1.1: Title of the Project

Fitness Tracker System

- A web application to track physical activities, health metrics, and fitness progress, with backend support using MySQL.

1.2: Objective of the Project

- **Main Objective**: To build a web-based fitness tracker that helps users log and monitor daily activities, health metrics, and long-term progress.
- **Sub-Objectives**:
    - To provide an easy-to-use web interface for tracking fitness data.
    - To use MySQL for secure and efficient data storage and retrieval.
    - To ensure user data privacy, security, and scalability.

1.3: Project Category

- **Type**:
    - **Health & Fitness**: Digital solution aimed at promoting healthy lifestyles.
    - **Web Application Development**: Implemented using web technologies for accessibility across devices.
- **Other Areas**:
    - Database Management
    - Frontend and Backend Development.

1.4: Programming Languages and Technologies Used

- **Frontend**:
    - **HTML, CSS, JavaScript**: To design and develop the user interface.
    - **Frameworks**: React, Vue.js, or Angular (optional) for a more dynamic UI.
- **Backend**:
    - **PHP/Node.js/Python (Django)**: For server-side logic and handling client requests.

    - **MySQL**: Database for managing and storing user data securely.
- **Other Technologies**:
    - **APIs** for connecting to external fitness data sources (e.g., Google Fit, Apple Health).

1.5: Hardware and Software Requirements

Hardware Requirements:

- **Basic Web Server**: Hosting environment for deploying the application (can use cloud services like AWS or local servers).

Software Requirements:

- **Web Development Environment**:
    - Code editor (e.g., VSCode)
    - Version control (e.g., Git)
- **Database**:
    - MySQL for managing user data, fitness logs, and historical data.
- **Web Server**:
    - Apache or Nginx for handling requests and serving the web application.

1.6: Structure of the Program

- **Frontend**:
    - **UI/UX Design**: User-friendly interface to log and view health metrics.
    - **Data Visualization**: Charts and graphs to represent data visually.
- **Backend**:
    - **API Layer**: Connects frontend to MySQL for data storage and retrieval.
    - **Database Management**: MySQL to store user profiles, fitness logs, and historical data.

- **Data Flow**:
    - Users input data through the frontend.
    - Backend processes requests, interacts with MySQL, and sends data back to the frontend for display.

1.7: Limitations

- **Data Accuracy**: Limited to self-reported data unless integrated with tracking devices.
- **Scalability**: Larger user bases may require database optimization.
- **Privacy**: User data handling requires strict privacy controls and encryption.
- **Device Dependency**: As a web-based platform, relies on users having access to internet and compatible devices.

1.8: Future Scope

- **Mobile-Friendly Design**:
    - Responsive web design for seamless use on mobile devices.
- **Integration with Wearables**:
    - Ability to sync data from fitness wearables (e.g., Fitbit, smartwatches).
- **Machine Learning**:
    - Analysis of user data to provide personalized health recommendations.
- **Data Insights and Trends**:
    - Advanced data analytics for predictive insights on user fitness trends.
- **Scalability Enhancements**:
    - Optimization of MySQL or migration to a more scalable database if required for high user volumes.

1.9: Conclusion

- **Summary**:
    - The fitness tracker is a web-based application leveraging MySQL for secure data management and user-friendly design to track fitness data.
- **Key Benefits**:
    - Real-time access to data, personalized insights, and easy-to-navigate interface accessible on any device with internet connectivity.

CHAPTER 2

2. Introduction

2.1 Purpose

This document provides a detailed specification of the software requirements for the Fitness Tracker Web Application. The app is designed to help users track and monitor their fitness and health data over time through a web interface, with backend support using MySQL for data management.

2.2 Scope

The Fitness Tracker Web Application will allow users to log and view their health metrics such as steps, calories burned, heart rate, and other fitness parameters. It will offer a simple and user-friendly interface for data visualization and goal setting. The backend system will use MySQL for data storage and retrieval.

2.3 Definitions, Acronyms, and Abbreviations

- **UI**: User Interface
- **MySQL**: Relational Database Management System
- **API**: Application Programming Interface
- **CRUD**: Create, Read, Update, Delete operations

- **SRS**: Software Requirements Specification

2.4 References

- Web Development Standards
- MySQL Database Documentation
- User Interface (UI) and User Experience (UX) Design Principles

---

2.1.1 Overall Description

2.1.2 Product Perspective

The Fitness Tracker Web Application is an independent system designed to operate on web browsers and be accessible on various devices, including desktops, tablets, and mobile phones. The web application interacts with a MySQL database to manage user data and will be hosted on a web server.

2.1.3 Product Features

1. **User Registration and Login**: Users can create accounts and log in securely.
2. **Profile Management**: Users can update and manage their profile data.
3. **Data Logging**: Users can manually log or import fitness data (steps, calories, etc.).
4. **Data Visualization**: Users can view their fitness data using charts and graphs.
5. **Goal Setting**: Users can set and track fitness goals.
6. **Progress Reports**: Weekly/monthly reports for tracking improvements.
7. **Admin Panel** (Optional): For managing users and reviewing data integrity.

2.1.4 User Classes and Characteristics

- **End User**: Regular users who will input and track their fitness data.
- **Admin** (Optional): Manages user accounts and monitors system usage.

2.1.5 Operating Environment

- **Frontend**: Accessible on any modern web browser (e.g., Chrome, Firefox, Safari).
- **Backend**: Runs on a web server with PHP/Node.js or Python (Django/Flask).
- **Database**: MySQL server for data storage.
- **Hosting**: Cloud-based hosting (e.g., AWS, DigitalOcean).

2.1.6 Constraints

- **Data Privacy**: Must comply with data privacy regulations (e.g., GDPR) for storing sensitive user information.
- **Scalability**: The database and server must handle scaling as user volume grows.
- **Responsiveness**: The app should be responsive across all screen sizes.

2.2 Specific Requirements

2.2.1 Functional Requirements

2.2.2 User Registration and Authentication

- **Description**: Users must register and log in to access the application.
- **Requirements**:
    - The system shall allow new users to register with a username, email, and password.
    - The system shall validate the user's email format and password strength.
    - The system shall authenticate users and allow secure logins.
    - The system shall provide password recovery options.

2.2.3 Profile Management

- **Description**: Users can manage and update personal information.
- **Requirements**:

- The system shall allow users to view and update their profile information, including age, weight, height, and fitness goals.

2.2.4 Fitness Data Logging

- **Description**: Users can log various fitness metrics.
- **Requirements**:
  - The system shall allow users to manually input metrics (e.g., steps, calories burned, distance).
  - The system shall save data to the MySQL database.
  - The system shall provide CRUD functionalities for fitness entries.

