Project Proposal: Fitness Tracker Dashboard with React

1. Project Planning & Management

Project Proposal

Overview

The Fitness Tracker Dashboard is a web application designed to help users monitor their workout activities, set fitness goals, and track progress over time. The application will feature an interactive and user-friendly interface built with React, integrated with a Node.js backend for managing user data. It will include data visualization through charts and graphs, allowing users to see their progress in an intuitive way.

Objective

- Develop a fitness tracking dashboard to provide real-time statistics and goal monitoring.
- Enable users to log daily workouts, set fitness goals, and track their improvements.
- Implement secure user authentication so individuals can manage their own records.
- Use charts and graphs (Chart.js or D3.js) for clear data visualization.

Scope

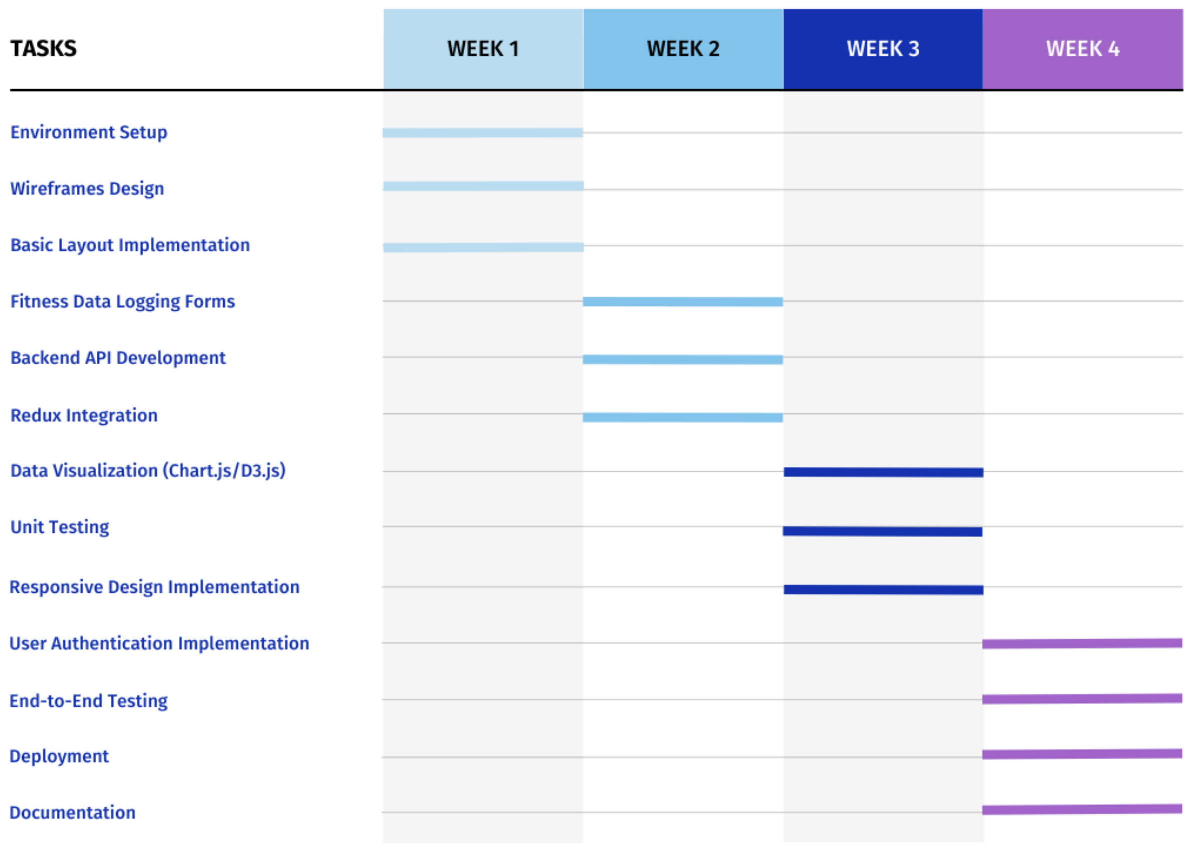
The project will cover:

- **User Registration & Authentication** – Secure login and personalized access.
- **Workout Logging System** – Users can input and update their daily activities.
- **Fitness Goal Tracking** – Set and track progress toward fitness milestones.
- **Data Visualization** – Graphs and charts to display real-time progress.
- **Backend Integration** – Node.js for managing user data and interactions.
- **API Integration** – Connecting external fitness APIs for additional insights.

Fitness Tracker Dashboard Project Plan

This project plan outlines the development of a fitness tracker dashboard using React, Node.js, and other specified technologies.

Project Timeline :



Milestones :

- **Week 1 :** Project Setup Complete - Environment configured, wireframes designed, and basic layout implemented.
- **Week 2 :** Core Functionality Implemented - Data logging forms functional, backend API operational, and Redux integrated.
- **Week 3 :** Visualizations and Testing Complete - Charts and graphs integrated, unit tests passed, and responsive design implemented.

- **Week 4 :** Project Completion - User authentication implemented, end-to-end testing complete, application deployed, and documentation finalized.

Deliverables :

- **Week 1 :**
    1. React and Node.js environment set up.
    2. Wireframes for the fitness dashboard.
    3. Basic layout for the dashboard and input forms.
- **Week 2 :**
    1. Functional form for logging fitness data.
    2. Backend API connected to MongoDB.
    3. Redux managing global fitness data.
- **Week 3 :**
    1. Graphs and charts visualizing fitness progress.
    2. Tested and working components.
    3. Responsive layout for the fitness dashboard.
- **Week 4 :**
    1. User authentication integrated.
    2. Deployed fitness tracker app.
    3. Complete documentation with API details and user instructions.

Resource Allocation

- **Frontend Developer :** Responsible for React development, UI implementation, data visualization, and responsive design. (All weeks)
- **Backend Developer :** Responsible for Node.js API development, database integration, and user authentication. (Weeks 2-4)
- **Tester :** Responsible for unit testing and end-to-end testing. (Weeks 3-4)
- **Project Manager :** Oversees project progress, manages communication, and ensures timely delivery. (All weeks)

Task Assignment & Roles

General Tasks

Task	Role	Assigned To
UI/UX Wireframe	UI/UX	Omnya Tarek
MongoDB Tables ERD Diagram	Database	Mariam Helmy / Esraa Mostafa
Dockerize Project	Docker	Samah Ali
Create a Project on GitHub	GitHub	Samah Ali

Backend Development

Task	Role	Assigned To
Sign-up Backend Endpoint	Backend	Samah Ali
Unit Test: Sign-up Backend Endpoint	Backend	Samah Ali
Sign-in Backend Endpoint	Backend	Esraa Mostafa
Unit Test: Sign-in Backend Endpoint	Backend	Esraa Mostafa
Profile Backend Endpoint - Add User Fitness Data	Backend	Mariam Helmy
Unit Test: Profile Add User Fitness Data	Backend	Mariam Helmy
Profile Backend Endpoint - Edit User Fitness Data	Backend	Omnya Tarek
Unit Test: Profile Edit User Fitness Data	Backend	Omnya Tarek
Profile Backend Endpoint - View User Fitness Data	Backend	Samah Ali

Unit Test: Profile View User Fitness Data	Backend	Samah Ali
Profile Backend Endpoint - Delete User Fitness Data	Backend	Esraa Mostafa
Unit Test: Profile Delete User Fitness Data	Backend	Esraa Mostafa
Profile Backend Endpoint - List All User Fitness Data	Backend	Mariam Helmy
Unit Test: Profile List All User Fitness Data	Backend	Mariam Helmy
Reset Password Backend Endpoint	Backend	Omnya Tarek
Forget Password Backend Endpoint	Backend	Omnya Tarek
Unit Test: Forget Password Backend Endpoint	Backend	Samah Ali
Unit Test: Reset Password Backend Endpoint	Backend	Samah Ali

Frontend Development

Task	Role	Assigned To
Sign-in Form Design + Call Backend Endpoint	Frontend	Esraa Mostafa
Unit Test: Sign-in Form Call Backend	Frontend	Esraa Mostafa
Sign-up Form Design + Call Backend Endpoint	Frontend	Mariam Helmy
Unit Test: Sign-up Form Call Backend	Frontend	Mariam Helmy
Profile Form Design - Add User Fitness Data	Frontend	Omnya Tarek
Unit Test: Profile Add User Fitness Data	Frontend	Omnya Tarek
Profile Edit Mode - Call Backend Endpoint	Frontend	Samah Ali
Unit Test: Profile Edit User Fitness Data	Frontend	Samah Ali
Profile View Mode - Call Retrieve Endpoint	Frontend	Esraa Mostafa
Unit Test: Profile Retrieve User Fitness Data	Frontend	Esraa Mostafa

Profile Delete Specific Fitness Record - Call Backend	Frontend	Mariam Helmy
Unit Test: Profile Delete Specific Fitness Record	Frontend	Mariam Helmy
Dashboard Design - List All User Fitness Data	Frontend	Omnya Tarek
Unit Test: Dashboard List All User Fitness Data	Frontend	Omnya Tarek
Reset Password Form Design + Call Backend Endpoint	Frontend	Samah Ali
Unit Test: Reset Password Call Backend	Frontend	Samah Ali
Forget Password Form Design + Call Backend Endpoint	Frontend	Esraa Mostafa
Unit Test: Forget Password Call Backend	Frontend	Esraa Mostafa

Risk Assessment & Mitigation Plan – Fitness Tracker Dashboard

Risks	**Solutions**
Data Integrity Risks Loss or corruption of user data due to system crashes or database issues.	Implement automated database backups, use transaction management in the database, validate all user inputs, and employ error handling mechanisms to prevent data corruption.
Scalability Risks System may not handle increased users and data over time.	Use load balancing, optimize API endpoints and consider cloud-based solutions like AWS or Firebase for scalability.
User Experience (UX) Risks Poor UI/UX design leading to low user engagement.	Conduct user testing, gather feedback, implement responsive design principles, and continuously iterate on design improvements based on analytics.
Third-Party Dependency Risks	Regularly update dependencies, monitor security advisories, have fallback

External libraries (Chart.js, D3.js) may become outdated or introduce vulnerabilities.	solutions, and consider alternative libraries if needed.
Project Timeline & Delivery Risks Delays in development due to unforeseen technical challenges or scope creep.	Follow Agile methodology, set clear milestones, conduct regular sprint reviews, allocate buffer time for unexpected issues, and ensure strong project management practices. By proactively addressing these risks with effective solutions, the **Fitness Tracker Dashboard** will be secure, efficient, and user-friendly, ensuring a seamless experience for users.

