gmail.com](mailto:dhivyadhivya1932007@gmail.com)

## Team Members:

* + - Durga.S [&durganaveen2206@gmail.com](mailto:&durganaveen2206@gmail.com)
    - Ezhil.T [&ezhil26204@gmail.com](mailto:&ezhil26204@gmail.com)
    - Janani.G [&jananigjanani545@gmail.com](mailto:&jananigjanani545@gmail.com)

# Project Overview

**Purpose:** 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:

* + **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

The system is built using a full-stack JavaScript architecture, ensuring seamless integration between the frontend, backend, and database components.

#### **Frontend**

* **Framework**: React.js
* **UI Libraries**: Bootstrap, Material UI
* **Description**: The client-side interface is developed using React.js for efficient component-based rendering and state management. Bootstrap and Material UI are integrated to provide responsive, consistent, and visually appealing design components.

#### **Backend**

* **Runtime & Framework**: Node.js with Express.js
* **Description**: The server is built with Node.js, using Express.js to handle routing, middleware, and API endpoint management. It processes requests from the frontend, communicates with the database, and handles business logic.

#### **Database**

* **Database**: MongoDB (NoSQL)
* **Description**: MongoDB is used as the primary data store. It holds structured collections for:
  + User profiles and authentication
  + Project listings and metadata
  + Application submissions and status
  + Real-time chat messages between users

Top of Form

Bottom of Form

# Setup Instructions

* **Installation of required tools**:

1. 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>

# Folder Structure

The project is organized into separate frontend and backend directories for clear separation of concerns and easier maintainability.

SB-Works/

│

├── client/ # React frontend application

│ ├── components/ # Reusable UI components (buttons, forms, etc.)

│ ├── pages/ # Page-level React components (routes/views)

│ ├── assets/ # Static assets like images, stylesheets

│ ├── services/ # API service functions to interact with backend

│ └── App.js # Main React component

│

├── server/ # Node.js backend application

│ ├── routes/ # Express route handlers (e.g., /users, /projects)

│ ├── models/ # Mongoose schemas/models for MongoDB

│ ├── controllers/ # Business logic for handling requests

│ ├── middleware/ # Custom Express middleware (e.g., auth)

│ ├── config/ # Configuration files (e.g., DB connection)

│ ├── utils/ # Utility functions (e.g., validation, helpers)

│ └── index.js # Entry point of the backend server

│

├── .env # Environment variables for backend

├── package.json # Root package (optional: separate for client/server)

└── README.md # Project documentation

# Folder Structure

SB-Works/

│

├── client/ # React frontend

│ ├── components/ # Reusable UI components (e.g., Header, Button)

│ └── pages/ # Route-level pages (e.g., Home, Login, Dashboard)

│

├── server/ # Node.js backend

│ ├── routes/ # Express route definitions

│ ├── models/ # Mongoose schemas for MongoDB

│ └── controllers/ # Request handlers and business logic

# Folder Structure

SB-Works/

│

├── client/ # React frontend

│ ├── components/ # Reusable UI components

│ └── pages/ # Page-level components (views/routes)

│

└── server/ # Node.js backend

├── routes/ # Express route definitions

├── models/ # Mongoose schemas for MongoDB

└── controllers/ # Logic handling for each route

# Running the Application

Follow these steps to start both the frontend and backend servers:

#### **Frontend**

cd client

npm start

* This will start the React app.
* By default, it runs on: <http://localhost:3000>

#### **Backend**

cd server

npm start

* This will start the Node.js/Express server.
* By default, it runs on: <http://localhost:5000>

Top of Form

Bottom of Form

# API Documentation

This section outlines the main REST API endpoints exposed by the backend.

#### **User Routes**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /api/user/register | Register a new user |
| POST | /api/user/login | Log in an existing user |

#### **Project Routes**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /api/projects | Get a list of all projects |
| POST | /api/projects | Create a new project |
| GET | /api/projects/:id | Get details of a specific project |
| PUT | /api/projects/:id | Update an existing project |
| DELETE | /api/projects/:id | Delete a specific project |

#### **User Routes**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /api/user/register | Register a new user |
| POST | /api/user/login | Log in an existing user |

#### **Project Routes**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /api/projects/create | Create a new project |
| GET | /api/projects/:id | Get details of a specific project |
| GET | /api/projects (optional) | Get a list of all projects |

#### **Application Routes**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /api/apply | Submit an application to a project |

#### **Chat Routes**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /api/chat/send | Send a chat message |
| GET | /api/chat/:userId | Get chat history with a specific user |

### Notes:

* All POST requests are expected to receive JSON payloads.
* Authentication may be required for some routes (e.g., project creation, chat).
* Consider using middleware for JWT-based auth (Authorization: Bearer <token>).