2.2.5 Data Visualization

- **Description**: Users can view historical fitness data in a visual format.
- **Requirements**:
  - The system shall generate graphs for metrics over time (e.g., weekly steps, monthly calories).
  - The system shall display charts using libraries such as Chart.js or D3.js.

2.2.6 Goal Setting

- **Description**: Users can set and track fitness goals.
- **Requirements**:
  - The system shall allow users to set personal fitness goals (e.g., daily step goal).
  - The system shall display progress toward goals on the user dashboard.

2.2.7 Progress Reports

- **Description**: Users can access reports to review their fitness trends.
- **Requirements**:
  - The system shall generate weekly and monthly reports.

    - The system shall display key statistics (e.g., average calories burned, total steps).

2.2.8 Non-Functional Requirements

2.2.9 Performance Requirements

- The system should support at least 500 concurrent users without performance degradation.
- The database queries should be optimized to handle data retrieval efficiently.

2.3 Security Requirements

- User data should be stored securely in MySQL using encryption where possible.
- The system should have authentication and authorization protocols.
- Passwords should be hashed and encrypted in the database.

2.3.1 Usability Requirements

- The application should be responsive on mobile and desktop screens.
- The user interface should be simple and intuitive, following UX best practices.

2.3.2 Reliability Requirements

- The system should have a minimum uptime of 99.5%.
- Data consistency should be maintained across sessions.

2.4. External Interface Requirements

2.4.1 User Interfaces

- **Login/Registration Page**: For user authentication.
- **Dashboard**: Shows user fitness data and quick stats.
- **Reports Page**: Displays weekly and monthly progress reports.
- **Settings**: Allows users to manage their profile and goal settings.

2.4.2 Software Interfaces

- **Database**: MySQL database to store user data and fitness logs.
- **Backend Server**: PHP, Node.js, or Python to interact with MySQL and serve frontend requests.

2.4.3 Communications Interfaces

- **Web Protocols**: HTTPS for secure data transmission.
- **APIs**: RESTful APIs for data exchange between frontend and backend.

2.5. Other Requirements

2.5.1 Data Backup and Recovery

- The system should have a data backup strategy to prevent data loss.

2.5.2 Future Enhancements

- Integration with wearable fitness devices (e.g., Fitbit).
- Machine learning recommendations based on user data.
- Social sharing of fitness achievements.

CHAPTER 3

SYATEM DESIGNS

3.1. High-Level Architecture

3.1.1 System Architecture Diagram

The system architecture consists of three main layers:

1. **Presentation Layer (Frontend)**
2. **Application Layer (Backend)**
3. **Data Layer (Database)**

Architecture Diagram Components:

- **Client Side (Frontend)**: User interfaces for login, fitness data logging, progress visualization, and profile management.
- **Server Side (Backend)**: API services and business logic implemented in a server-side language (e.g., PHP, Node.js, or Django with Python).

- **Database (MySQL)**: Relational database for storing user profiles, fitness data, and goal settings.
- **APIs**: Optional integration with third-party APIs for additional fitness data sources (e.g., Google Fit).

3.2. Detailed Design

3.2.1 Database Design (MySQL)

Key Database Tables

1. **Users Table**
    - **Columns**: user_id, username, email, password_hash, age, height, weight, created_at
    - **Purpose**: Stores information related to each registered user.
2. **FitnessLogs Table**
    - **Columns**: log_id, user_id, date, steps, calories_burned, distance, heart_rate
    - **Purpose**: Logs individual daily entries of fitness metrics.
    - **Relationship**: Linked to the Users table via user_id as a foreign key.
3. **Goals Table**
    - **Columns**: goal_id, user_id, goal_type (e.g., daily steps), target_value, start_date, end_date, status
    - **Purpose**: Manages user-specific fitness goals.
    - **Relationship**: Linked to the Users table via user_id.
4. **Reports Table**
    - **Columns**: report_id, user_id, report_type (e.g., weekly, monthly), data_summary, generated_at
    - **Purpose**: Stores precomputed summaries of fitness data for quick access.

3.2.2 Backend Design

Key Modules and APIs

1. **Authentication Module**
    - **Description**: Manages user registration, login, and password recovery.
    - **APIs**:
        - POST /register: Registers a new user.
        - POST /login: Authenticates a user and generates a session or token.
        - POST /password-reset: Sends a password reset link to the user's email.
2. **User Profile Module**
    - **Description**: Allows users to update and manage their personal information.
    - **APIs**:
        - GET /user/profile: Retrieves the current user profile.
        - PUT /user/profile: Updates the user profile data.
3. **Fitness Data Module**
    - **Description**: Manages CRUD operations for fitness data entries.
    - **APIs**:
        - POST /fitness-log: Adds a new fitness entry for the user.
        - GET /fitness-log: Retrieves all fitness logs for the current user within a date range.
        - PUT /fitness-log/{log_id}: Updates an existing fitness entry.
        - DELETE /fitness-log/{log_id}: Deletes a fitness entry.
4. **Goals Module**
    - **Description**: Allows users to set, update, and track fitness goals.
    - **APIs**:

        - POST /goals: Sets a new goal for the user.
        - GET /goals: Retrieves current and past goals for the user.
        - PUT /goals/{goal_id}: Updates an existing goal.
        - DELETE /goals/{goal_id}: Deletes a goal.
5. **Reports Module**
    - **Description**: Generates and retrieves reports for weekly or monthly summaries.
    - **APIs**:
        - GET /reports: Retrieves the most recent reports for the user.
        - POST /generate-report: Generates a new report based on the user's recent data.
6. **Data Visualization Module**
    - **Description**: Prepares and formats data for frontend display, focusing on charts and graphs.
    - **Internal Function**: Queries the database for user metrics and formats data for charts (e.g., JSON format for Chart.js).

3.2.3 Frontend Design

Key Pages and Components

1. **Login & Registration Page**
    - **Components**: Input fields, validation, and authentication status messages.
    - **Functionality**: Enables new user registration and returning user login.
2. **Dashboard**
    - **Components**: Overview of fitness stats, progress summary, quick links to goals and reports.
    - **Functionality**: Shows recent activity, current goals, and links to detailed reports.

3. **Data Entry Page**
    - **Components**: Input fields for manual data entry (steps, calories, etc.).
    - **Functionality**: Allows users to log their daily metrics manually.
4. **Goals Page**
    - **Components**: Goal setting form, list of current/past goals with status.
    - **Functionality**: Users can set new fitness goals and monitor their status.
5. **Reports Page**
    - **Components**: Graphs (e.g., line, bar, pie) to visualize fitness data.
    - **Functionality**: Users can view weekly/monthly summaries of their fitness progress.
6. **Settings/Profile Page**
    - **Components**: Form for updating personal information and password change.
    - **Functionality**: Users can update profile details and set privacy settings.