KPIs (Key Performance Indicators)

1. System Performance & Reliability

✅ **API Response Time:** Backend API calls should return responses within **≤ 200ms** on average.
✅ **System Uptime:** Maintain **99.9% uptime** for backend services.
✅ **Error Rate:** Less than **1% API failure rate** (measured via logs & monitoring tools).
✅ **Database Query Performance:** MongoDB queries should execute within **≤ 500ms**.
✅ **Page Load Speed:** Ensure the dashboard loads in **≤ 2 seconds**.

2. Code Quality & Testing

✅ **Unit Test Coverage:** Achieve at least **85% test coverage** for backend & frontend.
✅ **Bug Resolution Time:** Critical bugs resolved within **24 hours**, high-priority within **48 hours**.
✅ **Code Review Turnaround:** PRs reviewed & merged within **48 hours**.
✅ **CI/CD Build Success Rate:** At least **90% successful builds** in the continuous deployment pipeline.

3. User Adoption & Engagement

✅ **User Sign-ups:** At least **X new users per week** (defined based on goals).
✅ **Daily Active Users (DAU):** Aim for **at least 50% of signed-up users** logging workouts daily.
✅ **Feature Usage Rate: 80% of users log fitness data at least once per week**.
✅ **User Retention Rate:** At least **60% of users return** within a month.
✅ **Avg. Session Duration:** Users spend at least **5 minutes per session** engaging with charts and data.

4. Security & Compliance

✅ **Authentication Success Rate: Less than 2% login failures**.
✅ **Security Vulnerabilities:** No high-severity issues detected in security scans.
✅ **Data Privacy Compliance:** Ensure compliance with **GDPR (if applicable)**.
✅ **API Authorization:** No unauthorized data access incidents.

5. Deployment & Infrastructure

✅ **Deployment Frequency:** Push new features & fixes **at least every 2 weeks**.
✅ **Scalability:** System should handle **100+ concurrent users** without performance degradation.
✅ **Infrastructure Cost Optimization:** Ensure **server costs do not exceed budgeted limits**.

6. Dashboard & Data Visualization

✅ **Data Accuracy:** Fitness data displayed on charts must match database records **100%**.
✅ **Chart Load Time:** Charts render within **2 second** after API response.
✅ **Responsiveness:** Dashboard UI must pass **100% responsive tests** across devices.

3. Requirements Gathering

a. Stakeholder Analysis

Internal	External
Owner	Shareholders (Investors or Business Partners)
Engineers	Suppliers (Third-Party API Providers & Hosting Services)
Employees	Customers (End Users - Fitness Enthusiasts & Trainers):
Competitors	Health & Fitness Organizations (Gyms, Personal Trainers, Wellness Coaches

- **Internal Stakeholder**

- **Owner:**
  - **Interests:** Responsible for project funding, overseeing development, ensuring product value, and making strategic decisions regarding software versions and updates.
  - **Interdependence:** Plays a key role in the success, continuity, and growth of the project by managing resources and setting long-term goals.
- **Engineers (Developers):**
  - **Interests:** Responsible for designing, developing, and maintaining the fitness tracking dashboard, ensuring smooth functionality, security, and user experience.
  - **Interdependence:** Ensure product quality, performance, and usability meet project requirements and user expectations.
- **Employees (Development Team):**
  - **Interests:** Execute assigned tasks, such as coding, UI/UX design, API integration, and testing, to ensure the project's successful completion.
  - **Interdependence:** Work collaboratively to develop and deliver a high-quality application, ensuring all features function as intended.
- **Competitors:**

- **Interests:** Provide similar fitness tracking solutions, often at competitive prices or with enhanced features, to capture a larger market share.
- **Interdependence:** Their presence drives innovation and continuous improvement of the fitness tracker dashboard to maintain competitiveness in the market.

- **External Stakeholders**

1. **Shareholders (Investors or Business Partners):**
   - **Interests:** Generate profit from the platform, contribute to its expansion, and support marketing efforts to attract more users.
   - **Interdependence:** Their investment and support contribute to the financial growth and sustainability of the project.
2. **Suppliers (Third-Party API Providers & Hosting Services):**
   - **Interests:** Provide essential APIs (e.g., fitness data APIs, authentication services) and hosting solutions to support the platform's performance.
   - **Interdependence:** Ensuring reliable API services and hosting enhances the functionality and accessibility of the dashboard.
3. **Customers (End Users - Fitness Enthusiasts & Trainers):**
   - **Interests:** Use the platform to track fitness goals, log workout data, and analyze progress through charts and statistics.
   - **Interdependence:** Their engagement and feedback drive feature updates and improvements, making the product more successful.
4. **Health & Fitness Organizations (Gyms, Personal Trainers, Wellness Coaches):**
   - **Interests:** Utilize the platform to manage client progress, recommend workouts, and track fitness performance.
   - **Interdependence:** Increased adoption by gyms and trainers enhances platform credibility, user base, and overall success.

b. User Stories & Use Cases

1. User Authentication & Profile Management

User Story 1: Sign Up & Login

📌 **As a** user, I want to sign up and log in securely to access my personalized fitness data.

Use Case

- **Trigger:** The user visits the login/signup page.
- **Steps:**
  1. The user enters email, password, and optional profile details.
  2. The system validates credentials and creates an account or logs in.
  3. The user is redirected to the dashboard.
- **Success:** The user successfully logs in and accesses their profile.
- **Failure:** Incorrect credentials show an error message.

User Story 2: Reset & Forget Password

📌 **As a** user, I want to reset my password in case I forget it so that I can regain access.

Use Case

- **Trigger:** The user clicks "Forgot Password" on the login screen.
- **Steps:**
  1. The user enters their registered email.
  2. The system sends a password reset link.
  3. The user clicks the link and sets a new password.
- **Success:** The user logs in with the new password.
- **Failure:** Expired or incorrect reset link.

2. Fitness Data Logging & Tracking

User Story 3: Log Workout Activities

📌 **As a** user, I want to log my daily workout activities so that I can track my fitness progress.

Use Case

- **Trigger:** User visits the "Log Activity" section.

- **Steps:**
    1. The user selects the activity type (running, cycling, weightlifting, etc.).
    2. User inputs details (duration, calories burned, distance, etc.).
    3. The user submits the entry, and the system stores it.
- **Success:** Entry appears in the dashboard.
- **Failure:** Invalid inputs prompt an error message.

User Story 4: Edit/Delete Workout Entries

📌 **As a** user, I want to modify or delete my fitness entries so that I can keep my records accurate.

Use Case

- **Trigger:** The user selects an existing workout entry.
- **Steps:**
    1. The user clicks "Edit" and updates the entry.
    2. The user clicks "Delete" to remove the entry.
    3. The system updates or removes the entry.
- **Success:** Updated or deleted entries are reflected on the dashboard.
- **Failure:** Unauthorized access prevents modifications.

---

3. Fitness Goals & Progress Tracking

User Story 5: Set Fitness Goals

📌 **As a** user, I want to set fitness goals so that I can track my progress over time.

Use Case

- **Trigger:** The user navigates to the "Set Goals" section.
- **Steps:**
    1. The user sets a goal (e.g., run 5km daily, lose 5kg in a month).
    2. The system stores the goal and tracks progress.
    3. The dashboard displays the goal with a progress bar.
- **Success:** The goal is saved and displayed on the dashboard.
- **Failure:** Invalid goal inputs prevent saving.

User Story 6: Track Progress with Charts

📌 **As a** user, I want to see my fitness progress visualized in graphs so that I can analyze my improvements.

Use Case

- **Trigger:** The user opens the dashboard.
- **Steps:**
    1. The system retrieves fitness data.
    2. Charts display weekly/monthly progress.
    3. The user interacts with filters to customize views.
- **Success:** Accurate data visualization helps users analyze trends.
- **Failure:** The empty dataset displays a "No data available" message.

---

4. User Profile & Data Management

User Story 7: View & Update Profile

📌 **As a** user, I want to update my profile so that I can personalize my experience.

Use Case

- **Trigger:** User navigates to "Profile" settings.
- **Steps:**
    1. User updates details (name, age, weight, fitness level).
    2. The system validates and saves changes.
- **Success:** Updated profile reflects immediately.
- **Failure:** Invalid inputs show an error.

5. System Administration & Monitoring

User Story 8: Monitor System Uptime & Performance

📌 **As a** system administrator, I want to monitor API response time and server health so that I can ensure smooth operation.