Existing solutions

Git Hub Link[ <https://github.com/dhivyadhivya1932007-cpu/Fit-Flex-Your-Personal-Fitness-Companion.git> ]

# Authentication

**JWT-Based Authentication:**  
The system uses JSON Web Tokens (JWT) to securely authenticate users during login. Upon successful login, the server issues a signed JWT token that the client stores (usually in localStorage or cookies).

**Middleware for Route Protection:**  
Backend routes that require authentication are protected by middleware that:

* + Verifies the presence and validity of the JWT token in incoming requests.
  + Decodes the token to identify the user.
  + Denies access if the token is missing, expired, or invalid.

This ensures only authorized users can access private resources such as project management, applications, and chat features.

# User Interface

The application offers a user-friendly interface tailored to different user roles and functionalities, including:

* **Landing Page**  
  The homepage that introduces the platform, showcases featured projects, and provides navigation options for users to register or log in.
* **Freelancer Dashboard**  
  A personalized dashboard where freelancers can:
  + View and manage their active projects and applications.
  + Track messages and notifications.
  + Update their profiles and preferences.
* **Admin Panel**  
  A control panel for administrators to:
  + Manage users, projects, and applications.
  + Monitor platform activity and metrics.
  + Moderate content and handle reports.
* **Project Details Page**  
  A detailed view of individual projects including:
  + Project description, requirements, and deadlines.
  + Application status and submit options.
  + Related chat or communication channels.

Top of Form

Bottom of Form

# Testing

 **Manual Testing:**  
The application undergoes manual testing during key project milestones to ensure features meet requirements and function correctly across different user scenarios.