Frontend Technologies

- **HTML/CSS**: For structure and styling.
- **JavaScript**: For interactive elements.
- **Libraries/Frameworks**: React or Vue.js for component-based structure; Chart.js for visualizing fitness data.

3.2.4 Security Design

1. **Authentication and Authorization**
    - **JWT Tokens**: Use JSON Web Tokens (JWT) for secure authentication and session management.

- **Password Hashing**: Hash all user passwords before storing them in the database.

2. **Data Privacy and Encryption**
    - **HTTPS**: Ensure all data transmission between client and server is encrypted.
    - **Database Security**: Use MySQL permissions to restrict database access.
3. **Input Validation and Sanitization**
    - **Backend Validation**: Validate and sanitize all input to prevent SQL injection, XSS, and other attacks.
    - **Frontend Validation**: Apply form validation for user-friendly error handling.
4. **Data Backup**
    - Regular database backups to ensure data recovery in case of server failure.

3.2.5 System Workflow

1. **User Registration and Login**:
    - User registers or logs in.
    - Server authenticates and provides a session or JWT token.
2. **Data Logging**:
    - User enters fitness data.
    - Data is validated and saved in MySQL.
3. **Goal Tracking**:
    - User sets goals (e.g., daily steps).
    - Backend tracks progress and updates the goal status.
4. **Data Visualization and Reporting**:
    - User accesses reports.

- Data is fetched from MySQL and formatted for charts on the frontend.

---

3. Deployment and Hosting

1. **Hosting Environment**
   - **Web Server**: Use Apache or Nginx.
   - **Cloud Hosting**: AWS, DigitalOcean, or similar services for scalability.
2. **Database Configuration**
   - Use a managed MySQL instance if possible for high availability.
   - Regularly monitor and optimize queries for performance.

CHAPTER 4

DATABASE DESIGN

Database Schema

1. Users Table

Stores the personal information of each user.

Column Name	Data Type	Description
user_id	INT (Primary Key, Auto Increment)	Unique identifier for each user.
username	VARCHAR(50)	Username chosen by the user.
email	VARCHAR(100)	User's email address, unique.
password_hash	VARCHAR(255)	Encrypted password for secure login.
age	INT	User's age, optional field.
height	DECIMAL(5,2)	User’s height in centimeters.
weight	DECIMAL(5,2)	User’s weight in kilograms.

Column Name	Data Type	Description
created_at	TIMESTAMP	Timestamp when the account was created.
updated_at	TIMESTAMP	Timestamp of the last profile update.

---

2. FitnessLogs Table

Stores daily fitness entries for each user.

Column Name	Data Type	Description
log_id	INT (Primary Key, Auto Increment)	Unique identifier for each fitness log entry.
user_id	INT (Foreign Key)	Links each entry to a specific user.
date	DATE	Date of the fitness log.
steps	INT	Number of steps taken.
calories_burned	DECIMAL(6,2)	Total calories burned.
distance	DECIMAL(5,2)	Distance traveled in kilometers.
heart_rate	INT	Average heart rate for the day, optional.
created_at	TIMESTAMP	Timestamp when the log entry was created.
updated_at	TIMESTAMP	Timestamp of the last update to the log.

---

3. Goals Table

Stores fitness goals set by the user.

Column Name	Data Type	Description
goal_id	INT (Primary Key, Auto Increment)	Unique identifier for each goal.
user_id	INT (Foreign Key)	Links each goal to a specific user.
goal_type	ENUM ('steps', 'calories', 'distance', 'weight')	Type of fitness goal.
target_value	DECIMAL(6,2)	Target value for the goal (e.g., 10,000 steps).
start_date	DATE	Goal start date.
end_date	DATE	Goal end date.
status	ENUM ('active', 'completed', 'missed')	Tracks goal progress.
created_at	TIMESTAMP	Timestamp when the goal was set.
updated_at	TIMESTAMP	Timestamp of the last goal update.

---

4. Reports Table

Stores precomputed weekly or monthly summaries of the user's fitness data for quick retrieval.

Column Name	Data Type	Description
report_id	INT (Primary Key, Auto Increment)	Unique identifier for each report.
user_id	INT (Foreign Key)	Links report to a specific user.
report_type	ENUM ('weekly', 'monthly')	Type of report (weekly or monthly).

Column Name	Data Type	Description
data_summary	JSON	JSON object containing summary stats (e.g., total steps, average calories).
generated_at	TIMESTAMP	Timestamp when the report was generated.

---

5. Settings Table (Optional)

Allows users to adjust settings such as notification preferences or privacy.

Column Name	Data Type	Description
setting_id	INT (Primary Key, Auto Increment)	Unique identifier for each setting entry.
user_id	INT (Foreign Key)	Links each setting entry to a user.
notifications	BOOLEAN	Whether user receives notifications (true/false).
privacy_mode	BOOLEAN	Whether user profile is set to private.
created_at	TIMESTAMP	Timestamp when settings were created.
updated_at	TIMESTAMP	Timestamp when settings were last updated.

Relationships and Foreign Keys

1. **user_id Foreign Key**:
    - The FitnessLogs, Goals, Reports, and Settings tables have a user_id column that references the user_id primary key in the Users table.
    - This relationship ensures that all fitness data, goals, reports, and settings entries are linked to a specific user.

2. **One-to-Many Relationships**:
    - Each **user** can have multiple entries in the FitnessLogs, Goals, and Reports tables.
    - The **Users** table is the “one” side, while **FitnessLogs**, **Goals**, and **Reports** are the “many” side in these relationships.
3. **Unique Constraints and Indexes**:
    - **email** in the Users table should be unique to prevent duplicate registrations.
    - **user_id** in each of the linked tables can be indexed for faster lookup.

Entity-Relationship (ER) Diagram

The ER diagram for the database design would include the following:

1. **Users**: Primary table with one-to-many relationships to FitnessLogs, Goals, Reports, and optionally, Settings.
2. **FitnessLogs**: Contains fitness metrics data with a foreign key to Users.
3. **Goals**: Stores fitness goal information with a foreign key to Users.
4. **Reports**: Contains data summaries with a foreign key to Users.
5. **Settings** (Optional): Stores user-specific settings/preferences, linked to Users via user_id.

