# FIT-FLEX

## 1. Introduction

**Project Title:** fitflex

**Team Members:**

* **Team leader-S. Geetha**
* **Team Members- T. Divya**

**S. Ajitha**

**S. Arakashsamitha**

**R. Bhavanieaswari**

## 2. Project Overview

### ****Purpose:****

The fitness app is designed to help users achieve their health and fitness goals through personalized workout plans, progress tracking, and interactive features. It integrates AI-driven recommendations, nutrition guidance, and social engagement to enhance motivation and consistency. The app is suitable for beginners, fitness enthusiasts, and athletes, providing a seamless experience through mobile and wearable device integration.

### ****Features:****

* **User Profiles & Goals – Personalized fitness goals based on user input.**
* **Workout Plans – Customizable exercise routines for different fitness levels.**
* **Activity Tracking – Logs workouts, steps, and calories burned.**
* **Nutrition Guide – Meal planning and calorie tracking.**
* **AI-Powered Coaching – Personalized recommendations based on progress.**
* **Integration with Wearables – Syncs with fitness trackers and smartwatches.**
* **Community & Challenges – Social features, leaderboards, and challenges.**
* **Progress Reports – Visual charts and analytics on fitness progress.**
* **Live & On-Demand Workouts – Access to virtual training sessions.**
* **Reminders & Notifications – Motivational alerts for workouts and diet.**

## 3. Architecture

**Component Structure:**

* Presentation Layer (Frontend) – User interface and interactions
* Business Logic Layer (Application Layer) – Core app functions and logic
* Data Layer (Backend & Database) – Secure data storage and retrieval
* API Layer (Communication) – Data exchange between frontend and backend
* Third-Party Integrations – Enhancing app functionality
* Player.js: Controls music playback.
* DevOps & Security Layer – Deployment, monitoring, and security

**Routing:**

* **Frontend Navigation:**
* Stack Navigation: Login → Home → Workout → Summary
* Tab Navigation: Home | Workouts | Nutrition | Profile
* Drawer Navigation: Settings, Notifications, Support
* Deep Linking: Direct access via URLs or notifications
* **Backend API Routes**:
* Auth: /login, /register, /logout
* Workouts: /workouts, /workouts/{id}
* Nutrition: /meals, /nutrition-plan

## User: /profile, /settings, /progress4.

## Setup Instructions

**Prerequisites:**

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/

React.js:

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.

Install React.js, a JavaScript library for building user interfaces.

**Installation:**

* **Clone the Project**
* Use Git**:**
* Git clone <repository\_url>
* Cd project-folder
* **Install Dependencies**
* For Node.js/React Native:
* Npm install

**Development Environment:**

Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• Visual Studio Code: Download from https://code.visualstudio.com/download

• Sublime Text: Download from https://www.sublimetext.com/download

•WebStorm:Download from https://www.jetbrains.com/webstorm/download

## 5. Folder Structure

Client

* **Frontend (React Native/Flutter)**
* Fitness-app/
* │-- src/
* │ ├── components/ (Reusable UI elements)
* │ ├── screens/ (Workout, Diet, Profile)
* │ ├── navigation/ (App navigation)
* │ ├── services/ (API calls, authentication)
* │ ├── assets/ (Images, icons, fonts)
* │ ├── utils/ (Helper functions)
* │ ├── styles/ (Global styles)
* │-- App.js / main.dart (Entry point)
* **Backend (Node.js/Express)**
* Fitness-api/
* │-- src/
* │ ├── models/ (Database schemas)
* │ ├── routes/ (API endpoints)
* │ ├── controllers/ (Business logic)
* │ ├── middleware/ (Auth, validation)
* │ ├── config/ (DB, environment)
* │-- server.js (Entry point)

### ****Utilities (****src/utils/****)****

* Authentication – User login, signup, and password management.
* **Workout Tracking** – Log exercises, reps, sets, and progress.
* **Diet & Nutrition** – Meal tracking, calorie counter, and diet plans
* **Wearable Integration** – Sync with smartwatches (Google Fit, Apple Health).
* **Push Notifications** – Workout reminders and goal tracking.
* **AI & Analytics** – Personalized workout suggestions and progress insights.
* **Social Features** – Community challenges, leaderboards, and sharing.
* **Offline Mode** – Access workouts without the internet.
* **Payment & Subscriptions** – Premium plans for extra features.