Use Case

- **Trigger:** Admin accesses monitoring tools.
- **Steps:**
    1. The system logs API response times and uptime.

2. Alerts notify if response time exceeds 200ms.
3. Admin reviews logs and takes corrective action.

- **Success:** The system operates within performance thresholds.
- **Failure:** Alerts notify the admin of slow performance.

c. Functional Requirements:

1. User Registration and Account Management:

- **User Registration:**
  - The user should be able to create a new account using a username/email and password.
  - Email validation should be implemented (optional).
- **User Login:**
  - The user should be able to log in using their username/email and password.
  - A "Forgot Password" option should be provided.
- **Profile Editing:**
  - The user should be able to edit their profile information (e.g., name, height, weight).
- **User Logout:**
  - The user should be able to log out of their account.

2. Fitness Data Logging:

- **Activity Data Entry:**

    - The user should be able to enter daily fitness activity data (e.g., activity type, duration, calories burned, steps, distance).
    - The user should be able to specify the activity date.
- **Activity Log Display:**
    - The user should be able to view a log of recorded fitness activities.
    - The user should be able to edit and delete activity data.

3. Goal Management:

- **Goal Setting:**
    - The user should be able to set fitness goals (e.g., daily step count, calories burned, workout time).
    - The user should be able to set a goal start and end date.
- **Goal Display:**
    - The user should be able to view their set goals and their progress towards achieving them.
- **Goal Editing and Deletion:**
    - The user should be able to edit and delete their goals.

4. Data Visualization and Display:

- **Daily Activity Summary:**
    - The system should display a summary of daily fitness activity (steps, calories burned, workout time).
- **Chart Display:**
    - The system should display charts to visualize the user's progress in activities and goals over time (weekly, monthly).

    - Chart.js or D3.js should be used to create the charts.
- **Detailed Data Display:**
    - The system should display activity data in a detailed format.

5. User Interface and Interaction:

- **Responsive UI:**
    - The UI should be responsive and function correctly on various devices (desktops, tablets, smartphones).
- **User Interaction:**
    - The system should provide a smooth and user-friendly interaction.
    - Visual feedback should be provided to the user upon interaction.

6. API Backend:

- **Data Storage:**
    - The API should provide the ability to store user, activity, and goal data in a MongoDB database.
- **Data Retrieval:**
    - The API should provide the ability to retrieve user, activity, and goal data.
- **Data Update and Deletion:**
    - The API should provide the ability to update and delete data.
- **Data Validation:**
    - The API should validate incoming data.

7. Authentication:

- **User Authentication:**
    - The system should provide user authentication to protect user data.
    - Secure password storage methods should be used.

d. Non-Functional Requirements

These requirements define the qualities of the system, rather than its specific features.

1. Performance

- Response Time:
    - The dashboard should load within 3 seconds.
    - API requests (e.g., logging data, fetching data) should complete within 2 seconds.
    - Chart rendering should complete within 2 seconds.
- Scalability:
    - The system should be able to handle a growing number of users and data entries without significant performance degradation.
    - The backend API should be designed to support horizontal scaling.
- Efficiency:
    - Minimize the use of resources (CPU, memory, network bandwidth) on both the client and server sides.
    - Optimize database queries for fast data retrieval.

2. Security

- Authentication:

    - User authentication should be implemented using secure methods (e.g., JWT).
    - Passwords should be securely hashed and stored.
    - Implement secure password reset functionality.
- Authorization:
    - Users should only be able to access and modify their own fitness data.
    - Implement role-based access control if future versions include admin or coach accounts.
- Data Protection:
    - Sensitive data (e.g., user credentials, personal fitness information) should be encrypted both in transit (HTTPS) and at rest (database encryption).
    - Protect against common web vulnerabilities (e.g., XSS, CSRF, SQL injection).
    - Sanitize user inputs to prevent injection attacks.
- Data Privacy:
    - Comply with relevant data privacy regulations (e.g., GDPR, CCPA).
    - Provide users with clear information about how their data is collected and used.

3. Usability

- User Interface (UI):
    - The dashboard should have a clean, intuitive, and user-friendly interface.
    - Navigation should be clear and consistent.

    - Data input forms should be easy to use and provide clear feedback.
    - Visualizations should be easy to understand.
- Accessibility:
    - The application should be accessible to users with disabilities, adhering to WCAG guidelines.
    - Provide alternative text for images and charts.
    - Ensure keyboard navigation and screen reader compatibility.
- Responsiveness:
    - The dashboard should be fully responsive and adapt to different screen sizes and devices (desktops, tablets, mobile phones).
- Error Handling:
    - Provide clear and informative error messages to users.
    - Implement proper input validation to prevent errors.

4. Reliability

- Availability:
    - The system should be available 99.9% of the time.
    - Implement monitoring and alerting to detect and resolve issues quickly.
- Data Integrity:
    - Ensure that fitness data is stored and retrieved accurately.
    - Implement data validation and error handling to prevent data corruption.
- Maintainability:
    - The code should be well-structured, documented, and easy to maintain.

    - Use a modular design to facilitate future updates and enhancements.
- Testability:
    - The system should be designed to be easily testable.
    - Implement unit tests, integration tests, and end-to-end tests.
- Recoverability:
    - Implement data backups and recovery procedures to minimize data loss in case of failures.
    - Implement proper logging to track system activity and errors.

Example Implementation Notes:

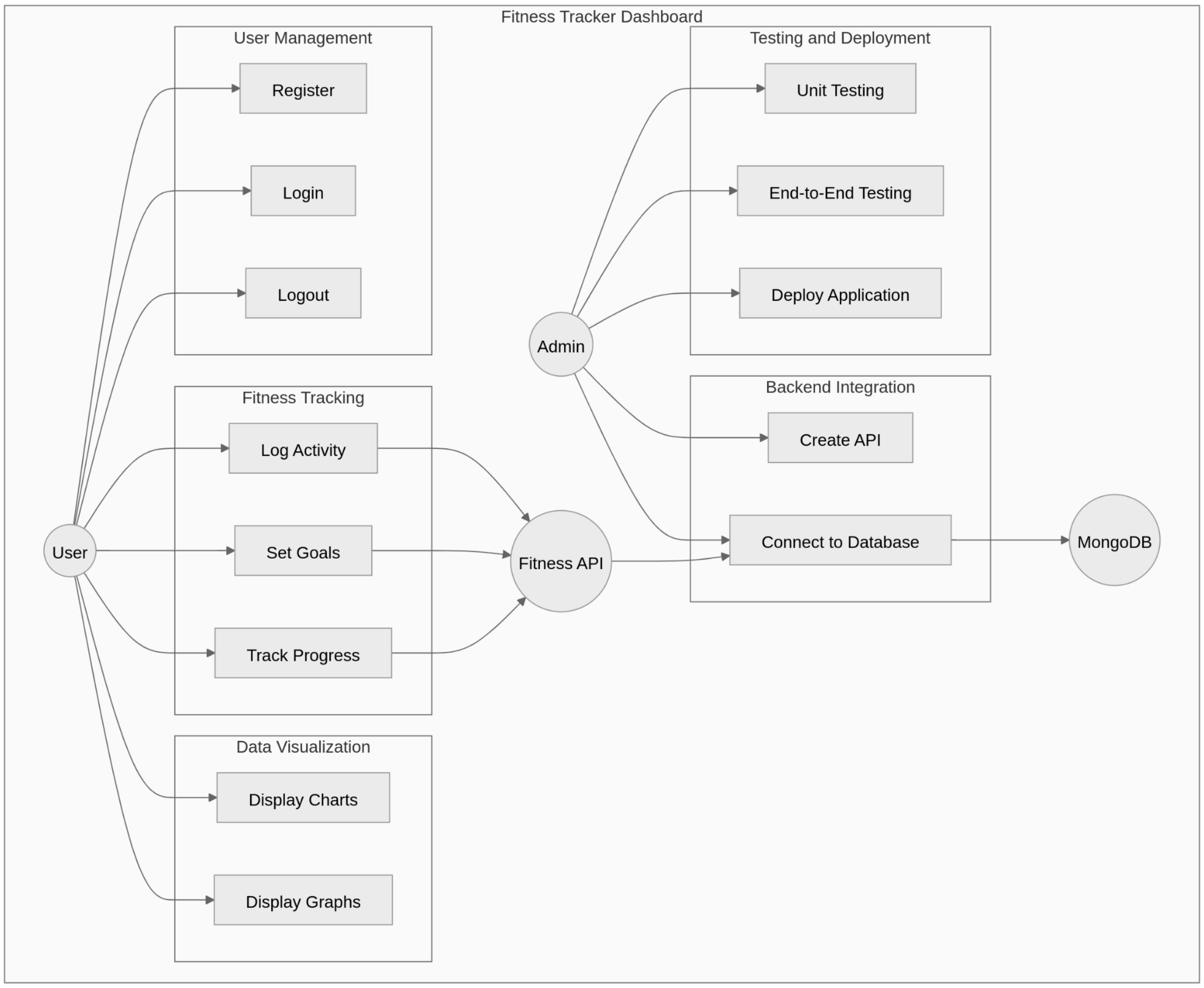
- Performance: Use code profiling tools to identify and optimize performance bottlenecks.
- Security: Use HTTPS, enforce strong password policies, and regularly update dependencies to patch security vulnerabilities.
- Usability: Conduct user testing to gather feedback and improve the UI.
- Reliability: Implement comprehensive testing, use a reliable hosting provider, and set up monitoring and alerting.

By addressing these non-functional requirements, you can build a robust, secure, and user-friendly Fitness Tracker Dashboard.

