**FIT FLEX: YOU’RE PERSONAL FITNESS COMPANION PROJECT DOCUMENTATION**

1**. INTRODUCTION**:

* Project Title : **Fit Flex: You’re personal fitness companion**
* Team ID : NM2025TMID41364
* TREM LEADER : MYLESH R (myleshr2006@gmail.com)
* ROLE : CODING AND DEVALOPMENT
* TEAM MEMBER : DHARNESSH R (saghul57@gmail.com)
* ROLE : CODING AND DEVALOPMENT
* TEAM MEMBER : SYEDABUDHAHIR M (lovelyabu28@gmail.com)
* ROLE : DEMO VIDEO
* TEAM MEMBER : MAHESHWARAN M (wmathes508@gmail.com)
* ROLE : DOCUMENT CREATER

# 2. PROJECT OVERVIEW

* + **Purpose:**The purpose of the fitflex to provide auser friendly fitness platform that helps individuals achieve their health and wellness goals.

## Features:

* + - Personalize workout plan
    - Nutrition guidance
    - Progress tracking
    - Flexible scheduling

# 3. ARCHITECTURE

* **Frontend: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/>
* Installation instructions: <https://nodejs.org/en/download/package-manager/>
* **Backend:**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.
* **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.
  + **Database:**MongoDBstoresuserdata,projectinformation, applications, andchat messages

# 4. SETUP INSTRUCTIONS

## Prerequisites:

## Node.js and npm

## React.js

## HTML, CSS, and JavaScript

## Version Control

## Development Environment

**Get the code:**

**Start the Development Server**:

## InstallationSteps

* **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>

# 5. FOLDER STRUCTURE

# fitflex/

# ├── client/ # Frontend (React)

# │ ├── public/ # Static assets (index.html, icons, etc.)

# │ ├── src/

# │ │ ├── assets/ # Images, fonts, etc.

# │ │ ├── components/ # Reusable UI components (Navbar, Button, etc.)

# │ │ ├── pages/ # Page-level components (Dashboard, Login, etc.)

# │ │ ├── services/ # API calls (e.g., axiosconfigs)

# │ │ ├── context/ # React context providers (Auth, Theme)

# │ │ ├── hooks/ # Custom React hooks

# │ │ ├── utils/ # Helper functions

# │ │ ├── App.js # Main app component

# │ │ └── index.js # Entry point

# │

# ├── server/ # Backend (Node.js/Express)

# │ ├── controllers/ # Route logic (e.g., authController.js)

# │ ├── models/ # Mongoose/Sequelize models

# │ ├── routes/ # API endpoints

# │ ├── middleware/ # Auth, error handling, etc.

# │ ├── config/ # DB config, environment setup

# │ ├── utils/ # Server-side helpers

# │ ├── server.js # Entry point

# │ └── .env # Environment variables

# │

# ├── database/ # DB scripts, migrations, seeders

# │ └── init.js

# │

# ├── docs/ # Documentation (API specs, architecture)

# │

# ├── tests/ # Unit and integration tests

# │ ├── client/ # Frontend tests

# │ └── server/ # Backend tests

# │

# ├── .gitignore

# ├── package.json # Project metadata and scripts

# └── README.md # Project overview

# 6. RUNNING THE APPLICATION

#### 1. **Frontend (React or similar)**

bash

cd client

npm install # Only needed the first time or after adding dependencies

npm start

This will typically launch the frontend on http://localhost:3000 unless configured otherwise.

#### 2. **Backend (Node.js/Express)**

bash

cd server

npm install # Again, only needed initially or after changes to dependencies

npm start

This usually starts the backend server on http://localhost:5000 or another port defined in your configuration.

🔧 Make sure your backend is running before you try to interact with APIs from the frontend.

Let me know if you want help setting up environment variables, connecting to a database, or deploying this project!

* + **Access:**Visit[http://localhost:3000](http://localhost:3000/)

# 7. API DOCUMENTATION HTTPS:

### 📘 API Endpoints Overview

#### 👤 User

* POST /api/user/register — Register a new user
* POST /api/user/login — Authenticate an existing user

#### 📁 Projects

* POST /api/projects/create — Create a new project
* GET /api/projects/:id — Retrieve details of a specific project by ID

#### 💬 Chats

* POST /api/chat/send — Send a chat message
* GET /api/chat/:userId — Retrieve chat history for a specific user

#### 📝 Applications

* POST /api/apply — Submit an application

# 8. AUTHENTICATION

### 🔐 JWT-Based Authentication

**JWT (JSON Web Token)** is a compact, URL-safe token used to securely transmit information between parties. It’s commonly used for authentication in web apps.