## 6. Running the Application

* Open music-streaming-app-main folder
* Start the frontend server: npm run dev

## 7. Component Documentation

**Key Components:**

* **Navigation**
* Bottom Navigation Bar – Home, Workouts, Nutrition, Progress, Profile.
* Side Menu – Settings, Community, Subscription.
* Back & Forward Buttons – For smooth navigation.
* **Home screen**
* User Greeting Widget – Personalized welcome message.
* Activity Summary Card – Steps, calories, heart rate.
* Workout Recommendations – Suggested workouts based on goals
* Quick Start Button – Instantly begin a workout.
* **Workout**
* Workout Category Cards – Strength, Cardio, Yoga, etc.
* Workout Detail Page – Video demo, duration, intensity.
* Live Workout Timer – Tracks time, reps, progress.

## 8. State Management

* Frontend State Management (Client-Side)
* Technologies: Redux (React Native), Provider & Riverpod (Flutter), MobX, Zustand
* Manages: UI state, user sessions, workout progress, form inputs
* Use Case: Keeps track of workout status, current exercises, and nutrition logs
* Backend State Management (Server-Side)
* Technologies: Node.js with Redis, Firebase Realtime Database, AWS DynamoDB
* Manages: User authentication, activity tracking, leaderboards, notifications
* Use Case: Syncs user progress across devices, stores workout history
* Global State Management (Hybrid Approach)
* Technologies: GraphQL with Apollo, React Query, SWR
* Manages: Cached API responses, background sync, real-time update
* Use Case: Live workout updates, leaderboard changes, AI recommendations

#### ****Caching & Persistent Storage:****

* **Caching (Fast Data Access)**
* AsyncStorage (React Native) / SharedPreferences (Android) / UserDefaults (iOS) – Store small data like user preferences.
* React Query / Apollo Cache – Cache API responses for faster load times.
* Redis – Backend caching for frequently accessed data.
* **Persistent Storage (Long-Term Data)**
* SQLite / Room DB (Android) / Core Data (iOS) – Store workouts, meals, and progress.
* RealmDB / Firebase Firestore – Sync data across devices.
* SecureStorage (for tokens & sensitive info).

## 9. User Interface

* **Home Screen (Dashboard) – Personalized greeting, daily stats, quick workout access, progress tracker.**
* **Workout Selection** – Categories, filters, workout previews, start button.
* **Live Workout Screen** – Timer, video demo, instructions, music controls
* **Nutrition & Meal Plan** – Calorie tracker, meal plans, food log, water intake.
* **Progress & Analytics** – Weekly/monthly reports, weight tracking, achievements.
* **Community & Challenges** – Leaderboards, workout buddies, forums.
* **Settings & Profile** – Goals, notifications, device integration, subscription.

## 10. Styling

Fit flex utilizes a combination of **Bootstrap, Tailwind CSS, and Styled-Components** to achieve a modern, responsive, and visually appealing user interface.

### ****CSS Frameworks/Libraries:****

* **Bootstrap**: Provides a responsive grid system and pre-designed components for faster UI development.
* **Tailwind CSS**: Used for utility-based styling, enabling a highly customizable and flexible design.
* **Styled-Components**: Allows dynamic styling through JavaScript, making it easier to implement theme-based styles.

### ****Theming:****

* Supports **Dark Mode and Light Mode** with smooth transitions.
* CSS variables and Styled-Components ensure easy theme customization.
* User preferences for themes are saved in local storage for a personalized experience.

## 11. Testing

Fit flex follows a structured testing approach to ensure the reliability and stability of the application. The testing process includes **unit, integration, and end-to-end (E2E) testing** to verify the correctness of components and user interactions.