4. System Analysis & Design

1. Problem Statement & Objectives

○ Use Case Diagram



○ Functional & Non-Functional Requirements

■ Functional Requirements

1. User Management

- Users should be able to create an account, log in, and log out.
- Each user should have a personal profile where their data is securely stored.
- User data should be encrypted and stored securely.

2. Workout Logging

- Users should be able to log their daily workout activities (e.g., workout type, duration, steps taken, calories burned).
- Each user's workout history should be stored separately.
- Users should be able to edit or delete previously logged workout data.

3. Goal Setting & Progress Tracking

- Users should be able to set fitness goals (e.g., calories to burn per week, daily step count, workout hours per week).
- Users should be able to track their progress toward their goals with visualizations.

4. Database Integration

- All user data, workout logs, and goals should be stored in a MongoDB database.
- A Node.js API should handle data retrieval and storage.

5. Data Visualization

- Chart.js or D3.js should be used to generate graphs and charts displaying user progress over time.
- Charts should provide insights into calories burned, steps taken, workout duration, and goal comparisons.

6. Frontend & User Interface

- The application should have a clean and user-friendly interface built with React and CSS.
- The UI should be fully responsive, adapting to desktops, tablets, and mobile devices.

7. Authentication & Authorization

- JWT (JSON Web Token) or a similar method should be used to authenticate users and protect their data.
- Users should only have access to their own data and should not see other users' information.

8. Performance & Responsiveness

- The application should be optimized for fast performance when logging or displaying data.
- Data should be loaded and displayed dynamically without requiring a full page refresh.

9. Testing & Quality Assurance

- Unit tests should be conducted for core components, including data logging, API requests, and chart rendering.

- The application should be tested on different browsers and devices to ensure proper functionality.

10. Deployment & Documentation

❖ The application should be deployed on a cloud platform like Vercel or Netlify for the frontend and Render or Heroku for the backend.

❖ Comprehensive documentation should be provided, including:
  - ➢ How to use the application
  - ➢ API details (endpoints, input/output data)
  - ➢ Setup and deployment instructions for developers

- **Nonfunctional Requirements**

1. Performance Requirements:

- Response Time:
  - The dashboard should load within 3 seconds for users with a standard internet connection.
  - API calls for data retrieval and submission should complete within 2 seconds.
  - Chart rendering should occur within 2 seconds of data availability.
- Scalability:

    - The system should be able to handle a growing number of users and data entries without significant performance degradation.
    - The database (MongoDB) and backend (Node.js) should be scalable to accommodate increased load.
- Throughput:
    - The system should be able to handle a minimum of 1000 concurrent users.
    - The API should be able to handle 100 requests per second.

2. Usability Requirements:

- User Interface (UI) Clarity:
    - The dashboard should have an intuitive and easy-to-navigate interface.
    - Data visualizations (charts and graphs) should be clear and understandable.
    - Input forms should be straightforward and provide clear feedback.
- Accessibility:
    - The application should adhere to basic web accessibility guidelines to ensure usability for users with disabilities.
    - The interface should be navigable using keyboard controls.
- Responsiveness:
    - The dashboard should be fully responsive and adapt to various screen sizes (desktops, tablets, and mobile devices).
    - The user interface should maintain a consistent layout and functionality across different browsers.

3. Reliability Requirements:

- Availability:
  - The system should be available 99% of the time (or a higher percentage depending on the criticality).
  - Downtime for maintenance should be scheduled and minimized.
- Data Integrity:
  - Data stored in the database (MongoDB) should be accurate and consistent.
  - Data validation should be implemented to prevent invalid data entries.
  - Ensure that user data is not lost during system failures.
- Error Handling:
  - The system should handle errors gracefully and provide informative error messages to the user.
  - API errors should be logged and monitored for troubleshooting.

4. Security Requirements:

- Authentication:
  - User authentication should be implemented to protect user data.
  - Passwords should be securely stored using hashing and salting.
- Authorization:
  - Users should only have access to their own fitness data.
  - Appropriate authorization mechanisms should be in place to prevent unauthorized access.
- Data Protection:

    - Sensitive user data should be protected from unauthorized access and modification.
    - Protect against common web vulnerabilities, such as cross-site scripting (XSS) and SQL injection (if applicable).

- HTTPS:
    - The application should use HTTPS to encrypt data transmitted between the client and server.

5. Maintainability Requirements:

- Code Quality:
    - The codebase should be well-structured, documented, and easy to maintain.
    - Coding standards and best practices should be followed.
- Testability:
    - The system should be designed to facilitate unit testing and end-to-end testing.
- Modularity:
    - The code should be broken into small reusable modules to improve maintainablity.
- Logging:
    - Implement robust logging to track application behavior, errors, and security events.

6. Portability Requirements:

- Browser Compatibility:
    - The application should be compatible with modern web browsers (Chrome, Firefox, Safari, Edge).
- Platform Independence:
    - The backend should be able to be hosted on multiple cloud platforms.

Important Notes:

- Quantify: Whenever possible, quantify your NFRs (e.g., response time in seconds, availability as a percentage).
- Prioritize: Not all NFRs are equally important. Prioritize them based on your project's goals and constraints.
- Test: NFRs should be tested and validated throughout the development process.
- Realistic Expectations: Make sure that the NFR's that you create are able to be reached.

- ## Software Architecture

  The **Fitness Tracker Dashboard** follows the **Model-View-Controller (MVC)** architecture to keep the system well-organized and scalable.

  1. Main Components

  1. **Model (Backend - Node.js & Database)**

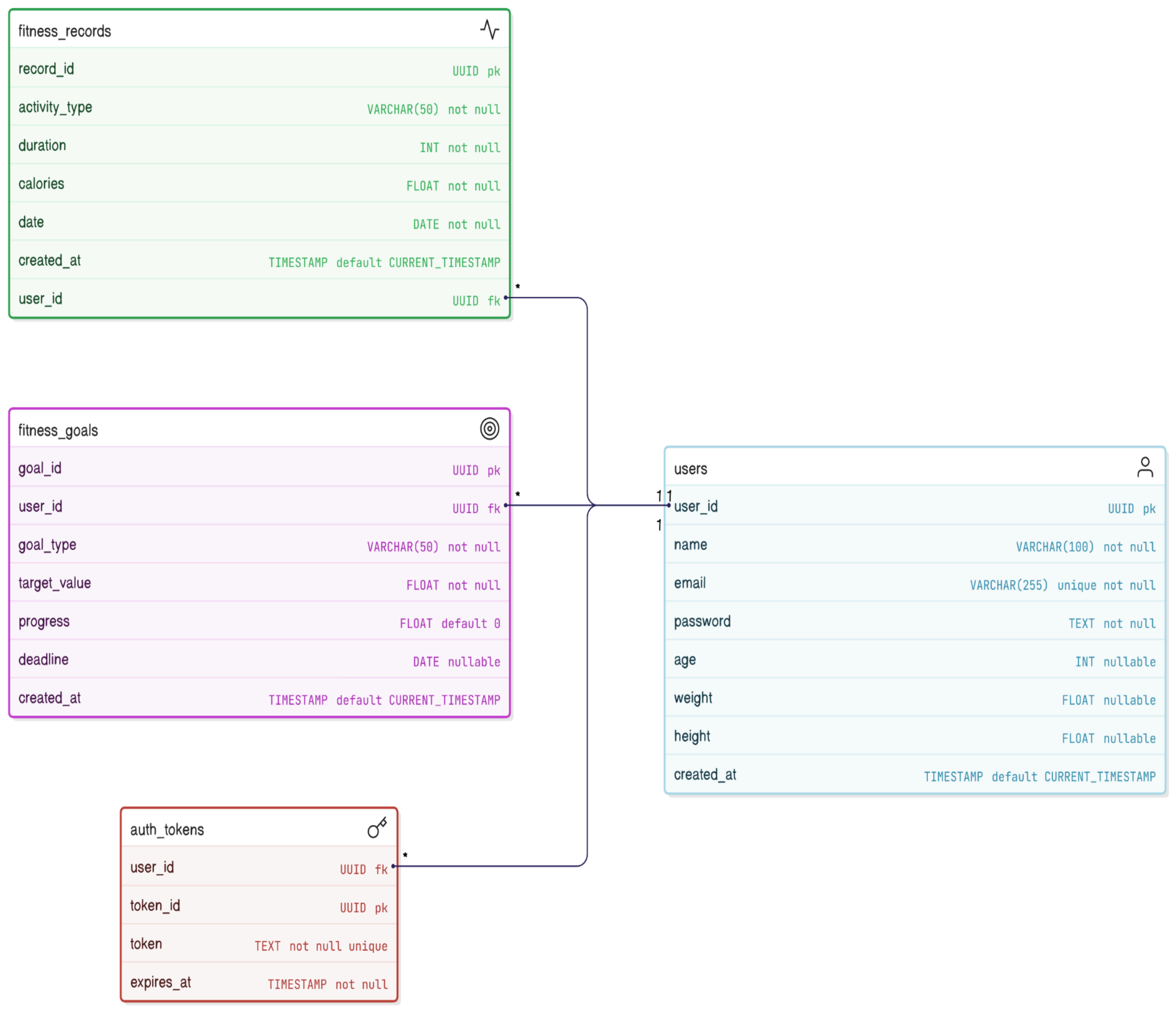
- Manages user data, workouts, and goals.
- Stores information in a **database (MongoDB)**.

