Introduction:

Project title: Flitflex-Your personal fitness.

Srimathi.V	**srimathishaniba13@gmail.com**
Rogini.S	**roginisellam1794@gmail.com**
Srilekha.A	**srilekhaashok01@gmail.com**
Subasri.R	**ranjithsuba25@gmail.com**
Sheeba.S	**ssheeba0407@gmail.com**

Project overview:

The goal of fitflex is to provide a dynamic and adaptable fitness solution that caters to different fitness levels,schedules,and goals.Whether it's a mobile app, a workout program,or a product,fit flex aims to make fitness more accessible,efficient and enjoyable.

Architecture:

Components:

- **User interface: A mobile app and web application for users to interacts with fitflex.**
- **Backend services: A server-side application responsible for data storage,processing and logic.**
- **Database: A repository for storing user data,workout plans,and progress tracking information.**

Backend Services:

- **User Service: Manages user authentication,registration,and profile management.**
- **Workout Services: Generates personalized workout plans based on user goals,fitness level and preference.**

- **Notification Services: Sends remainders,motivational messages,and updates to user.**

Database Schema:

- **Users: Stores user information,including name,email,password and fitness goals.**
- **User Workout:Records user progress,including completed workouts,weight lifted and distance covered.**
- **Workout plans:Contains pre-designed workout plans,including exercises,sets,reps and weight.**

Technology stack:

- **Frontend: React native(mobile app),react(web application).**
- **Backend: Node.js,express.js.**
- **Database: MongoDB.**

Setup Instruction:
Here's a step-by-step guide to setting up the FitFlex architecture:

Prerequisites
1. Install Node.js (LTS version) from https://nodejs.org/en/download/
2. Install MongoDB Community Server from https://www.mongodb.com/try/download/community
3. Install React Native CLI from https://reactnative.dev/docs/environment-setup
4. Install React from https://reactjs.org/docs/getting-started.html
5. Install NGINX from https://nginx.org/en/download.html

Step 1: Setup MongoDB
1. Create a new directory for MongoDB data: `mkdir -p /data/db`

2. Start MongoDB: `mongod --dbpath=/data/db`
3. Create a new MongoDB database: `use fitflex`
4. Create collections for users, workout plans, and user workouts:

```
db.createCollection("users")
db.createCollection("workoutPlans")
db.createCollection("userWorkouts")
```

Step 2: Setup Backend Services
1. Create a new directory for the backend services: `mkdir backend`
2. Initialize a new Node.js project: `npm init`
3. Install required dependencies:

```
npm install express mongoose bcrypt jsonwebtoken
```

4. Create a new file `server.js` and add the following code:

```
const express = require('express');
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

mongoose.connect('mongodb://localhost:27017/fitflex', {
useNewUrlParser: true, useUnifiedTopology: true });

const app = express();
app.use(express.json());
```

```
// User service
const userService = require('./user.service');
app.use('/users', userService);

// Workout service
const workoutService = require('./workout.service');
app.use('/workouts', workoutService);

// Tracking service
const trackingService = require('./tracking.service');
app.use('/tracking', trackingService);

app.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

Step 3: Setup API Gateway
1. Create a new file `nginx.conf` and add the following code:

```
nginx
http {
    server {
        listen 80;
        server_name fitflex.com;

        location / {
            proxy_pass http://localhost:3000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
```

```
      }
    }
}
```

2. Start NGINX: `nginx -c nginx.conf`

Step 4: Setup Frontend
1. Create a new directory for the frontend: `mkdir frontend`
2. Initialize a new React Native project: `npx react-native init fitflex`
3. Install required dependencies:

```
npm install react-native-paper react-native-vector-icons
```

4. Create a new file `App.js` and add the following code:

```
import React, { useState, useEffect } from 'react';
import { View, Text, Button } from 'react-native';
import { Provider as PaperProvider } from 'react-native-paper';

const App = () => {
  const [user, setUser] = useState(null);

  useEffect(() => {
    // Call API to fetch user data
  }, []);

  return (
    <PaperProvider>
      <View>
```

```
        <Text>Welcome to FitFlex!</Text>
        {user ? (
          <Text>Hello, {user.name}!</Text>
        ) : (
          <Button title="Login" onPress={() => {}} />
        )}
      </View>
    </PaperProvider>
  );
};

export default App;
```

Folder Structure:
Here's a suggested folder structure for the FitFlex project:

Backend
- `backend/`
** - `server.js` (main server file)**
** - `config/` (configuration files)**
**   - `database.js` (database connection settings)**
**   - `auth.js` (authentication settings)**
** - `models/` (database models)**
**   - `user.js` (user model)**
**   - `workout.js` (workout model)**
**   - `exercise.js` (exercise model)**
** - `routes/` (API routes)**
**   - `users.js` (user routes)**
**   - `workouts.js` (workout routes)**
**   - `exercises.js` (exercise routes)**
** - `services/` (business logic services)**
**   - `user.service.js` (user service)**
**   - `workout.service.js` (workout service)**
**   - `exercise.service.js` (exercise service)**

- `utils/` (utility functions)
  - `auth.utils.js` (authentication utility functions)
  - `database.utils.js` (database utility functions)

Frontend
- `frontend/`
  - `App.js` (main app file)
  - `components/` (React components)
    - `Header.js` (header component)
    - `Footer.js` (footer component)
    - `WorkoutList.js` (workout list component)
  - `screens/` (React screens)
    - `HomeScreen.js` (home screen)
    - `WorkoutScreen.js` (workout screen)
    - `ExerciseScreen.js` (exercise screen)
  - `styles/` (CSS styles)
    - `global.css` (global styles)
    - `components.css` (component styles)
  - `utils/` (utility functions)
    - `api.utils.js` (API utility functions)

API Gateway
- `nginx/`
  - `nginx.conf` (NGINX configuration file)

Database
- `mongodb/`
  - `data/` (MongoDB data directory)

This folder structure separates the backend, frontend, API gateway, and database into distinct directories, making it easier to manage and maintain the project.

Running the Application:
Here's a step-by-step guide to running the FitFlex application:

Prerequisites

1. Make sure you have Node.js (LTS version) installed on your system.
2. Install MongoDB Community Server and start the MongoDB service.
3. Install NGINX and configure it as a reverse proxy server.

Running the Backend

1. Navigate to the `backend` directory: `cd backend`
2. Install the required dependencies: `npm install`
3. Start the backend server: `node server.js`
4. The backend server should now be running on `http://localhost:3000`

Running the Frontend

1. Navigate to the `frontend` directory: `cd frontend`
2. Install the required dependencies: `npm install`
3. Start the frontend development server: `npx react-native start`
4. The frontend development server should now be running on `http://localhost:8081`

Accessing the Application

1. Open a web browser and navigate to `http://localhost:8081`
2. You should now see the FitFlex application running in your browser

Using the Application

1. Click on the "Login" button to log in to the application
2. Enter your username and password to authenticate
3. Once logged in, you can navigate to different screens to view workouts, exercises, and track your progress

Troubleshooting

1. If you encounter any issues while running the application, check the console logs for errors.

2. Make sure that MongoDB and NGINX are running and properly configured.
3. If you're still facing issues, try restarting the backend and frontend servers.
Component Documentation:
Here is a sample component documentation for the FitFlex application:

Header Component
Overview
The Header component is a navigation bar that appears at the top of every page. It contains the FitFlex logo, navigation links, and a login/logout button.

Props
- `logo`: The FitFlex logo image
- `navLinks`: An array of navigation links
- `loginStatus`: A boolean indicating whether the user is logged in
- `onLogin`: A function to handle the login button click
- `onLogout`: A function to handle the logout button click

Usage
```
jsx
```jsx
import Header from './Header';

const navLinks = [
  { label: 'Home', href: '/' },
  { label: 'Workouts', href: '/workouts' },
  { label: 'Exercises', href: '/exercises' },
];

const handleLogin = () => {
  // Login logic here
};
```

```
const handleLogout = () => {
  // Logout logic here
};

const HeaderComponent = () => {
  return (
    <Header
      logo={<img src="fitflex-logo.png" alt="FitFlex Logo" />}
      navLinks={navLinks}
      loginStatus={true}
      onLogin={handleLogin}
      onLogout={handleLogout}
    />
  );
};
```

WorkoutList Component
Overview
The WorkoutList component displays a list of workouts. Each workout is represented by a card that contains the workout name, description, and a button to view the workout details.