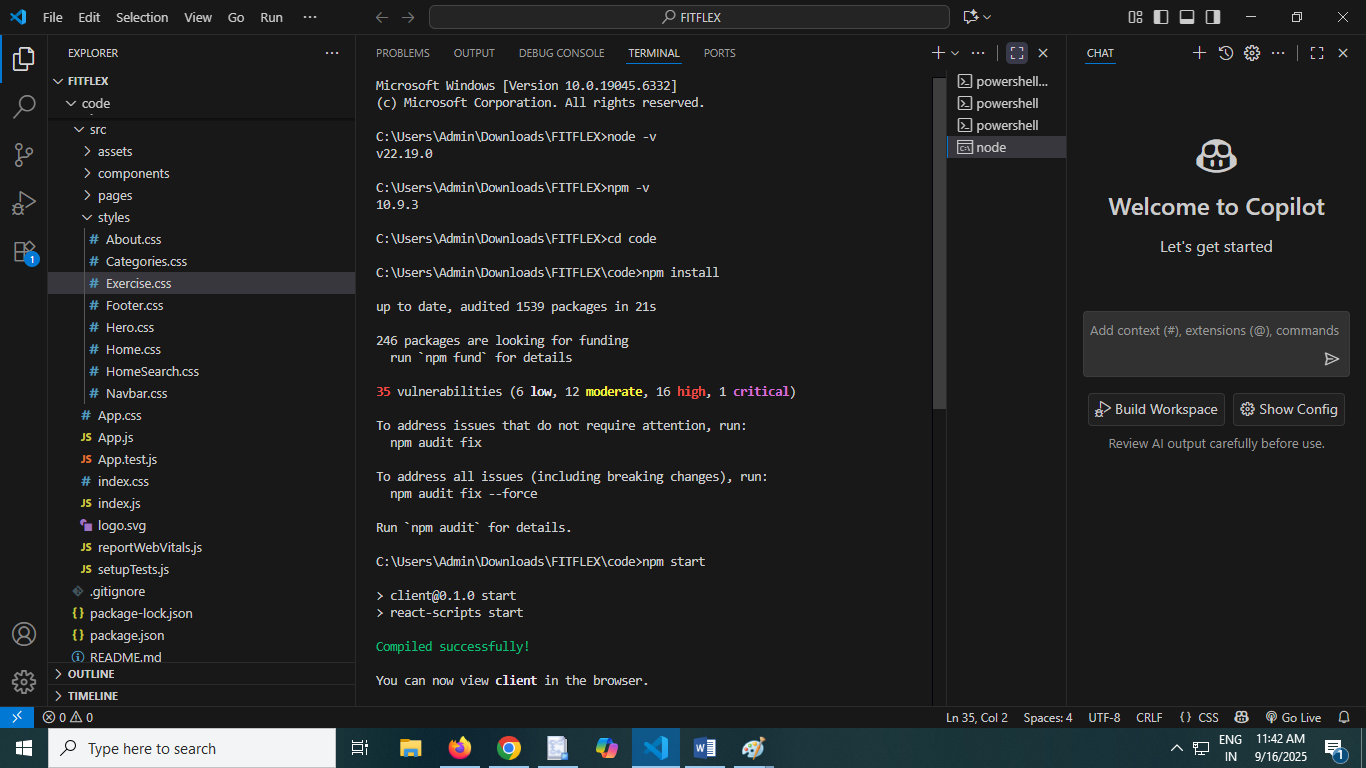
 **Testing Tools:**

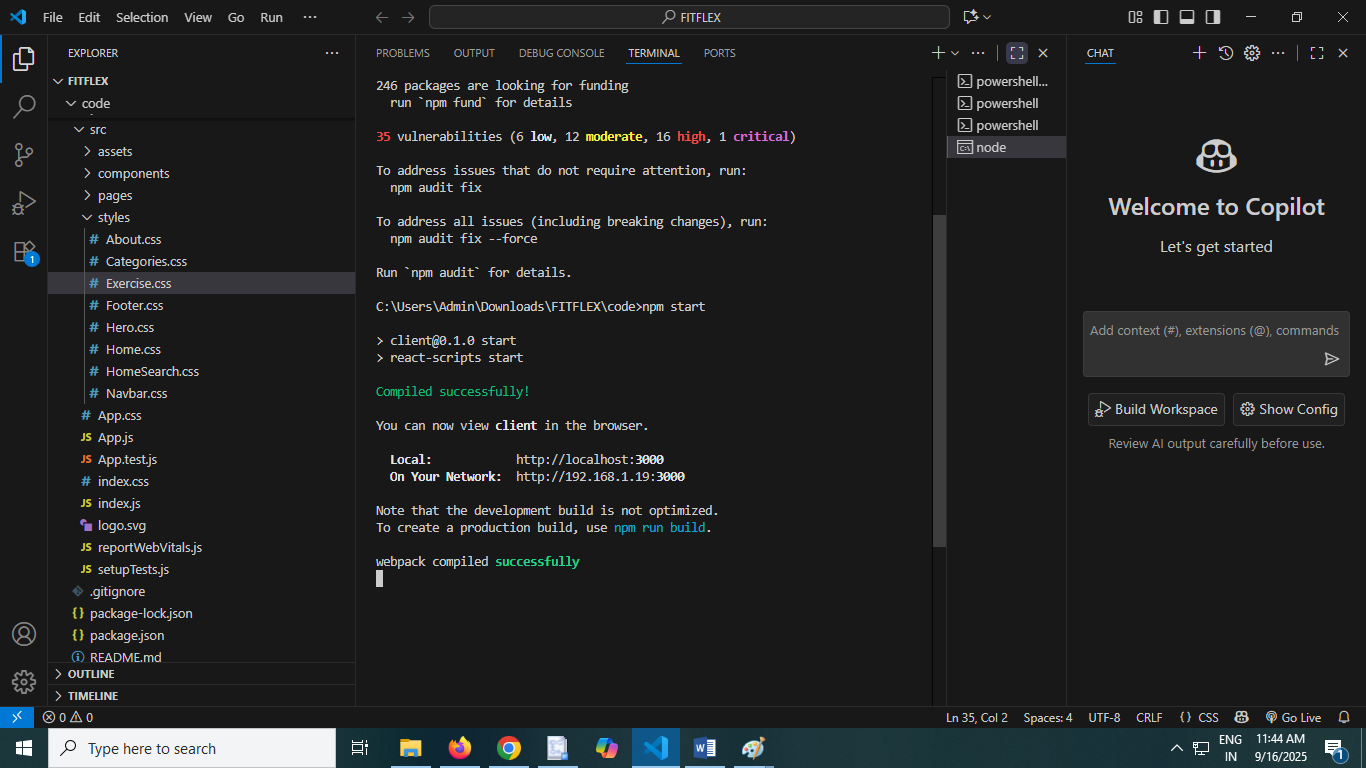
* **Postman:** Used for testing and validating backend API endpoints, including request payloads, responses, and error handling.
* **Chrome Dev Tools:** Utilized for frontend debugging, inspecting network requests, performance monitoring, and troubleshooting UI issues.