#### ✅ How it works:

* **Login**: User submits credentials (e.g., email and password).
* **Token Generation**: Server verifies credentials and issues a JWT containing user info (e.g., user ID, role).
* **Client Storage**: JWT is stored on the client side (usually in localStorage or cookies).
* **Authenticated Requests**: Client sends the JWT in the Authorization header (Bearer <token>) with each request to protected endpoints.

#### 🔒 Benefits:

* Stateless: No need to store session data on the server.
* Scalable: Works well with distributed systems.
* Secure: Can be signed and optionally encrypted.

### 🛡️ Middleware to Protect Private Routes

Middleware acts as a gatekeeper for routes that require authentication.

#### 🧩 Typical Middleware Flow:

js

constauthenticateJWT = (req, res, next) => {

const token = req.headers.authorization?.split(' ')[1];

if (!token) return res.status(401).json({ message: 'Access denied' });

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

req.user = decoded; // Attach user info to request

next(); // Proceed to route handler

} catch (err) {

res.status(403).json({ message: 'Invalid token' });

}

};

#### 🔐 Usage:

js

app.get('/dashboard', authenticateJWT, (req, res) => {

res.json({ message: `Welcome, ${req.user.name}` });

});

# 9. USER INTERFACE

### 🖥️ 1. User Interface (UI)

The **User Interface** is the overall visual and interactive layer of the platform. It includes:

* **Navigation menus** (top bar, sidebar)
* **Buttons, icons, and input fields**
* **Typography and color schemes**
* **Responsive design** for mobile and desktop
* **Accessibility features** (contrast, keyboard navigation, etc.)

A clean, intuitive UI ensures users can easily access features like job listings, messages, payments, and profile settings.

### 🌐 2. Landing Page

The **Landing Page** is the first impression for visitors and potential users. It should:

* **Highlight platform benefits** (e.g., “Hire top freelancers” or “Find remote gigs fast”)
* Include **calls to action** like “Sign Up” or “Post a Job”
* Showcase **testimonials, stats, or featured freelancers**
* Offer **quick navigation** to key areas (e.g., Browse Jobs, How It Works)
* Be optimized for **SEO and conversion**

Think of it as the marketing front door to your freelancer ecosystem.

### 🧑‍💻 3. Freelancer Dashboard

The **Freelancer Dashboard** is the control center for registered freelancers. It typically includes:

* **Job feed** tailored to their skills
* **Proposal management** (sent, accepted, rejected)
* **Earnings overview** and withdrawal options
* **Client messages and notifications**
* **Profile editing** and skill updates
* **Performance metrics** (ratings, completed jobs, etc.)

This dashboard should be efficient, motivating, and easy to navigate—helping freelancers manage their work and grow their reputation.

# 10. TESTING

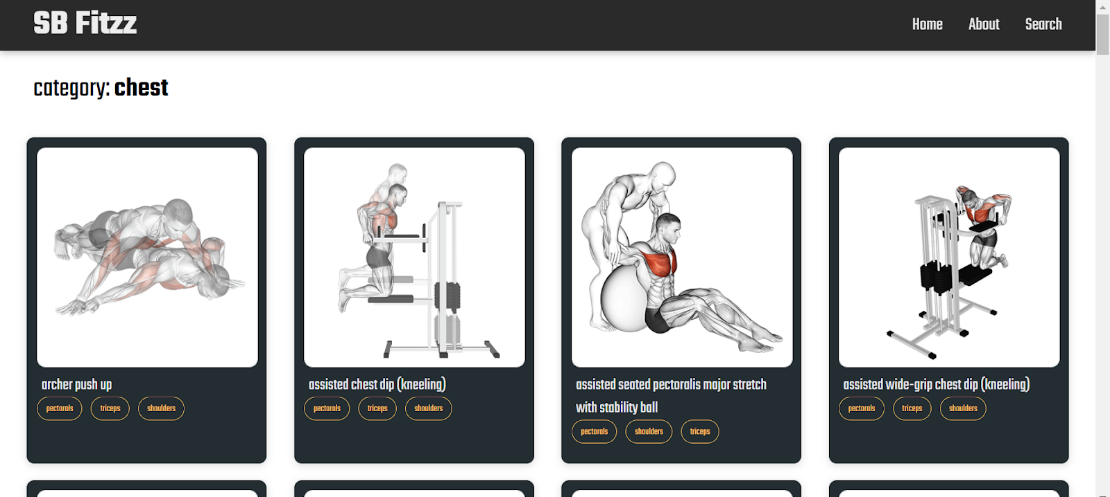
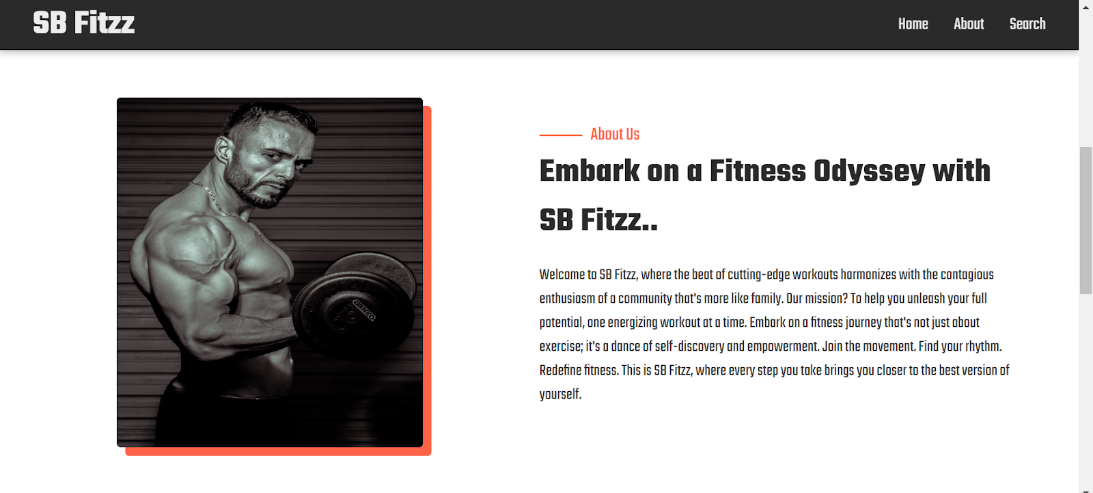
#### ✅ Manual Testing During Milestones