### ****Testing Strategy:****

* **Unit Testing**: Conducted using **Jest** to test individual components and utility functions.
* **Integration Testing**: Utilizes **React Testing Library** to verify component interactions and data flow.
* **End-to-End (E2E) Testing**: Uses **Cypress** to simulate real user journeys, ensuring smooth navigation and functionality.

### ****1. Unit Testing (Jest)****

Unit tests focus on individual functions or components. Here’s an example test for the **Player** component.

#### Player.test.js ****(Unit Test)****

Import { render } from ‘@testing-library/react-native’;

Import WorkoutScreen from ‘../screens/WorkoutScreen’;

Test(‘renders workout screen with start button’, () => {

Const { getByText } = render(<WorkoutScreen />);

Expect(getByText(‘Start Workout’)).toBeTruthy();

});

const request = require('supertest');

const app = require('../server');

const { expect } = require('chai');

describe('GET /api/workouts', () => {

it('should return status 200 and workouts', async () => {

const res = await request(app).get('/api/workouts');

expect(res.status).to.equal(200);

expect(res.body).to.be.an('array');

});

});

**2. Integration Testing (React Testing Library)**

Integration tests verify if different components work together as expected.

#### Playlist.test.js ****(Integration Test)****

Import { render, fireEvent, waitFor } from ‘@testing-library/react-native’;

Import WorkoutScreen from ‘../screens/WorkoutScreen’;

Import { fetchWorkouts } from ‘../services/workoutService’;

// Mock API call

Jest.mock(‘../services/workoutService’, () => ({

fetchWorkouts: jest.fn(() => Promise.resolve([{ id: 1, name: ‘Push-Ups’ }])),

}));

Test(‘loads and starts a workout’, async () => {

Const { getByText, queryByText } = render(<WorkoutScreen />);

Expect(queryByText(‘Loading…’)).toBeTruthy();

// Wait for API data to load

Await waitFor(() => expect(fetchWorkouts).toHaveBeenCalled());

Expect(getByTex

### ****3. End-to-End (E2E) Testing (Cypress)****

E2E tests verify the full user journey, ensuring all components (UI, API, and database) work together.