2. **View (Frontend - React)**
   - Displays user interface (dashboard, charts, workout logs).
   - Fetches data from the backend via **APIs**.
3. **Controller (API & Routing - Express.js in Node.js)**
   - Handles user requests (e.g., logging a workout, setting goals).
   - Sends and retrieves data from the **Model** and responds to the **View**.

2. Database Design & Data Modeling

1- ER Diagram (Entity-Relationship Diagram)

Fitness Tracking System Data Model



2. Logical Schema (ERD - Entity-Relationship Diagram)

Entities & Relationships

1. **Users** (Stores user account details)
   - One user can have many fitness records.
   - One user can set multiple fitness goals.

    - One user can have authentication credentials.
2. **Fitness_Data** (Stores user activity logs)
    - Belongs to a user.
    - Contains workout details (type, duration, calories burned, etc.).
3. **Fitness_Goals** (Stores user-defined fitness goals)
    - Belongs to a user.
    - Tracks progress based on fitness records.
4. **Auth_Tokens** (Stores user authentication tokens for session management)
    - Linked to users for login sessions.

3. Physical Schema (Database Tables & Attributes)

Users Table

Column Name	Data Type	Constraints	Description
user_id	UUID (PK)	PRIMARY KEY	Unique identifier for the user
name	VARCHAR(100)	NOT NULL	User's full name
email	VARCHAR(255)	UNIQUE, NOT NULL	User's email address
password	TEXT	NOT NULL	Hashed password for authentication
age	INT	NULLABLE	User's age
weight	FLOAT	NULLABLE	User's weight (kg/lbs)
height	FLOAT	NULLABLE	User's height (cm/in)
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Account creation timestamp

Fitness_Records Table

Column Name	Data Type	Constraints	Description
record_id	UUID (PK)	PRIMARY KEY	Unique identifier for record

user_id	UUID (FK)	FOREIGN KEY -> Users(user_id)	User who logged the activity
activity_type	VARCHAR(50)	NOT NULL	Type of activity (e.g., running, cycling)
duration	INT	NOT NULL	Duration in minutes
calories	FLOAT	NOT NULL	Calories burned
date	DATE	NOT NULL	Date of workout
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp of record creation

Fitness_Goals Table

Column Name	Data Type	Constraints	Description
goal_id	UUID (PK)	PRIMARY KEY	Unique identifier for goal
user_id	UUID (FK)	FOREIGN KEY -> Users(user_id)	User setting the goal
goal_type	VARCHAR(50)	NOT NULL	Type of goal (e.g., weight loss, running distance)
target_value	FLOAT	NOT NULL	Target value (e.g., 5kg weight loss, 10km run)
progress	FLOAT	DEFAULT 0	Progress towards the goal
deadline	DATE	NULLABLE	Deadline to achieve the goal
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp of goal creation

Auth_Tokens Table

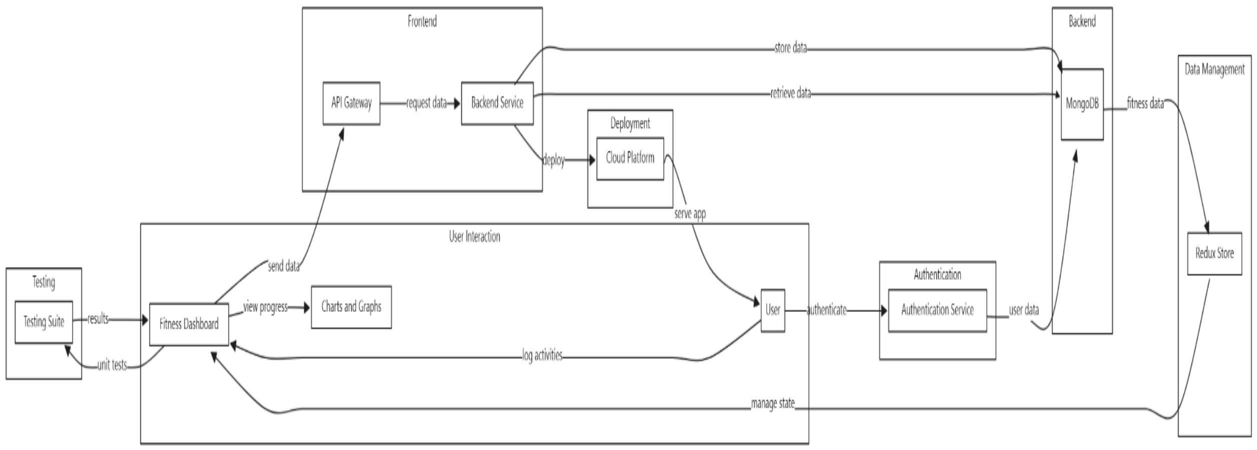
Column Name	Data Type	Constraints	Description
token_id	UUID (PK)	PRIMARY KEY	Unique identifier for token
user_id	UUID (FK)	FOREIGN KEY -> Users(user_id)	User who owns the token
token	TEXT	NOT NULL, UNIQUE	Authentication token
expires_at	TIMESTAMP	NOT NULL	Expiration time for token

3. Normalization Considerations

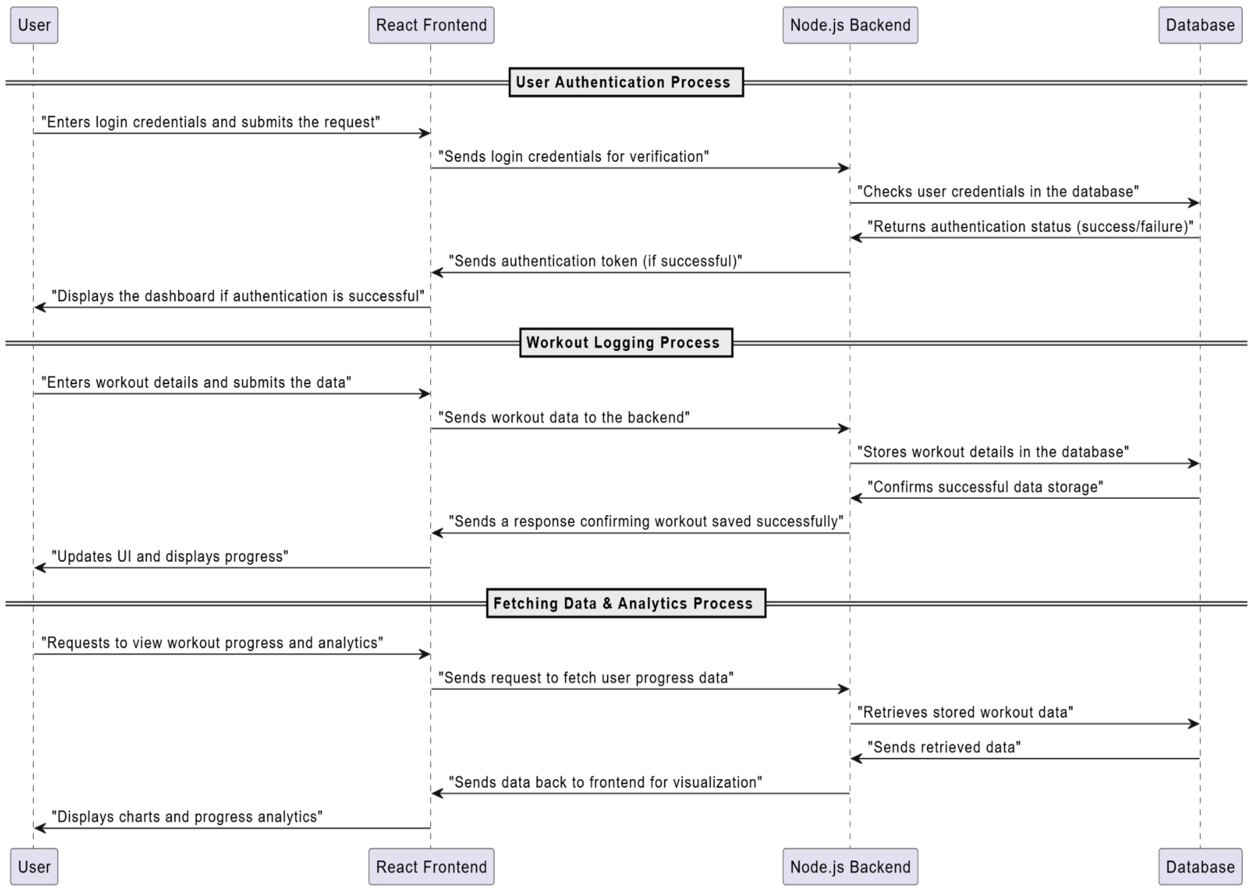
- **1NF (First Normal Form)**: Each table has atomic values; no duplicate columns.
- **2NF (Second Normal Form)**: No partial dependencies; all non-key attributes depend on the primary key.
- **3NF (Third Normal Form)**: No transitive dependencies; attributes only depend on their table's primary key.