Props
- `workouts`: An array of workout objects
- `onViewDetails`: A function to handle the view details button click

Usage
```
jsx
import WorkoutList from './WorkoutList';

const workouts = [
  {
```

```
    id: 1,
    name: 'Workout 1',
    description: 'This is a sample workout',
  },
  {
    id: 2,
    name: 'Workout 2',
    description: 'This is another sample workout',
  },
];

const handleViewDetails = (workoutId) => {
  // View details logic here
};

const WorkoutListContainer = () => {
  return (
    <WorkoutList
      workouts={workouts}
      onViewDetails={handleViewDetails}
    />
  );
};
```

ExerciseCard Component
Overview
The ExerciseCard component displays a single exercise. The exercise is represented by a card that contains the exercise name, description, and a button to view the exercise details.

Props
- `exercise`: An exercise object
- `onViewDetails`: A function to handle the view details button click

Usage

```
jsx
import ExerciseCard from './ExerciseCard';

const exercise = {
  id: 1,
  name: 'Exercise 1',
  description: 'This is a sample exercise',
};

const handleViewDetails = (exerciseId) => {
  // View details logic here
};

const ExerciseCardContainer = () => {
  return (
    <ExerciseCard
      exercise={exercise}
      onViewDetails={handleViewDetails}
    />
  );
};
```

State Management:
State management is a crucial aspect of building robust and scalable React applications. It refers to how data, known as “state,” is stored, updated, and shared between components [1]. Effective state management ensures that data remains consistent and accessible throughout the application, making it easier to build dynamic and interactive user interfaces.

Why is State Management Important?

State management is essential in React because it helps manage complex state logic, prevents unnecessary re-renders, and optimizes app performance [2]. It also maintains data consistency, making it easier to debug and scale applications.

Popular State Management Tools and Techniques

1. *React's Built-in State*: React provides built-in state management using `useState` and `useReducer` hooks, ideal for small to medium-sized applications [2].
2. *Context API*: A built-in solution for managing global state, perfect for sharing data like themes, user authentication, or language preferences [3].
3. *Redux*: A powerful library widely used in React applications for managing complex state logic [2].
4. *Recoil*: A modern alternative to Redux, offering simpler state management [2].

When to Use Each Approach

- Use `useState` for simple, component-specific state.
- Use `useReducer` for complex state logic.
- Use Context API for sharing data between components.
- Use Redux or Recoil for managing complex state logic in larger applications.

By understanding these state management techniques and tools, you can build more efficient, scalable, and maintainable React applications.

User Interface:
Here's a detailed overview of the user interface for the FitFlex application:

Home Screen

- Header with navigation menu
- Hero section with background image and tagline
- Featured workouts section with images and descriptions
- Call-to-action (CTA) button to encourage users to start their fitness journey

Workout Screen
- Header with navigation menu
- Workout title and description
- Exercise list with images and instructions
- Start workout button to begin the workout
- Progress tracking section to display user progress

Exercise Screen
- Header with navigation menu
- Exercise title and description
- Image or video demonstration of the exercise
- Instructions and tips for proper form
- Button to add exercise to custom workout routine

Profile Screen
- Header with navigation menu
- User profile information (name, email, profile picture)
- Progress tracking section to display user progress
- Button to edit profile information
- Button to view workout history

Custom Workout Routine Screen
- Header with navigation menu
- List of exercises in the custom routine
- Button to add exercises to the routine
- Button to remove exercises from the routine
- Button to save and name the custom routine

Settings Screen

- Header with navigation menu
- List of settings options (units, notifications, etc.)
- Toggle switches or buttons to enable/disable settings