#### describe('Workout Flow Test', () => {

#### it('User starts a workout session', () => {

#### cy.visit('/'); // Open app

#### cy.get('[data-testid="login-button"]').click(); // Login

#### cy.get('[data-testid="workout-list"]').should('exist'); // Check workouts

#### cy.get('[data-testid="start-workout"]').click(); // Start a workout

#### cy.contains('Workout in Progress').should('be.visible'); // Verify workout started

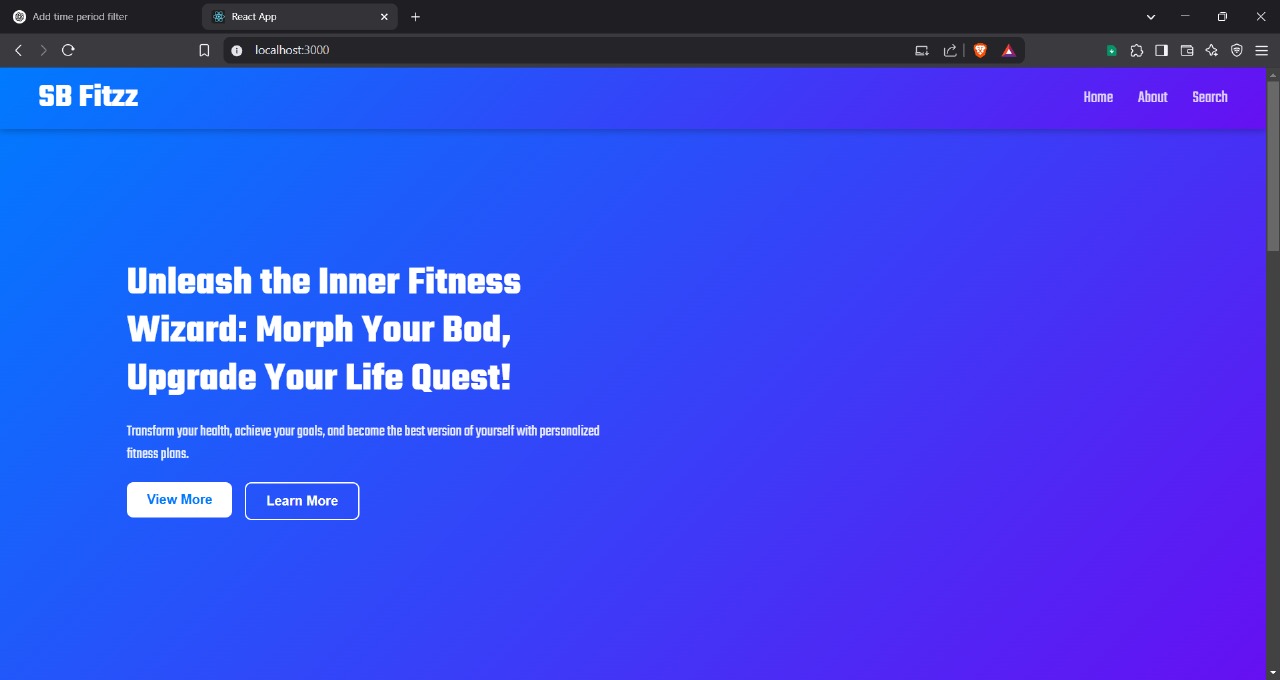
#### });

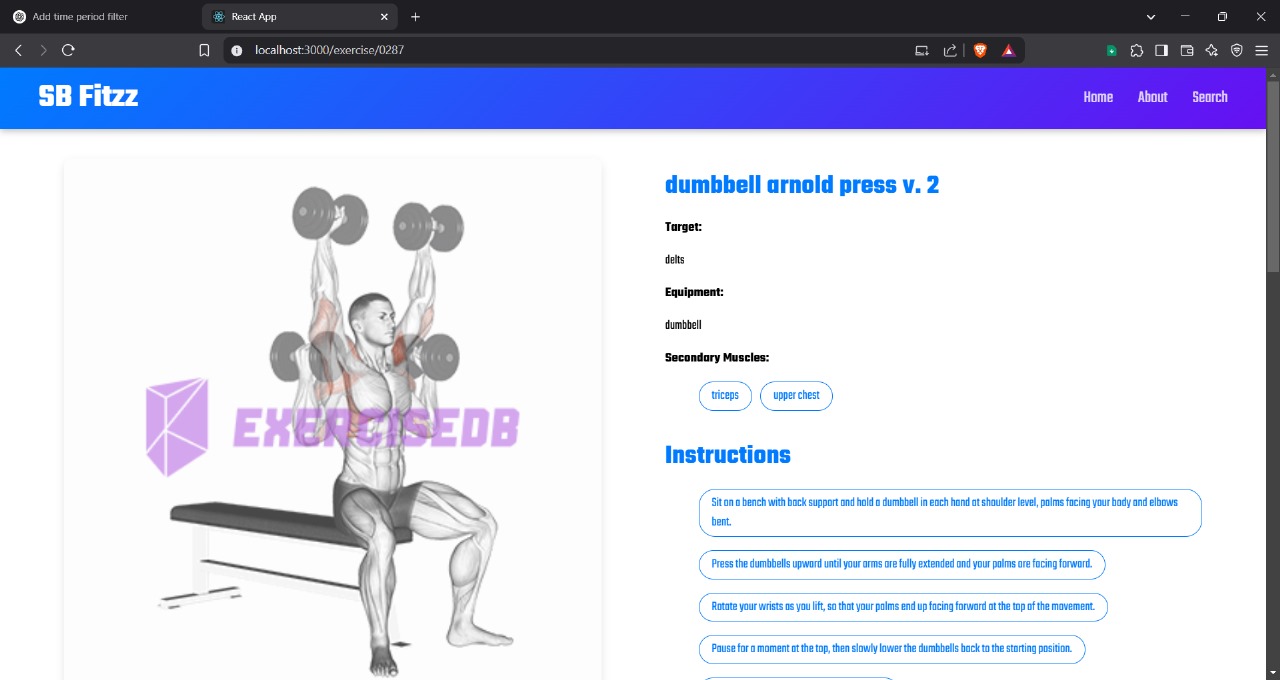
### });