* **Purpose**: Validate core functionality and user experience at each development phase.
* **Approach**:
  + Test critical flows like registration, job posting, proposal submission, and payment processing.
  + Use test cases and checklists aligned with milestone goals.
  + Involve QA testers and stakeholders for feedback.
* **Benefits**:
  + Early detection of bugs
  + Ensures alignment with user expectations
  + Reduces risk of regression

#### 🛠️ Tools Used

| **Tool** | **Purpose** |
| --- | --- |
| **Postman** | API testing: validate endpoints, check response formats, simulate requests |
| **Chrome Dev Tools** | Frontend debugging: inspect elements, monitor network activity, analyze performance |

**11. SCREENSHOTSOR DEMO**



**12. KNOWN ISSUES**

#### 1. **Authentication & Authorization**

* Some versions rely heavily on Azure Active Directory, which may cause login issues if misconfigured.
* Limited support for social logins (e.g., Google, Facebook) in certain forks.

#### 2. **API Integration Challenges**

* FitFlex uses the **Wger API** for exercise data, which can occasionally return inconsistent results or experience downtime.
* Lack of fallback mechanisms when third-party APIs fail.

#### 3. **Frontend Bugs**

* UI components built with React may show rendering glitches, especially in calendar and chart modules.
* Responsive design issues on smaller screens (e.g., mobile layout misalignment).

#### **4.**P**erformance bottlenecks**

* Real-time updates and chart rendering can slow down on lower-end devices or browsers.
* Heavy reliance on client-side rendering without optimization.

#### 5. **Deployment & Configuration**

* Docker containerization is supported but may require manual tweaks for local development.
* Ejecting from Create React App can lead to complex build configurations that are hard to maintain.

#### 6. **Monitoring & Debugging**

* Azure Monitor and Application Insights are integrated but may not capture all frontend errors.
* Limited logging for user actions and API failures.

**13. Future Enhancements**

#### **1. AI-Powered Personalization**

* Integrate machine learning to tailor workout plans based on user goals, progress, and preferences.
* Smart recommendations for nutrition, rest days, and intensity adjustments.

#### **2. Gamification Features**

* Add badges, streaks, and challenges to boost user motivation.
* Leaderboards for community engagement and friendly competition.

#### 3. **Expanded API Support**

* Broaden integration beyond Wger API to include nutrition databases, wearable fitness trackers (e.g., Fitbit, Apple Health), and sleep monitoring tools.

#### 4. **Mobile App Development**

* Native iOS and Android apps for better performance and offline access.
* Push notifications for reminders and progress updates.

#### 5. **Advanced Analytics Dashboard**

* Visualize trends in strength, endurance, and body metrics.
* Predictive insights to prevent overtraining or plateaus.

#### 6. **Community & Social Features**

* In-app forums or chat groups for sharing tips and progress.
* Ability to follow other users and view public workout logs.

#### 7. **Accessibility & Localization**

* Multilingual support and screen reader compatibility.
* Regional units (kg/lbs, km/miles) and culturally relevant content.

#### 8. **Subscription & Monetization Options**

* Freemium model with premium features like expert coaching, exclusive plans, and ad-free experience.
* Integration with payment gateways for seamless upgrades.