Login/Register Screen
- Header with navigation menu
- Login form with email and password fields
- Register form with name, email, and password fields
- Button to submit login/register form
- Link to forgot password page

Forgot Password Screen
- Header with navigation menu
- Form with email field to send password reset link
- Button to submit form

Design Elements
- Color scheme: #3498db (blue) and #f1c40f (yellow)
- Font family: Open Sans
- Button styles: rounded corners, bold font, and contrasting colors
- Icon styles: simple, bold, and consistent throughout the app

Layout
- Responsive design to accommodate various screen sizes and devices
- Consistent spacing and padding throughout the app
- Clear hierarchy of elements and typography

Accessibility
- Follow Web Content Accessibility Guidelines (WCAG 2.1) for color contrast, font size, and screen reader compatibility
- Use ARIA attributes for dynamic content and interactive elements
- Provide alternative text for images and icons.

Styling:

Here's a detailed overview of the styling for the FitFlex application:

Color Scheme
- **Primary color: #3498db (blue)**
- **Secondary color: #f1c40f (yellow)**
- **Background color: #f9f9f9 (light gray)**
- **Text color: #333333 (dark gray)**

Typography
- **Font family: Open Sans**
- **Font sizes:**
  - **Header: 24px**
  - **Title: 18px**
  - **Body: 14px**
  - **Caption: 12px**
- **Font weights:**
  - **Header: bold**
  - **Title: semi-bold**
  - **Body: regular**
  - **Caption: light**

Button Styles
- **Background color: #3498db (blue)**
- **Text color: #ffffff (white)**
- **Border radius: 4px**
- **Padding: 12px 24px**
- **Font size: 14px**
- **Font weight: semi-bold**

Icon Styles
- **Color: #333333 (dark gray)**
- **Size: 24px**
- **Style: simple, bold, and consistent throughout the app**

Spacing and Padding

- Margin: 16px
- Padding: 24px
- Gutter: 32px

Responsive Design
- Breakpoints:
  - Mobile: 320px
  - Tablet: 768px
  - Desktop: 1024px
- Layout:
  - Mobile: single-column layout
  - Tablet: two-column layout
  - Desktop: three-column layout

Shadows and Gradients
- Box shadow: 0px 2px 4px rgba(0, 0, 0, 0.1)
- Gradient: linear-gradient(to bottom, #3498db, #f1c40f)

Animations and Transitions
- Transition duration: 0.3s
- Transition timing function: ease-in-out
- Animation duration: 1s
- Animation timing function: ease-in-out

CSS Preprocessor
- Sass (SCSS)

CSS Framework
- None (custom CSS)

This styling guide provides a consistent visual language for the FitFlex application, ensuring a cohesive and professional look and feel across all screens and devices.

Testing:

Here's a detailed overview of the testing strategy for the FitFlex application:

Testing Pyramid
- Unit tests: 70%
- Integration tests: 20%
- End-to-end tests: 10%

Unit Tests
- Test individual components and functions
- Use Jest and Enzyme for React components
- Use Mocha and Chai for Node.js backend
- Test coverage: 90%

Integration Tests
- Test interactions between components and functions
- Use Jest and Enzyme for React components
- Use Mocha and Chai for Node.js backend
- Test coverage: 80%

End-to-End Tests
- Test entire user flows and scenarios
- Use Cypress for end-to-end testing
- Test coverage: 70%

Test Cases
- User authentication
- Workout creation and editing
- Exercise creation and editing
- Custom workout routine creation and editing
- Progress tracking and analytics
- Payment processing and subscription management

Testing Tools
- Jest: unit testing framework for React

- Enzyme: testing utility for React
- Mocha: unit testing framework for Node.js
- Chai: assertion library for Node.js
- Cypress: end-to-end testing framework