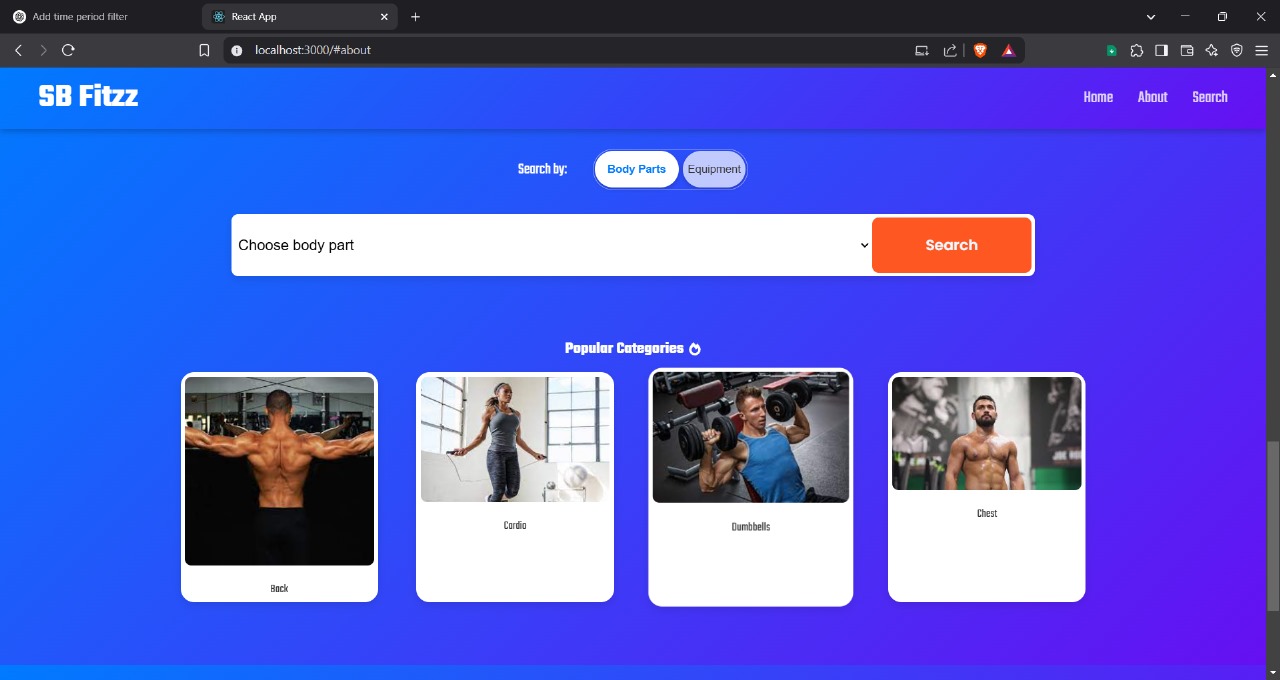
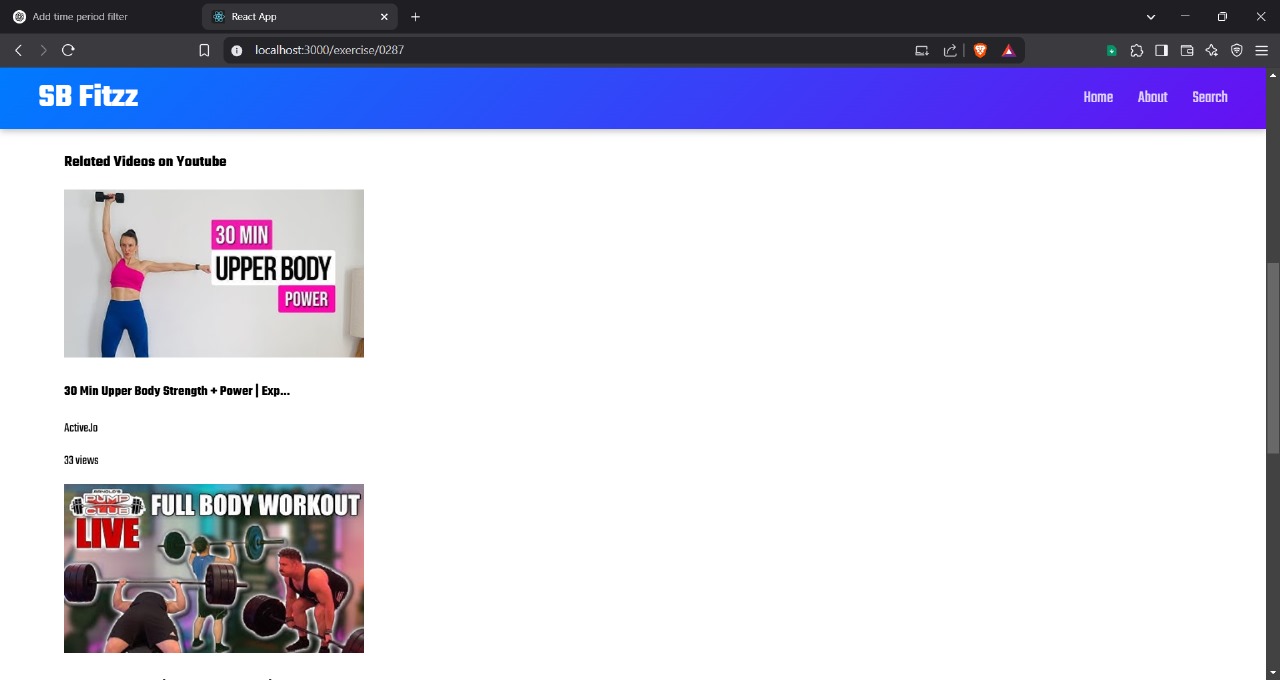
### ****Code Coverage:****