3. Data Flow & System Behavior

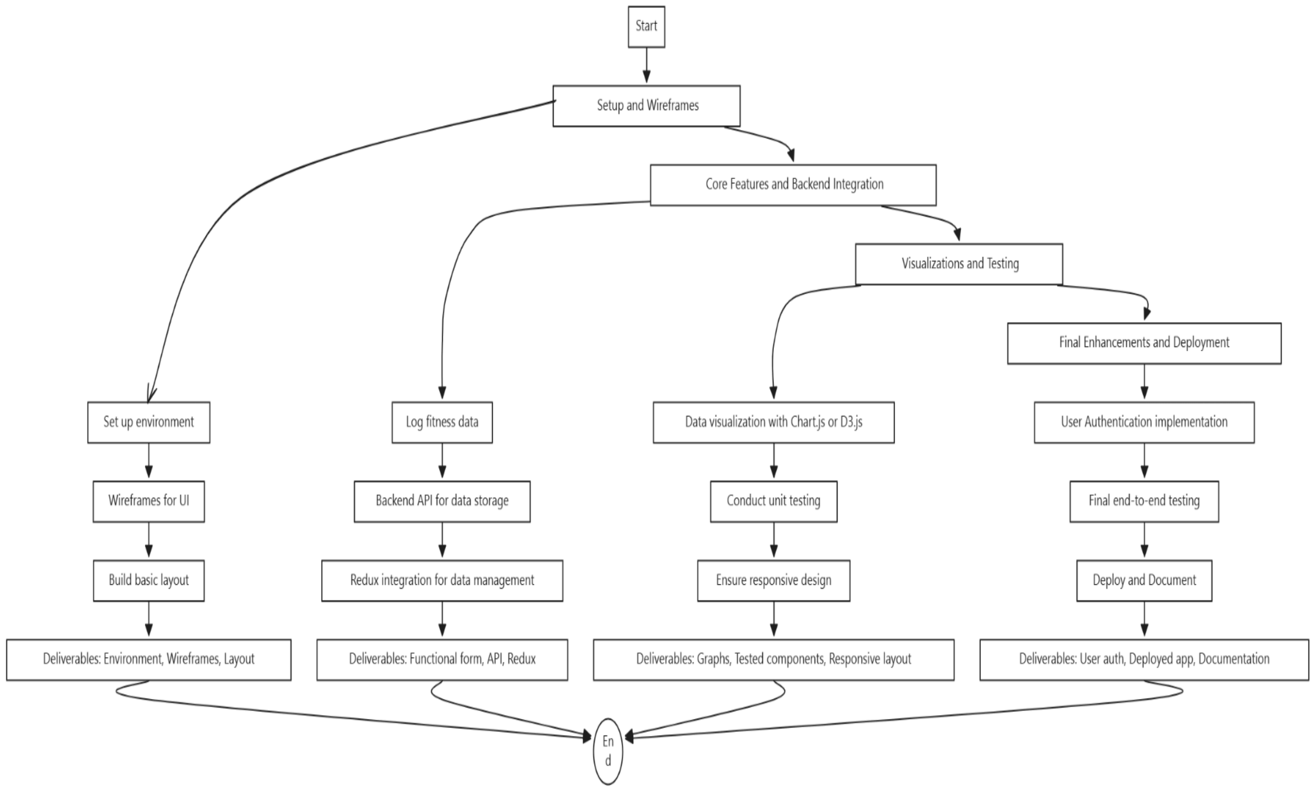
- **DFD (Data Flow Diagram)**



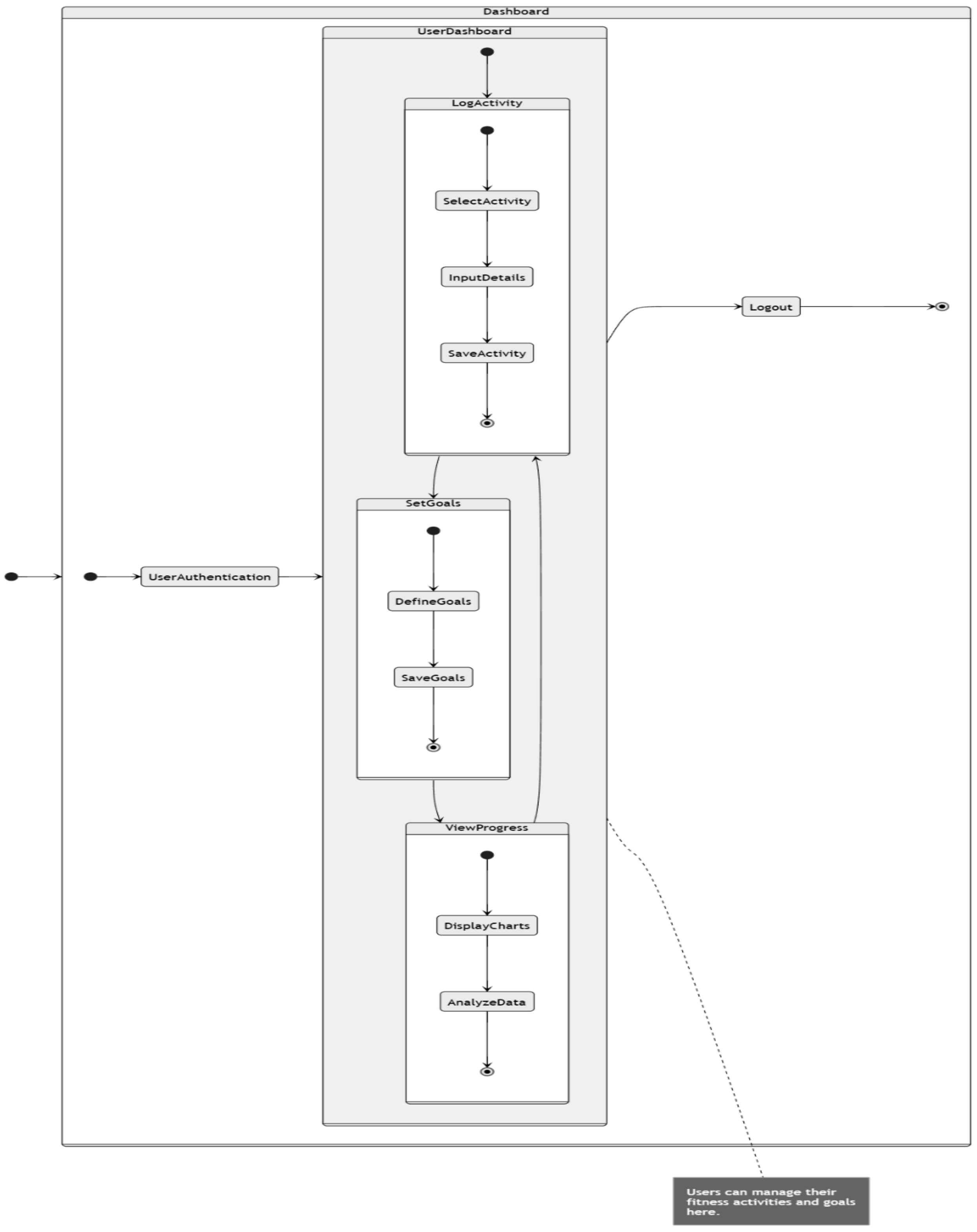
- **Sequence Diagrams**


User
React Frontend
Node.js Backend
Database
User Authentication Process
"Enters login credentials and submits the request"
"Sends login credentials for verification"
"Checks user credentials in the database"
"Returns authentication status (success/failure)"
"Sends authentication token (if successful)"
"Displays the dashboard if authentication is successful"
Workout Logging Process
"Enters workout details and submits the data"
"Sends workout data to the backend"
"Stores workout details in the database"
"Confirms successful data storage"
"Sends a response confirming workout saved successfully"
"Updates UI and displays progress"
Fetching Data & Analytics Process
"Requests to view workout progress and analytics"
"Sends request to fetch user progress data"
"Retrieves stored workout data"
"Sends retrieved data"
"Sends data back to frontend for visualization"
"Displays charts and progress analytics"
User
React Frontend
Node.js Backend
Database