CHAPTER 5

CODING

```
import axios from "axios";


const API = axios.create({
  baseURL: "https://fitnesstrack-vtv1.onrender.com/api/",
```

```
});

export const UserSignUp = async (data) => API.post("/user/signup", data);
export const UserSignIn = async (data) => API.post("/user/signin", data);

export const getDashboardDetails = async (token) =>
  API.get("/user/dashboard", {
    headers: { Authorization: Bearer ${token} },
  });

export const getWorkouts = async (token, date) =>
  await API.get(/user/workout${date}, {
    headers: { Authorization: Bearer ${token} },
  });

export const addWorkout = async (token, data) =>
  await API.post(/user/workout, data, {
    headers: { Authorization: Bearer ${token} },
  });




import { CircularProgress } from "@mui/material";
import React from "react";
import styled from "styled-components";
```

```
const Button = styled.div`
  border-radius: 10px;
  color: white;
  font-size: 14px;
  cursor: pointer;
  transition: all 0.3s ease;
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 6px;
  height: min-content;
  padding: 16px 26px;
  box-shadow: 1px 20px 35px 0px ${({ theme }) => theme.primary + 40};
  border: 1px solid ${({ theme }) => theme.primary};
  @media (max-width: 600px) {
    padding: 8px 12px;
  }

  ${({ type, theme }) =>
    type === "secondary"
      ? `
  background: ${theme.secondary};
border: 1px solid ${({ theme }) => theme.secondary};
  `
      : `
  background: ${theme.primary};
```

```
`}

 ${({ isDisabled }) =>
   isDisabled &&
   `
 opacity: 0.8;
 cursor: not-allowed;

 `}
 ${({ isLoading }) =>
   isLoading &&
   `
   opacity: 0.8;
 cursor: not-allowed;
`}
${({ flex }) =>
   flex &&
   `
   flex: 1;
`}

${({ small }) =>
   small &&
   `
padding: 10px 28px;
`}
```

```
  ${({ outlined, theme }) =>
    outlined &&
    `
background: transparent;
color: ${theme.primary};
  box-shadow: none;
`}
  ${({ full }) =>
    full &&
    `
  width: 100%;`}
`;

const button = ({
  text,
  isLoading,
  isDisabled,
  rightIcon,
  leftIcon,
  type,
  onClick,
  flex,
  small,
  outlined,
  full,
}) => {
```

```
  return (
    <Button
      onClick={() => !isDisabled && !isLoading && onClick()}
      isDisabled={isDisabled}
      type={type}
      isLoading={isLoading}
      flex={flex}
      small={small}
      outlined={outlined}
      full={full}
    >
      {isLoading && (
        <CircularProgress
          style={{ width: "18px", height: "18px", color: "inherit" }}
        />
      )}
      {leftIcon}
      {text}
      {isLoading && <> . . .</>}
      {rightIcon}
    </Button>
  );
};

export default button;
```

```
import React, { useState } from "react";
import styled from "styled-components";
import LogoImg from "../utils/Images/Logo.png";
import { Link as LinkR, NavLink } from "react-router-dom";
import { MenuRounded } from "@mui/icons-material";
import { Avatar } from "@mui/material";
import { useDispatch } from "react-redux";
import { logout } from "../redux/reducers/userSlice";

const Nav = styled.div`
  background-color: ${({ theme }) => theme.bg};
  height: 80px;
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 1rem;
  position: sticky;
  top: 0;
  z-index: 10;
  color: white;
  border-bottom: 1px solid ${({ theme }) => theme.text_secondary + 20};
`;
const NavContainer = styled.div`
  width: 100%;
  max-width: 1400px;
  padding: 0 24px;
```

```
  display: flex;
  gap: 14px;
  align-items: center;
  justify-content: space-between;
  font-size: 1rem;
`;
const NavLogo = styled(LinkR)`
  width: 100%;
  display: flex;
  align-items: center;
  gap: 16px;
  padding: 0 6px;
  font-weight: 600;
  font-size: 18px;
  text-decoration: none;
  color: ${({ theme }) => theme.black};
`;
const Logo = styled.img`
  height: 42px;
`;
const Mobileicon = styled.div`
  color: ${({ theme }) => theme.text_primary};
  display: none;
  @media screen and (max-width: 768px) {
    display: flex;
    align-items: center;
```

```
  }
`;

const NavItems = styled.ul`
  width: 100%;
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 32px;
  padding: 0 6px;
  list-style: none;

  @media screen and (max-width: 768px) {
    display: none;
  }
`;
const Navlink = styled(NavLink)`
  display: flex;
  align-items: center;
  color: ${({ theme }) => theme.text_primary};
  font-weight: 500;
  cursor: pointer;
  transition: all 1s slide-in;
  text-decoration: none;
  &:hover {
    color: ${({ theme }) => theme.primary};
```

```
  }
  &.active {
    color: ${({ theme }) => theme.primary};
    border-bottom: 1.8px solid ${({ theme }) => theme.primary};
  }
`;

const UserContainer = styled.div`
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: flex-end;
  gap: 16px;
  align-items: center;
  padding: 0 6px;
  color: ${({ theme }) => theme.primary};
`;
const TextButton = styled.div`
  text-align: end;
  color: ${({ theme }) => theme.secondary};
  cursor: pointer;
  font-size: 16px;
  transition: all 0.3s ease;
  font-weight: 600;
  &:hover {
    color: ${({ theme }) => theme.primary};
```

```
  }
`;

const MobileMenu = styled.ul`
  display: flex;
  flex-direction: column;
  align-items: start;
  gap: 16px;
  padding: 0 6px;
  list-style: none;
  width: 90%;
  padding: 12px 40px 24px 40px;
  background: ${({ theme }) => theme.bg};
  position: absolute;
  top: 80px;
  right: 0;
  transition: all 0.6s ease-in-out;
  transform: ${({ isOpen }) =>
    isOpen ? "translateY(0)" : "translateY(-100%)"};
  border-radius: 0 0 20px 20px;
  box-shadow: 0 0 10px 0 rgba(0, 0, 0, 0.2);
  opacity: ${({ isOpen }) => (isOpen ? "100%" : "0")};
  z-index: ${({ isOpen }) => (isOpen ? "1000" : "-1000")};
`;

const Navbar =S ({ currentUser }) => {
```

```
const dispatch = useDispatch();
const [isOpen, setisOpen] = useState(false);
return (
  <Nav>
    <NavContainer>
      <Mobileicon onClick={() => setisOpen(!isOpen)}>
        <MenuRounded sx={{ color: "inherit" }} />
      </Mobileicon>
      <NavLogo to="/">
        <Logo src={LogoImg} />
        Fittrack
      </NavLogo>

      <MobileMenu isOpen={isOpen}>
        <Navlink to="/">Dashboard</Navlink>
        <Navlink to="/workouts">Workouts</Navlink>
        <Navlink to="/tutorials">Tutorials</Navlink>
        <Navlink to="/blogs">Blogs</Navlink>
        <Navlink to="/contact">Contact</Navlink>
      </MobileMenu>

      <NavItems>
        <Navlink to="/">Dashboard</Navlink>
        <Navlink to="/workouts">Workouts</Navlink>
        <Navlink to="/tutorials">Tutorials</Navlink>
        <Navlink to="/blogs">Blogs</Navlink>
```

```
          <Navlink to="/contact">Contact</Navlink>
        </NavItems>

        <UserContainer>
          <Avatar src={currentUser?.img}>{currentUser?.name[0]}</Avatar>
          <TextButton onClick={() => dispatch(logout())}>Logout</TextButton>
        </UserContainer>
      </NavContainer>
    </Nav>
  );
};

export default Navbar;

import React, { useState } from "react";
import styled from "styled-components";
import TextInput from "./TextInput";
import Button from "./Button";
import { UserSignIn } from "../api";
import { useDispatch } from "react-redux";
import { loginSuccess } from "../redux/reducers/userSlice";

const Container = styled.div`
  width: 100%;
  max-width: 500px;
  display: flex;
```

```
  flex-direction: column;
  gap: 36px;
`;
const Title = styled.div`
  font-size: 30px;
  font-weight: 800;
  color: ${({ theme }) => theme.text_primary};
`;
const Span = styled.div`
  font-size: 16px;
  font-weight: 400;
  color: ${({ theme }) => theme.text_secondary + 90};
`;

const SignIn = () => {
  const dispatch = useDispatch();
  const [loading, setLoading] = useState(false);
  const [buttonDisabled, setButtonDisabled] = useState(false);
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  const validateInputs = () => {
    if (!email || !password) {
      alert("Please fill in all fields");
      return false;
    }
```

```
  return true;
};

const handelSignIn = async () => {
  setLoading(true);
  setButtonDisabled(true);
  if (validateInputs()) {
    await UserSignIn({ email, password })
      .then((res) => {
        dispatch(loginSuccess(res.data));
        alert("Login Success");
        setLoading(false);
        setButtonDisabled(false);
      })
      .catch((err) => {
        alert(err.response.data.message);
        setLoading(false);
        setButtonDisabled(false);
      });
  }
};

return (
  <Container>
    <div>
      <Title>Welcome to Fittrack 👏</Title>
```

```
  <Span>Please login with your details here</Span>
</div>
<div
  style={{
    display: "flex",
    gap: "20px",
    flexDirection: "column",
  }}
>
  <TextInput
    label="Email Address"
    placeholder="Enter your email address"
    value={email}
    handelChange={(e) => setEmail(e.target.value)}
  />
  <TextInput
    label="Password"
    placeholder="Enter your password"
    password
    value={password}
    handelChange={(e) => setPassword(e.target.value)}
  />
  <Button
    text="SignIn"
    onClick={handelSignIn}
    isLoading={loading}
```

```
          isDisabled={buttonDisabled}
        />
      </div>
    </Container>
  );
};

export default SignIn;



import { CloseRounded, Visibility, VisibilityOff } from "@mui/icons-material";
import React, { useState } from "react";
import styled from "styled-components";

const Container = styled.div`
  flex: 1;
  display: flex;
  flex-direction: column;
  gap: 6px;
`;

const Label = styled.label`
  font-size: 12px;
  color: ${({ theme }) => theme.text_primary};
  padding: 0px 4px;
  ${({ error, theme }) =>
```

```
    error &&
    `
    color: ${theme.red};
  `}
  ${({ small }) =>
    small &&
    `
    font-size: 8px;
  `}
  ${({ popup, theme }) =>
    popup &&
    `
  color: ${theme.popup_text_secondary};
  `}
`;

const OutlinedInput = styled.div`
  border-radius: 8px;
  border: 0.5px solid ${({ theme }) => theme.text_secondary};
  background-color: transparent;
  color: ${({ theme }) => theme.text_primary};
  outline: none;
  padding: 16px;
  display: flex;
  align-items: center;
  gap: 12px;
```

```
&:focus-within {
  border-color: ${({ theme }) => theme.secondary};
}
${({ error, theme }) =>
  error &&
  `
  border-color: ${theme.red};
`}

${({ chipableInput, height, theme }) =>
  chipableInput &&
  `
  background: ${theme.card};
  flex-direction: column;
  align-items: flex-start;
  gap: 8px;
  min-height: ${height}
`}

${({ small }) =>
  small &&
  `
  border-radius: 6px;
  padding: 8px 10px;
`}
```

```
  ${({ popup, theme }) =>
    popup &&
    `
  color: ${theme.popup_text_secondary};
  border: 0.5px solid ${theme.popup_text_secondary + 60};
  `}
`;

const Input = styled.input`
  width: 100%;
  font-size: 14px;
  outline: none;
  border: none;
  background-color: transparent;
  color: ${({ theme }) => theme.text_primary};