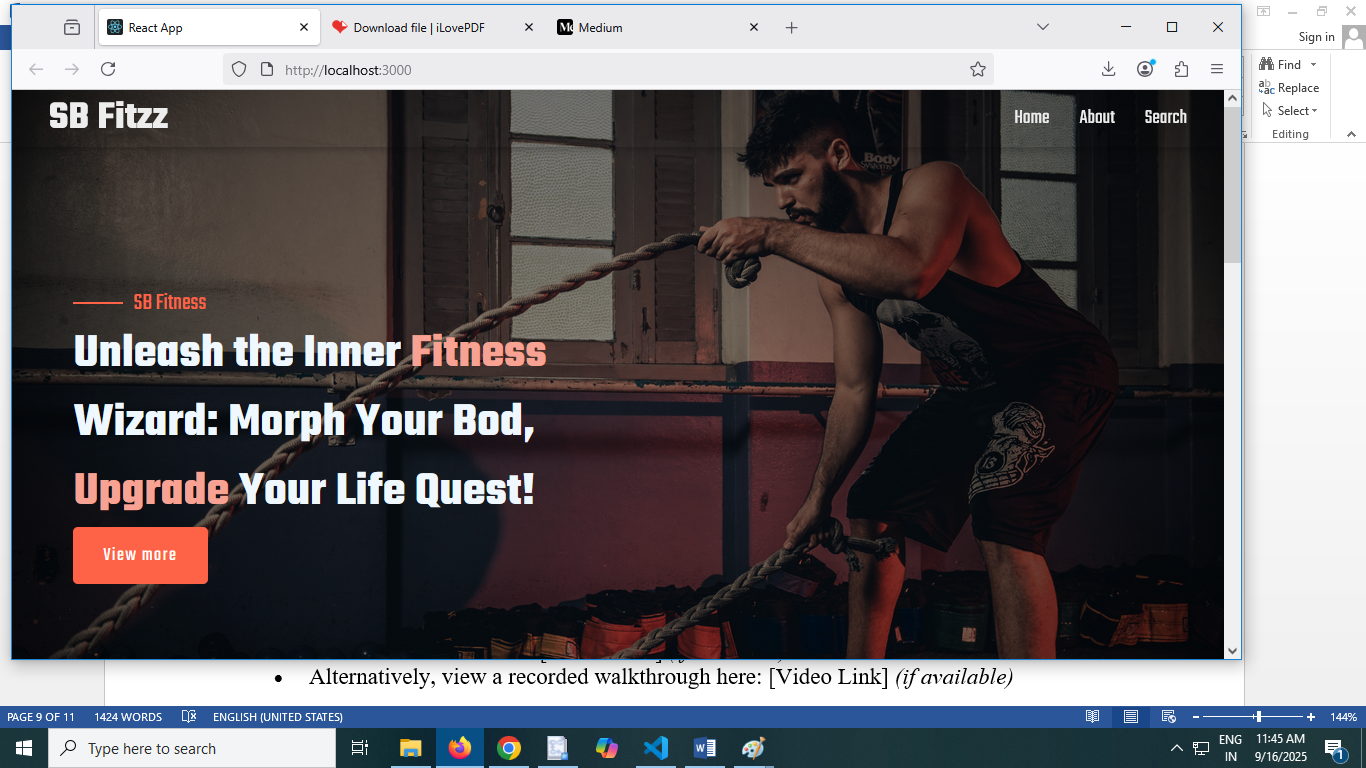
1. **Screenshots or Demo**

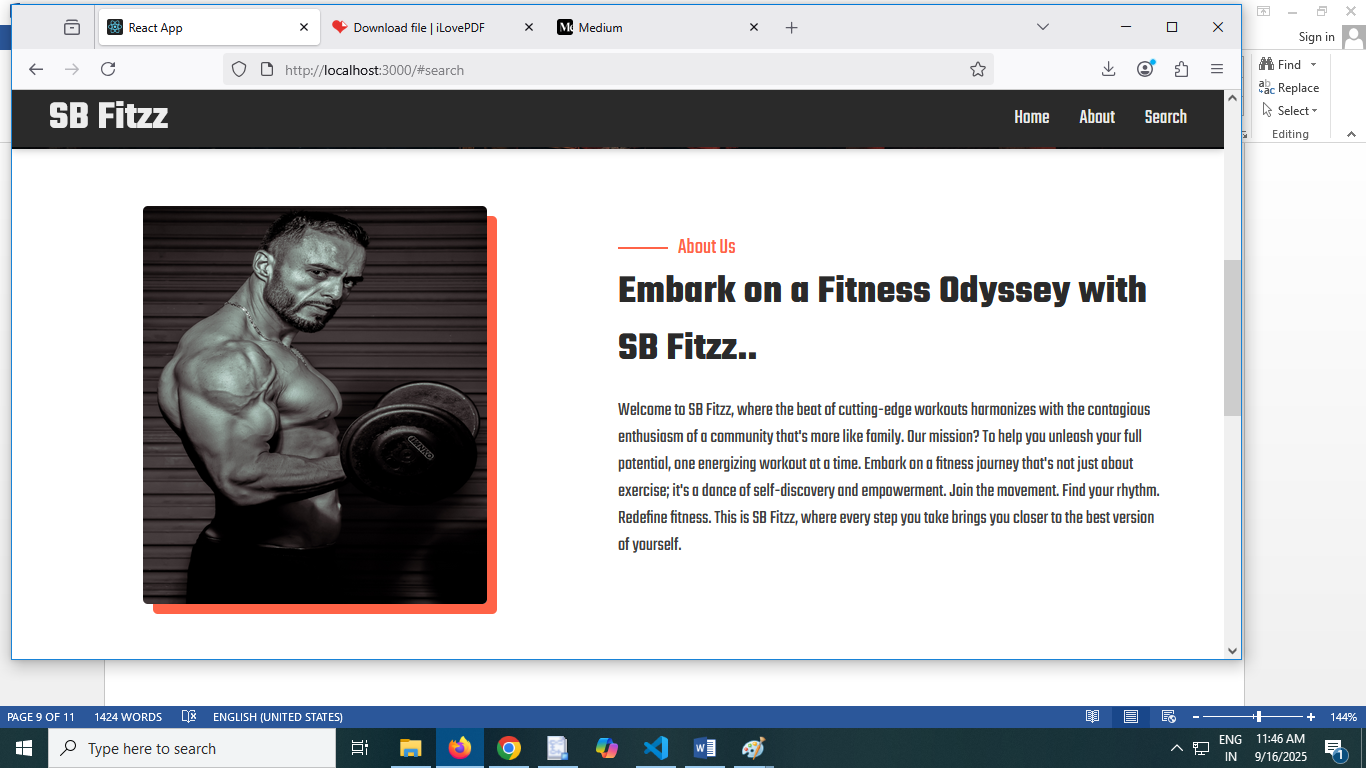
To provide a visual overview of the application, below are screenshots and/or a demo link showcasing key features and interfaces:

#### **Screenshots**









#### **Demo**

* Access the live demo at: [Demo URL] (if available)
* Alternatively, view a recorded walkthrough here: [Video Link] (if available)

1. **Known Issues**

* **Issue #1:**  
  Description: [Briefly describe the issue]  
  Status: [Open/In Progress/Planned Fix]  
  Impact: [Low/Medium/High]  
  Workaround: [If any]
* **Issue #2:**  
  Description: [Briefly describe the issue]  
  Status: [Open/In Progress/Planned Fix]  
  Impact: [Low/Medium/High]  
  Workaround: [If any]

**Example:**

* **User login occasionally times out on slow connections**  
  Status: Open  
  Impact: Medium  
  Workaround: Refresh the page and try again.
* **Chat messages sometimes fail to load immediately after sending**  
  Status: In Progress  
  Impact: Low  
  Workaround: Reload the chat window.

Top of Form

Bottom of Form

1. **Future Enhancements**

Future enhancements aim to improve the overall performance, usability, and functionality of the project. Potential areas of development include:

* **Feature Expansion:** Adding new capabilities based on user feedback and emerging needs to increase the system’s utility.
* **Performance Optimization:** Refining algorithms and infrastructure to improve speed, efficiency, and scalability.
* **User Interface Improvements:** Enhancing the design and accessibility to provide a more intuitive and seamless user experience.
* **Security Enhancements:** Implementing advanced security measures to protect data and ensure compliance with updated standards.
* **Integration:** Enabling compatibility with additional platforms, tools, or APIs for extended functionality.
* **Technology Upgrades:** Incorporating emerging technologies to keep the system modern and competitive.

These enhancements will be prioritized based on user requirements and technological advancements to ensure continuous improvement.

Top of Form

Bottom of Form