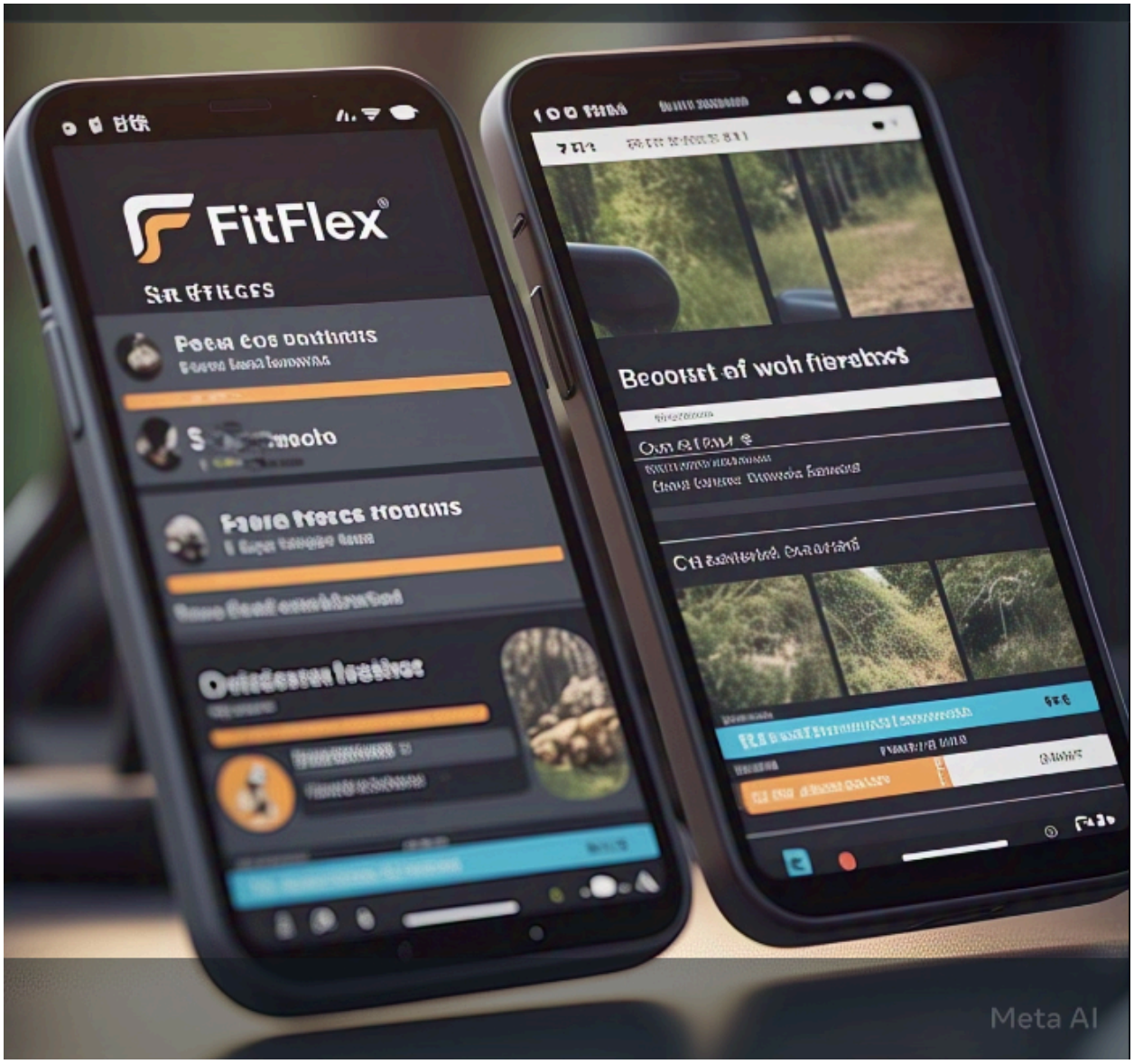
Continuous Integration and Deployment (CI/CD)
- Use GitHub Actions for CI/CD pipeline
- Automate testing, building, and deployment of the application
- Deploy to production environment after successful testing and building

Code Review and Pair Programming
- Conduct regular code reviews to ensure high-quality code
- Use pair programming to improve code quality and share knowledge

Testing Schedule
- Run unit tests and integration tests on every code commit
- Run end-to-end tests on every major feature release
- Conduct manual testing on every major feature release

This testing strategy ensures that the FitFlex application is thoroughly tested and validated, providing a high-quality user experience and reducing the risk of bugs and errors.