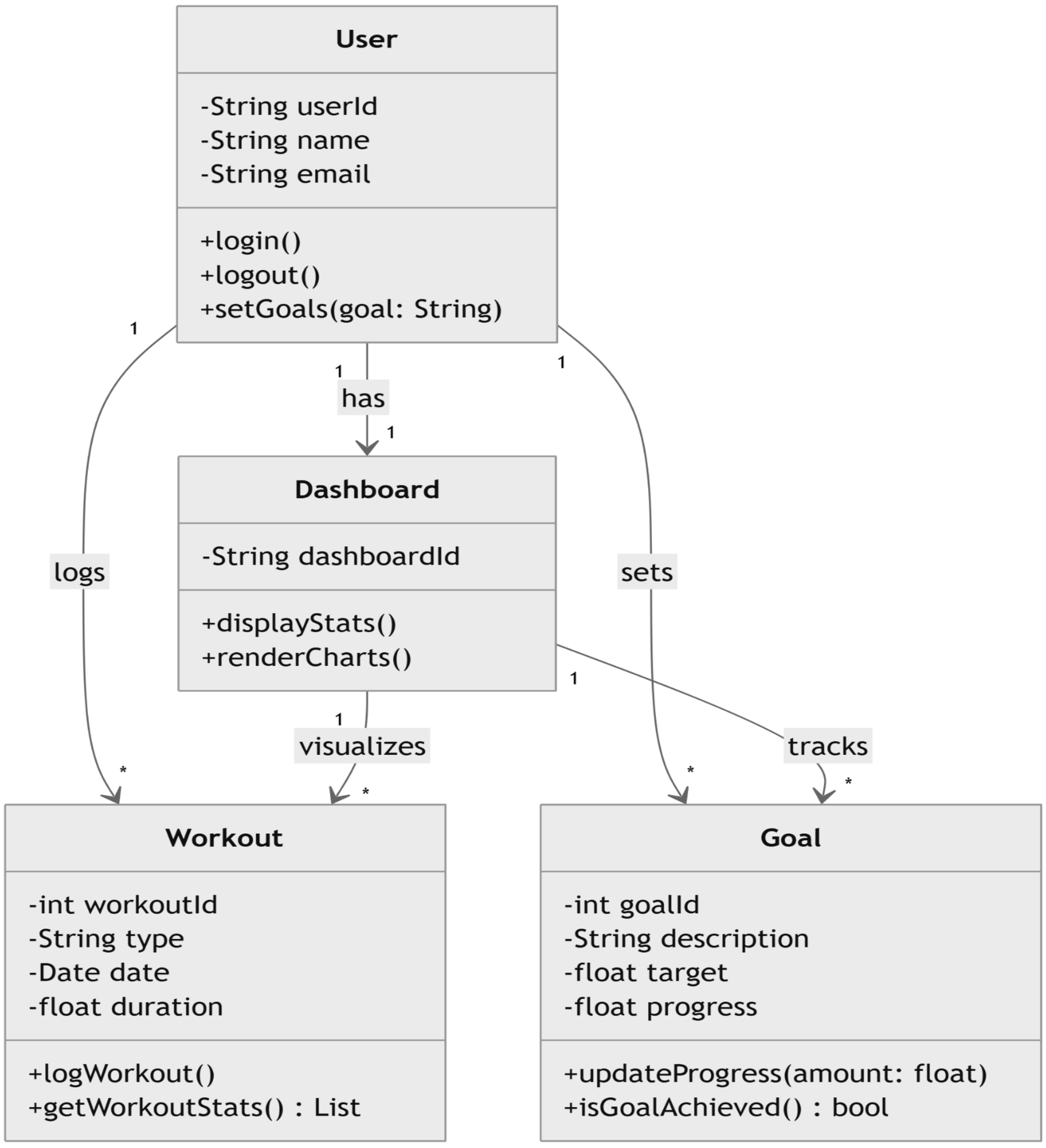
- **Activity Diagram**


Start
Setup and Wireframes
Core Features and Backend Integration
Visualizations and Testing
Final Enhancements and Deployment
Set up environment
Wireframes for UI
Build basic layout
Deliverables: Environment, Wireframes, Layout
Log fitness data
Backend API for data storage
Redux integration for data management
Deliverables: Functional form, API, Redux
Data visualization with Chart.js or D3.js
Conduct unit testing
Ensure responsive design
Deliverables: Graphs, Tested components, Responsive layout
User Authentication implementation
Final end-to-end testing
Deploy and Document
Deliverables: User auth, Deployed app, Documentation
End

- **State Diagram**

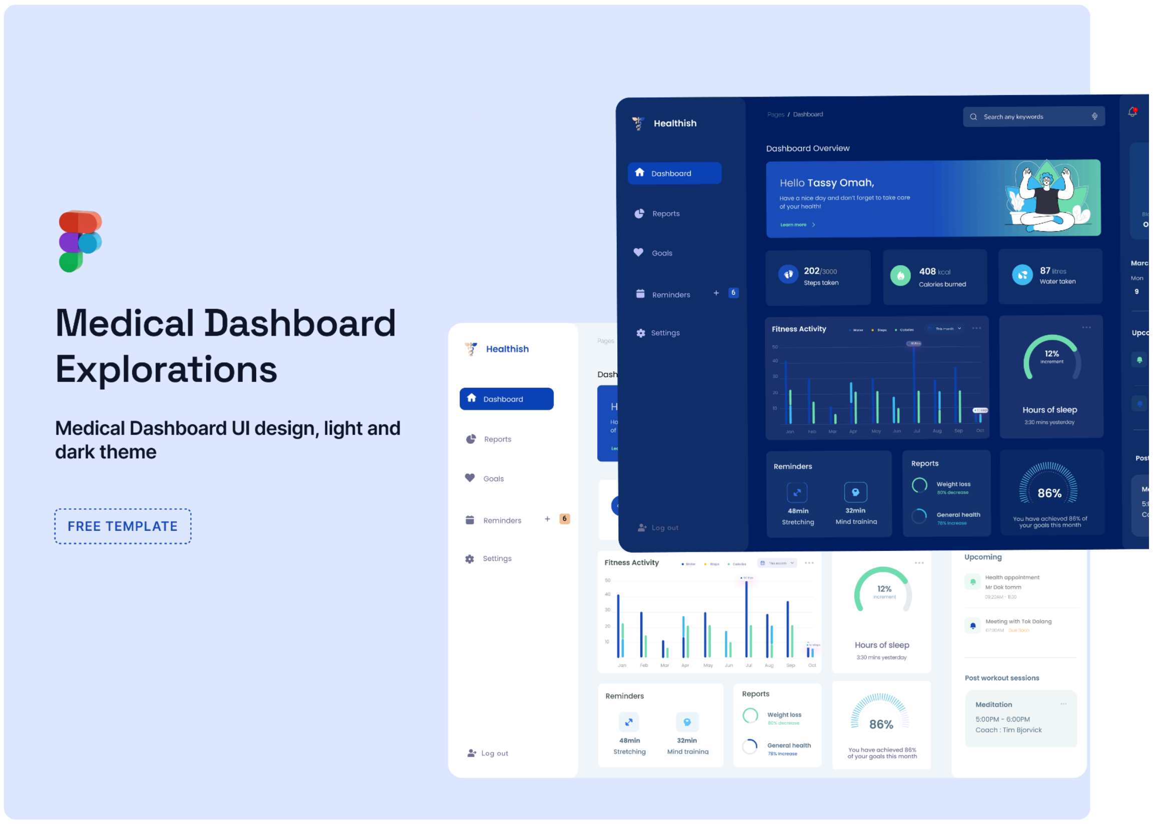


- **Class Diagram**



4. UI/UX Design & Prototyping

1. **Wireframes & Mockups**
   Figma Link



2. **UI/UX Guidelines**
   a. Design Principles
      ✅ Consistency:
      Maintain a uniform design across all pages (workout logging, goal setting, progress tracking).
      Use a fixed color scheme, typography, and spacing to ensure visual harmony.

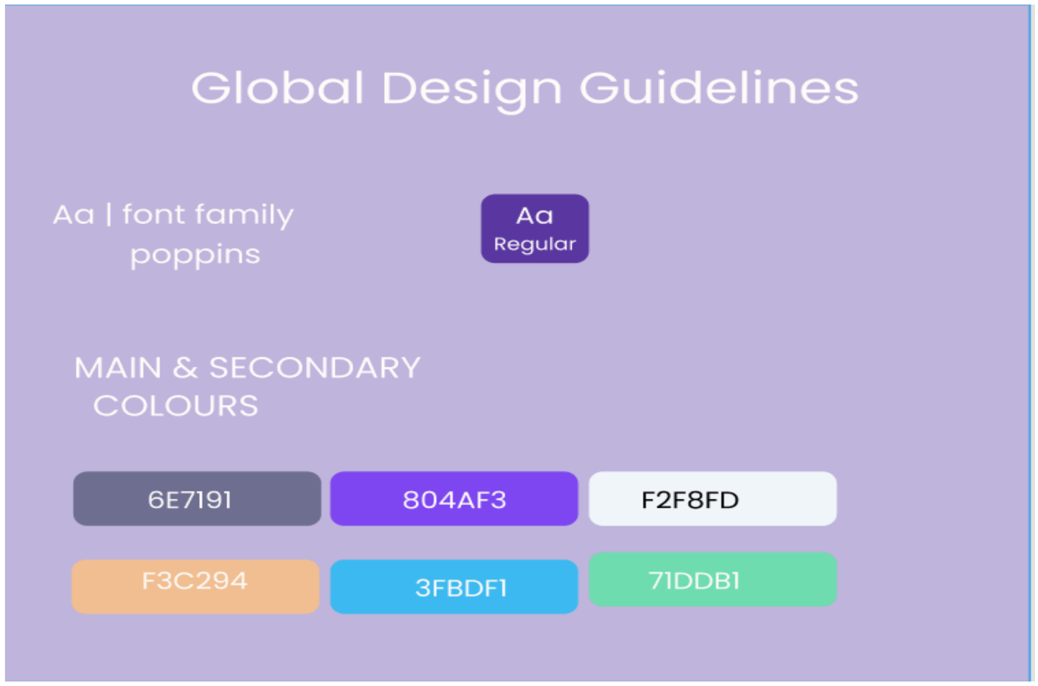
      ✅ Simplicity:
      Avoid clutter; keep only essential elements visible.
      Use clear navigation (dashboard, workouts, goals, progress, settings).

      ✅ Hierarchy:

Prioritize important data (e.g., today's progress, active goals) at the top.
Use size and color to differentiate headings, subheadings, and body text.

✅ Feedback & Interactivity:
Provide real-time updates (e.g., when a workout is logged, show a confirmation message).
Use subtle animations for button clicks, hover effects, and progress bars.

✅ User-Centered Design:
Optimize for mobile & desktop (responsive design).
Keep workout logs quick & intuitive (e.g., use dropdowns, sliders, or quick-add buttons).