  &:focus {
    outline: none;
  }
  ${({ small }) =>
    small &&
    `
    font-size: 12px;
  `}

  ${({ popup, theme }) =>
    popup &&
```

```
    `
  color: ${theme.popup_text_secondary};
  `} ${({ theme }) => theme.popup_text_secondary};
`;

const Error = styled.p`
  font-size: 12px;
  margin: 0px 4px;
  color: ${({ theme }) => theme.red};
  ${({ small }) =>
    small &&
    `
    font-size: 8px;
  `}
`;

const ChipWrapper = styled.div`
  display: flex;
  flex-wrap: wrap;
  gap: 6px;
`;

const Chip = styled.div`
  padding: 5px 10px;
  border-radius: 8px;
  background: ${({ theme }) => theme.primary + 10};
```

```
  color: ${({ theme }) => theme.primary};
  font-size: 12px;
  display: flex;
  align-items: center;
  gap: 4px;
  cursor: pointer;
  transition: all 0.3s ease;
`;

const TextInput = ({
  label,
  placeholder,
  name,
  value,
  error,
  handelChange,
  textArea,
  rows,
  columns,
  chipableInput,
  chipableArray,
  removeChip,
  height,
  small,
  popup,
  password,
```

```
}) => {
  const [showPassword, setShowPassword] = useState(false);
  return (
    <Container small={small}>
      <Label small={small} popup={popup} error={error}>
        {label}
      </Label>
      <OutlinedInput
        small={small}
        popup={popup}
        error={error}
        chipableInput={chipableInput}
        height={height}
      >
        {chipableInput ? (
          <ChipWrapper>
            {chipableArray.map((chip, index) => (
              <Chip key={index}>
                <span>{chip}</span>
                <CloseRounded
                  sx={{ fontSize: "14px" }}
                  onClick={() => removeChip(name, index)}
                />
              </Chip>
            ))}
            <Input
```

```
      placeholder={placeholder}
      name={name}
      value={value}
      onChange={(e) => handelChange(e)}
    />
  </ChipWrapper>
) : (
  <>
    <Input
      popup={popup}
      small={small}
      as={textArea ? "textarea" : "input"}
      name={name}
      rows={rows}
      columns={columns}
      placeholder={placeholder}
      value={value}
      onChange={(e) => handelChange(e)}
      type={password && !showPassword ? "password" : "text"}
    />
    {password && (
      <>
        {showPassword ? (
          <>
            <Visibility onClick={() => setShowPassword(false)} />
          </>
```

```
            ) : (
              <>
                <VisibilityOff onClick={() => setShowPassword(true)} />
              </>
            )}
          </>
        )}
      </>
    )}
    </OutlinedInput>
    {error && (
      <Error small={small} popup={popup}>
        {error}
      </Error>
    )}
  </Container>
 );
};

export default TextInput;



import React, { useState } from "react";
import styled from "styled-components";
import LogoImage from "../utils/Images/Logo.png";
import AuthImage from "../utils/Images/AuthImage.jpg";
```

```
import SignIn from "../components/SignIn";
import SignUp from "../components/SignUp";

const Container = styled.div`
  flex: 1;
  height: 100%;
  display: flex;
  background: ${({ theme }) => theme.bg};
  @media (max-width: 700px) {
    flex-direction: column;
  }
`;
const Left = styled.div`
  flex: 1;
  position: relative;
  @media (max-width: 700px) {
    display: none;
  }
`;
const Logo = styled.img`
  position: absolute;
  width: 70px;
  top: 40px;
  left: 60px;
  z-index: 10;
`;
```

```
const Image = styled.img`
  position: relative;
  height: 100%;
  width: 100%;
  object-fit: cover;
`;

const Right = styled.div`
  flex: 1;
  position: relative;
  display: flex;
  flex-direction: column;
  padding: 40px;
  gap: 16px;
  align-items: center;
  justify-content: center;
`;

const Text = styled.div`
  font-size: 16px;
  text-align: center;
  color: ${({ theme }) => theme.text_secondary};
  margin-top: 16px;
  @media (max-width: 400px) {
    font-size: 14px;
  }
```

```
`;
const TextButton = styled.span`
  color: ${({ theme }) => theme.primary};
  cursor: pointer;
  transition: all 0.3s ease;
  font-weight: 600;
`;

const Authentication = () => {
  const [login, setLogin] = useState(false);
  return (
    <Container>
      <Left>
        <Logo src={LogoImage} />
        <Image src={AuthImage} />
      </Left>
      <Right>
        {!login ? (
          <>
            <SignIn />
            <Text>
              Don't have an account?{" "}
              <TextButton onClick={() => setLogin(true)}>SignUp</TextButton>
            </Text>
          </>
        ) : (
```

```
          <>
            <SignUp />
            <Text>
              Already have an account?{" "}
              <TextButton onClick={() => setLogin(false)}>SignIn</TextButton>
            </Text>
          </>
        )}
      </Right>
    </Container>
  );
};

export default Authentication;



import React, { useEffect, useState } from "react";
import styled from "styled-components";
import { counts } from "../utils/data";
import CountsCard from "../components/cards/CountsCard";
import WeeklyStatCard from "../components/cards/WeeklyStatCard";
import CategoryChart from "../components/cards/CategoryChart";
import AddWorkout from "../components/AddWorkout";
import WorkoutCard from "../components/cards/WorkoutCard";
import { addWorkout, getDashboardDetails, getWorkouts } from "../api";
```

```
const Container = styled.div`
  flex: 1;
  height: 100%;
  display: flex;
  justify-content: center;
  padding: 22px 0px;
  overflow-y: scroll;
`;
const Wrapper = styled.div`
  flex: 1;
  max-width: 1400px;
  display: flex;
  flex-direction: column;
  gap: 22px;
  @media (max-width: 600px) {
    gap: 12px;
  }
`;
const Title = styled.div`
  padding: 0px 16px;
  font-size: 22px;
  color: ${({ theme }) => theme.text_primary};
  font-weight: 500;
`;
const FlexWrap = styled.div`
```

```
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;
  gap: 22px;
  padding: 0px 16px;
  @media (max-width: 600px) {
    gap: 12px;
  }
`;
const Section = styled.div`
  display: flex;
  flex-direction: column;
  padding: 0px 16px;
  gap: 22px;
  padding: 0px 16px;
  @media (max-width: 600px) {
    gap: 12px;
  }
`;
const CardWrapper = styled.div`
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  gap: 20px;
  margin-bottom: 100px;
  @media (max-width: 600px) {
```

```
    gap: 12px;
  }
`;

const Dashboard = () => {
  const [loading, setLoading] = useState(false);
  const [data, setData] = useState();
  const [buttonLoading, setButtonLoading] = useState(false);
  const [todaysWorkouts, setTodaysWorkouts] = useState([]);
  const [workout, setWorkout] = useState(`#Legs
-Back Squat
-5 setsX15 reps
-30 kg
-10 min`);

  const dashboardData = async () => {
    setLoading(true);
    const token = localStorage.getItem("fittrack-app-token");
    await getDashboardDetails(token).then((res) => {
      setData(res.data);
      console.log(res.data);
      setLoading(false);
    });
  };
  const getTodaysWorkout = async () => {
    setLoading(true);
```

```
  const token = localStorage.getItem("fittrack-app-token");
  await getWorkouts(token, "").then((res) => {
    setTodaysWorkouts(res?.data?.todaysWorkouts);
    console.log(res.data);
    setLoading(false);
  });
};

const addNewWorkout = async () => {
  setButtonLoading(true);
  const token = localStorage.getItem("fittrack-app-token");
  await addWorkout(token, { workoutString: workout })
    .then((res) => {
      dashboardData();
      getTodaysWorkout();
      setButtonLoading(false);
    })
    .catch((err) => {
      alert(err);
    });
};

useEffect(() => {
  dashboardData();
  getTodaysWorkout();
}, []);
```

```
return (
  <Container>
    <Wrapper>
      <Title>Dashboard</Title>
      <FlexWrap>
        {counts.map((item) => (
          <CountsCard item={item} data={data} />
        ))}
      </FlexWrap>

      <FlexWrap>
        <WeeklyStatCard data={data} />
        <CategoryChart data={data} />
        <AddWorkout
          workout={workout}
          setWorkout={setWorkout}
          addNewWorkout={addNewWorkout}
          buttonLoading={buttonLoading}
        />
      </FlexWrap>

      <Section>
        <Title>Todays Workouts</Title>
        <CardWrapper>
          {todaysWorkouts.map((workout) => (
            <WorkoutCard workout={workout} />
```

```
          ))}
        </CardWrapper>
      </Section>
    </Wrapper>
  </Container>
 );
};

export default Dashboard;



import { createSlice } from "@reduxjs/toolkit";

const initialState = {
 currentUser: null,
};

export const userSlice = createSlice({
 name: "user",
 initialState,
 reducers: {
  loginSuccess: (state, action) => {
   state.currentUser = action.payload.user;
   localStorage.setItem("fittrack-app-token", action.payload.token);
  },
  logout: (state) => {
```

```
      state.currentUser = null;
      localStorage.removeItem("fitttrack-app-token");
    },
  },
});

export const { loginSuccess, logout } = userSlice.actions;

export default userSlice.reducer;



export const lightTheme = {
  bg: "#FFFFFF",
  bgLight: "#FFFFFF",
  primary: "#007AFF",
  secondary: "#5B86E5",

  disabled: "#b1b2b3",
  menubar: "#191c29",
  navbar: "#242B3F",
  arrow: "#AFAFB5",
  menu_primary_text: "#F2F3F4",
  menu_secondary_text: "#b1b2b3",
  table_header: "#242445",
  text_primary: "#404040",
  text_secondary: "#4d4c4c",
```

```
  card: "#FFFFFF",
  black: "#000000",
  white: "#FFFFFF",
  shadow: "#00000020",
  green: "#00ff6a",
  yellow: "#e8ba00",
  red: "#ef5350",
  orange: "#F7AD63",
  popup: "#242B3F",
  popup_text_primary: "#F2F3F4",
  popup_text_secondary: "#b1b2b3",
  output_node: "#49516b",
};


import { ThemeProvider, styled } from "styled-components";
import { lightTheme } from "./utils/Themes";
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Authentication from "./pages/Authentication";
import { useState } from "react";
import { useSelector } from "react-redux";
import Navbar from "./components/Navbar";
import Dashboard from "./pages/Dashboard";
import Workouts from "./pages/Workouts";

const Container = styled.div`
```

```
  width: 100%;
  height: 100vh;
  display: flex;
  flex-direction: column;
  background: ${({ theme }) => theme.bg};
  color: ${({ theme }) => theme.text_primary};
  overflow-x: hidden;
  overflow-y: hidden;
  transition: all 0.2s ease;
`;

function App() {
  const { currentUser } = useSelector((state) => state.user);
  return (
    <ThemeProvider theme={lightTheme}>
      <BrowserRouter>
        {currentUser ? (
          <Container>
            <Navbar currentUser={currentUser} />
            <Routes>
              <Route path="/" exact element={<Dashboard />} />
              <Route path="/workouts" exact element={<Workouts />} />
            </Routes>
          </Container>
        ) : (
          <Container>
```