* Jest's built-in coverage tool is used to track tested components and functions.
* Automated tests are integrated into the **CI/CD pipeline** to catch issues early and ensure stability before deployment.
* End-to-end testing using Cypress.

## Demo or Screenshots







## 13. Known Issues

* **Buggy Tracking** – Inaccurate step counting, calorie tracking, or workout logging.
* **Syncing Problems** – Issues connecting with wearables, smartwatches, or other fitness devices.
* **Crashes & Freezes** – App crashing during workouts or failing to load.
* **Limited Customization** – Lack of personalized workout plans or dietary options.
* **Inconsistent Notifications** – Missed reminders for workouts or hydration.
* **Poor User Interface** – Complicated navigation or unclear workout instructions.
* **Subscription & Payment Issues** – Unexpected charges or difficulty canceling.

## 14. Future Enhancements

* **AI-Powered Coaching** – Personalized workout plans based on user progress.
* **AR/VR Workouts** – Immersive virtual training experiences.
* **Gamification** – Challenges, leaderboards, and rewards for motivation.
* **Advanced Wearable Integration** – Seamless syncing with smart devices.
* **Nutritional AI Assistant** – Meal suggestions based on fitness goals.
* **Mental Wellness Features** – Meditation, stress tracking, and sleep analysis.
* **Social & Community Engagement** – Group workouts and live coaching.
* **Voice & Gesture Control** – Hands-free interaction for convenience.
* **Enhanced Data Security** – Improved encryption for personal health data.
* **Offline Mode** – Access workouts without internet connectivity.