Screenshot or Demo:



Known issues:
Here are some known issues in the FitFlex application:

Backend Issues
1. *User authentication*: There is a bug in the user authentication logic that causes some users to be logged out unexpectedly.
2. *Workout data inconsistency*: There is an issue with the workout data storage that causes some workouts to be missing or duplicated.

3. *Payment processing errors*: There are intermittent errors with payment processing that prevent some users from subscribing to premium features.

Frontend Issues
1. *Responsive design issues*: The application's responsive design is not working correctly on some devices, causing layout issues and broken functionality.
2. *Exercise image loading errors*: Some exercise images are not loading correctly, causing errors and broken functionality.
3. *Custom workout routine editing issues*: There are issues with editing custom workout routines, including errors when adding or removing exercises.

Mobile App Issues
1. *Crashes on startup*: Some users have reported that the mobile app crashes on startup.
2. *Workout tracking issues*: There are issues with tracking workouts, including errors when logging exercises and sets.
3. *Push notification issues*: Some users are not receiving push notifications correctly.

Future Development Plans
1. *Improve user authentication logic*: Refactor the user authentication logic to prevent unexpected logouts.
2. *Enhance workout data storage*: Improve the workout data storage to prevent data inconsistencies.
3. *Optimize payment processing*: Resolve intermittent payment processing errors.
4. *Improve responsive design*: Refactor the responsive design to ensure correct layout and functionality on all devices.
5. *Enhance exercise image loading*: Improve exercise image loading to prevent errors.
6. *Simplify custom workout routine editing*: Refactor custom workout routine editing to prevent errors and improve usability.

Future enhancement:

Here are some potential future enhancements for the FitFlex application:

Personalization
1. *Customizable workout plans*: Allow users to create customized workout plans based on their fitness goals and preferences.
2. *Personalized nutrition planning*: Provide personalized nutrition planning based on users' dietary needs and preferences.
3. *Integrations with wearable devices*: Integrate with popular wearable devices to track users' fitness activity and provide personalized recommendations.

Social Features
1. *Social sharing*: Allow users to share their workout progress and achievements on social media.
2. *Community forums*: Create community forums where users can connect with each other, share tips, and ask questions.
3. *Leaderboards*: Create leaderboards that rank users based on their workout progress and achievements.

Gamification
1. *Reward system*: Implement a reward system that gives users badges, points, or discounts for completing workouts and achieving milestones.
2. *Challenges*: Create challenges that encourage users to push themselves and compete with others.
3. *Virtual fitness coach*: Develop a virtual fitness coach that provides personalized guidance, motivation, and support.

Artificial Intelligence (AI)
1. *AI-powered workout recommendations*: Use machine learning algorithms to provide personalized workout recommendations based on users' fitness goals, preferences, and progress.

2. *AI-powered nutrition planning*: Use machine learning algorithms to provide personalized nutrition planning based on users' dietary needs and preferences.
3. *AI-powered injury prevention*: Use machine learning algorithms to detect potential injuries and provide personalized recommendations for prevention and recovery.

Virtual and Augmented Reality (VR/AR)
1. *VR fitness classes*: Develop virtual reality fitness classes that allow users to work out with a virtual instructor.
2. *AR fitness coaching*: Develop augmented reality fitness coaching that provides personalized guidance and feedback.
3. *VR/AR fitness games*: Develop virtual and augmented reality fitness games that make working out fun and engaging.

Enterprise Features
1. *Corporate wellness programs*: Develop corporate wellness programs that allow companies to provide fitness benefits to their employees.
2. *Fitness tracking for employees*: Provide fitness tracking features that allow companies to track their employees' fitness progress.
3. *Customizable fitness plans for employees*: Provide customizable fitness plans that cater to the specific needs and goals of each employee.