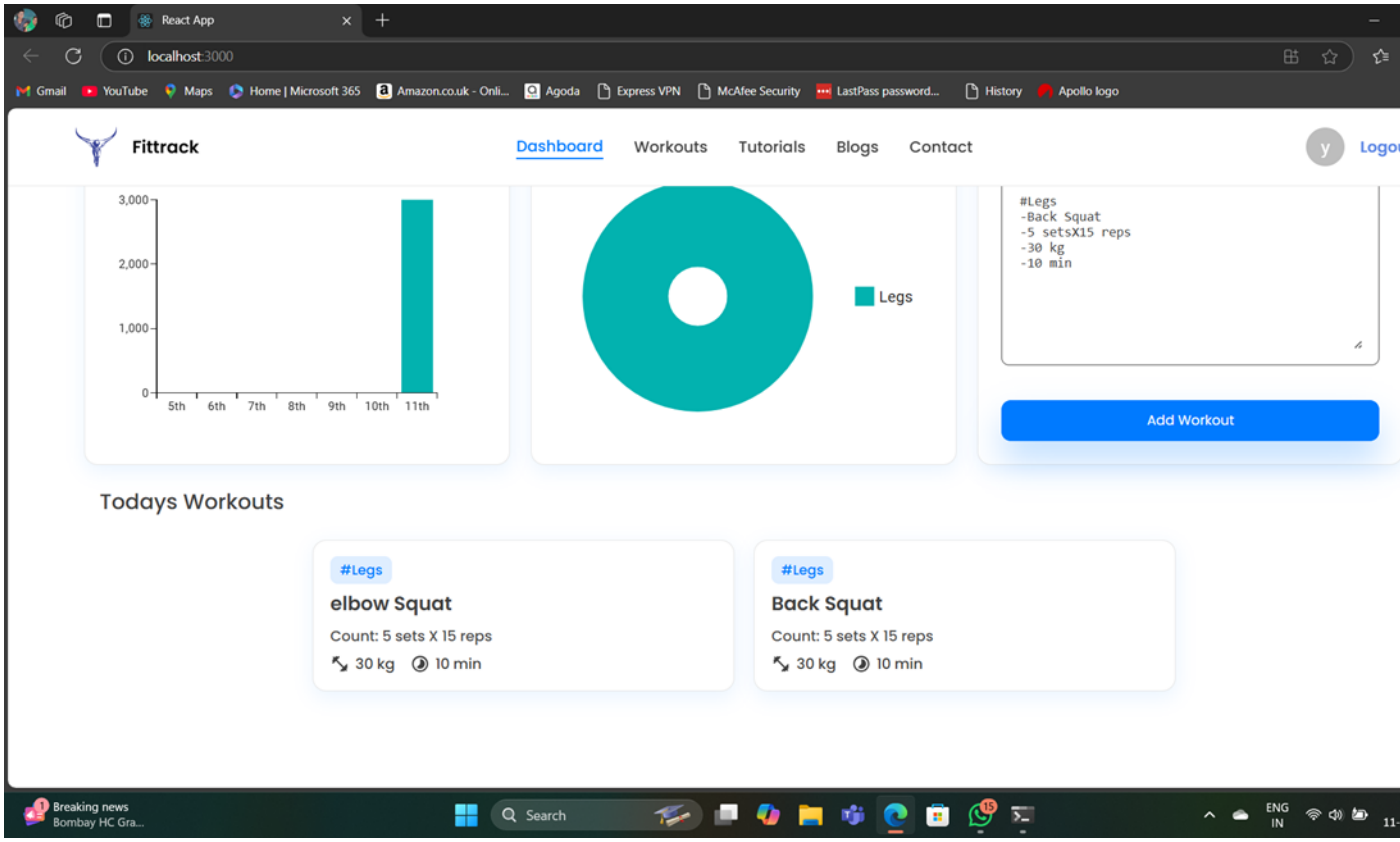
```
          <Authentication />
        </Container>
      )}
    </BrowserRouter>
  </ThemeProvider>
 );
}

export default App;
```

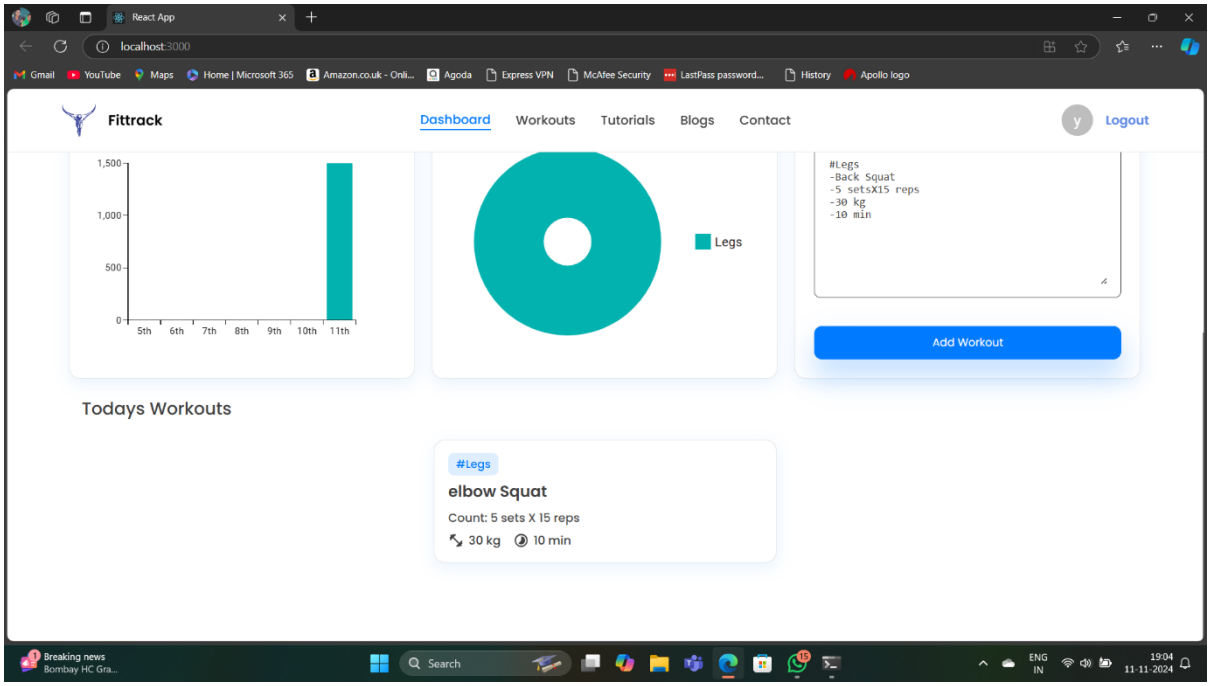
```
@import
url("https://fonts.googleapis.com/css2?family=Poppins:wght@100;300;400;50
0;600;700&display=swap");
html {
 scroll-behavior: smooth;
}
body {
 margin: 0;
 padding: 0;
 font-family: "Poppins", sans-serif;
 -webkit-font-smoothing: antialiased;
 -moz-osx-font-smoothing: grayscale;
}

/* width */
::-webkit-scrollbar {
 width: 2px;
```

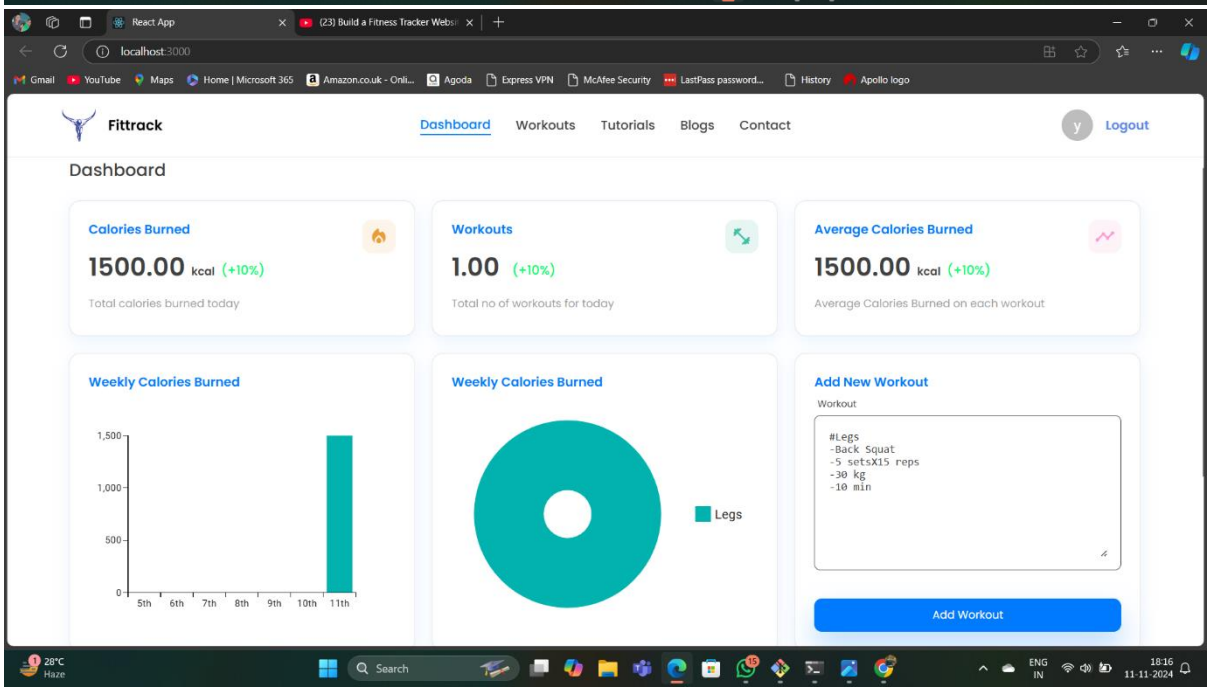
```
}
/* Track */
::-webkit-scrollbar-track {
}

/* Handle */
::-webkit-scrollbar-thumb {
  background: #888;
  border-radius: 6px;
  height: 50px;
}
```

CHAPTER 6

SCREENSHOTS







CHAPTER 7

CONCLUSIONS

Enhanced User Experience: The Fitness Tracker application provides users with an interactive and user-friendly platform for monitoring their health metrics. By allowing users to log daily fitness data like steps, calories burned, and heart rate, the application promotes health awareness and goal setting.

Scalability and Flexibility: Built with a scalable backend (Node.js and MySQL), the application can be expanded to accommodate more features, such as activity tracking, social sharing, and personalized health recommendations. This flexibility ensures it can grow with user needs and advancements in fitness tracking.

Data Security and Privacy: With JWT-based authentication and password encryption using bcrypt, the application prioritizes user data security and privacy. This is essential to build trust among users and ensure they feel comfortable storing personal health information in the app.

Learning and Skill Development: The project involved using essential web development technologies, including MySQL for relational data management and Express for backend functionality. Working on this project strengthened knowledge in both backend and frontend development, along with database integration.

Potential for Future Enhancements: The Fitness Tracker application has laid a strong foundation for future developments. Upcoming enhancements could include data visualization, integration with wearable devices, personalized goal setting, and AI-driven health insights. This opens up exciting possibilities for future updates and feature additions.

Challenges and Limitations: While the application effectively tracks basic fitness metrics, it has limitations in advanced functionalities, such as real-time data syncing from devices or comprehensive data analysis. These areas provide avenues for further research and development.