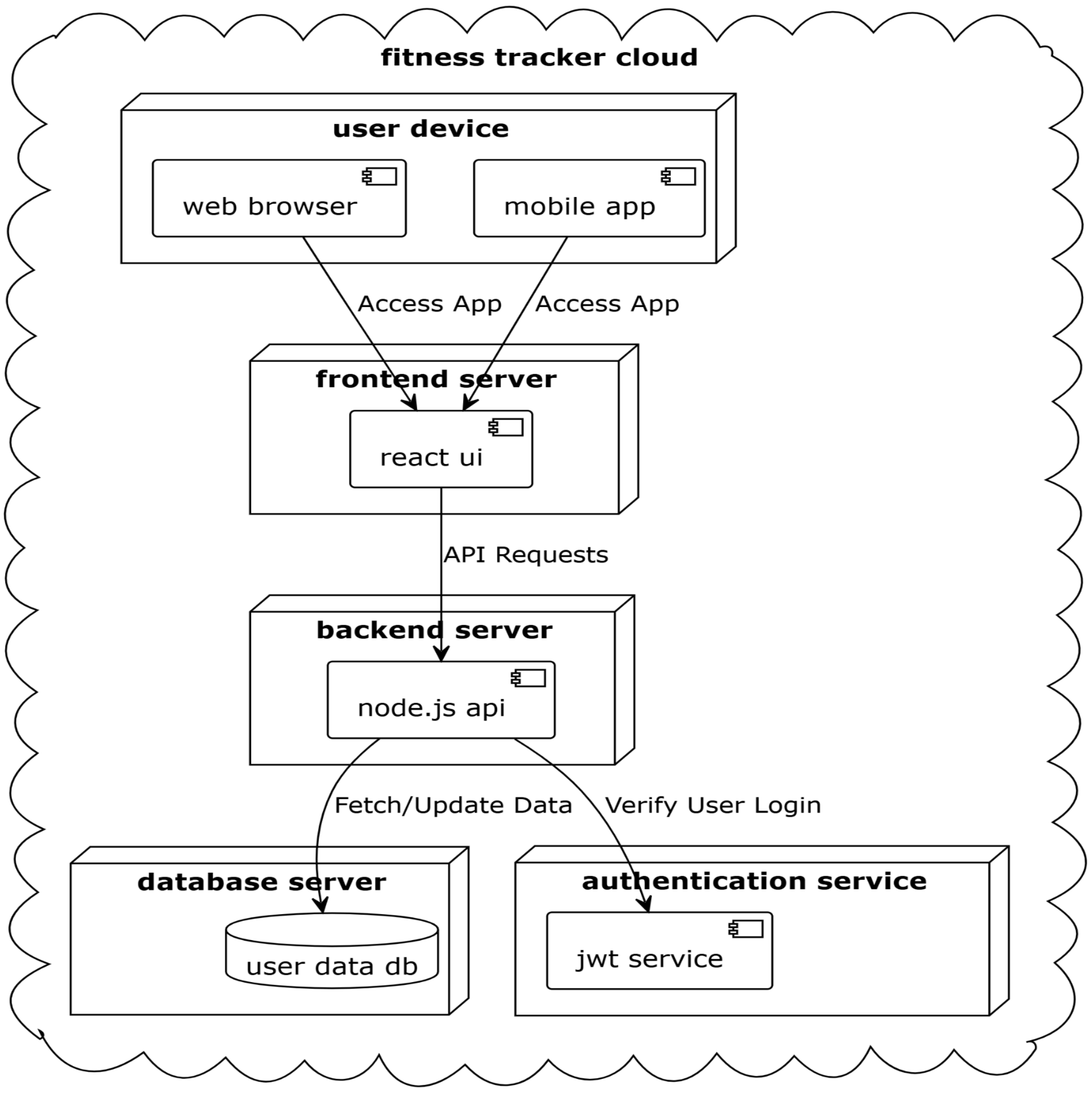


5. System Deployment & Integration

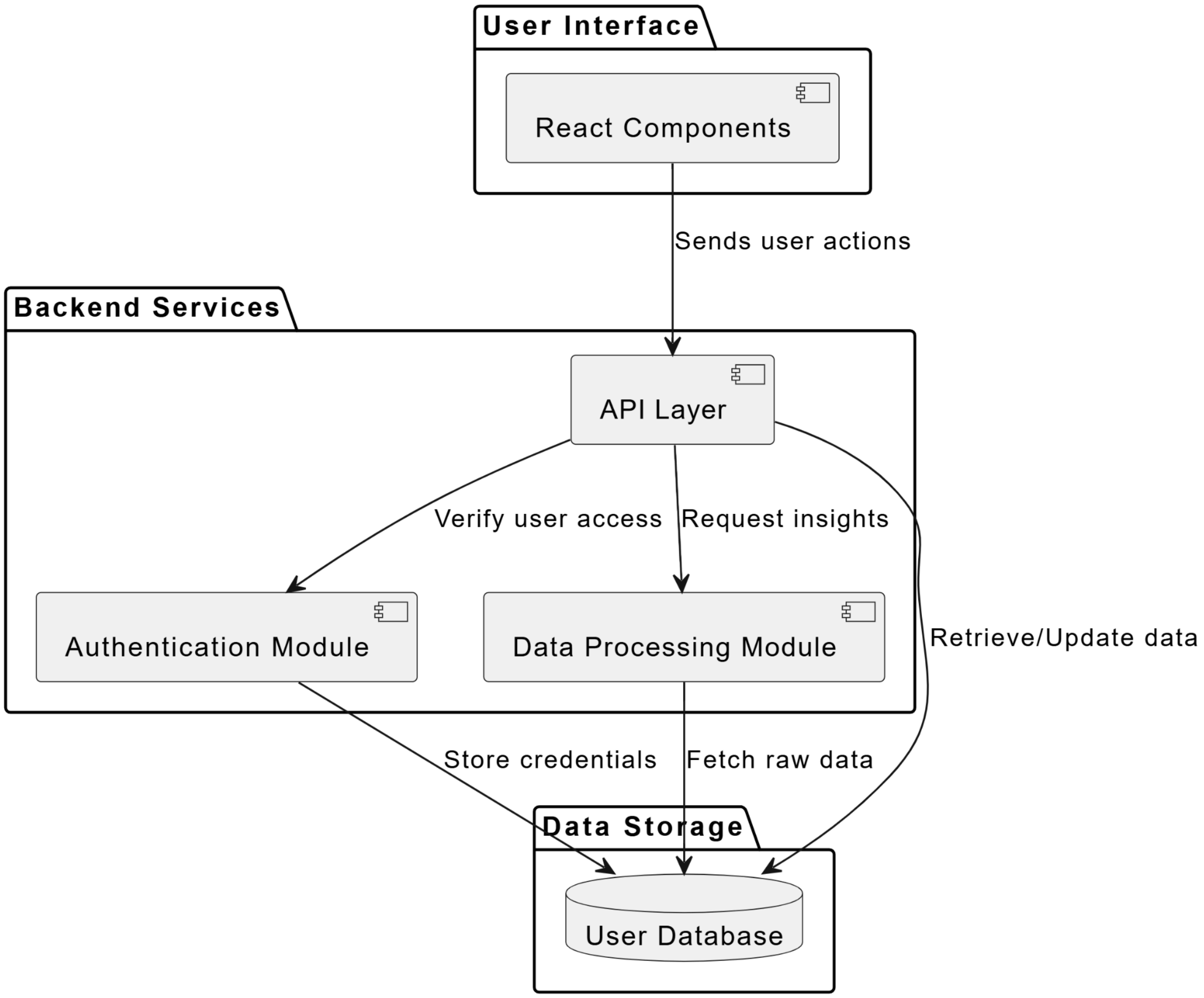
1. Technology Stack

a. **Frontend:** React, HTML, CSS, JavaScript (User interface).
b. **Backend:** Node.js, Express (Handles requests and processes data).
c. **Database:** MongoDB or MySQL (Stores user and fitness data).

2. Deployment Diagram



3. Component Diagram



6. Additional Deliverables

Fitness Tracker Dashboard API Documentation

Base URL

https://api.fitness-tracker.com/v1

Authentication

User Login

POST /auth/login

- **Description:** Authenticates the user and returns an access token.

- **Request Body:**

```
{
  "email": "user@example.com",
  "password": "securepassword"
}
```

- **Response:**

```
{
  "token": "your-access-token",
  "expires_in": 3600
}
```

User Registration

POST /auth/register

- **Description:** Creates a new user account.
- **Request Body:**

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "securepassword",
  "age": 30,
  "weight": 70,
  "height": 175
}
```

- **Response:**

```
{
  "message": "User registered successfully",
  "user_id": "12345"
}
```

}

User Profile

Get User Profile

GET /users/profile

- **Description:** Retrieves the user's profile.
- **Headers:**
  - Authorization: Bearer <token>
- **Response:**

```
{
  "user_id": "12345",
  "name": "John Doe",
  "email": "john@example.com",
  "age": 30,
  "weight": 70,
  "height": 175
}
```

Update User Profile

PUT /users/profile

- **Description:** Updates the user's profile.
- **Headers:**
  - Authorization: Bearer <token>
- **Request Body:**

```
{
  "age": 31,
  "weight": 72,
  "height": 176
}
```

- **Response:**

```
{
  "message": "Profile updated successfully"
}
```

Fitness Data

Log Workout Activity

POST /fitness/log

- **Description:** Logs a new workout session.
- **Headers:**
  - Authorization: Bearer <token>
- **Request Body:**

```
{
  "activity_type": "Running",
  "duration": 30,
  "calories": 250,
  "date": "2024-02-20"
}
```

- **Response:**

```
{
  "message": "Workout logged successfully",
  "record_id": "67890"
}
```

Get Workout History

GET /fitness/history

- **Description:** Fetches the user's workout history.
- **Headers:**
  - Authorization: Bearer <token>
- **Response:**

```
[
  {
    "record_id": "67890",
    "activity_type": "Running",
    "duration": 30,
    "calories": 250,
    "date": "2024-02-20"
  }
]
```

Delete a Workout Record

DELETE /fitness/delete/{record_id}

- **Description:** Deletes a specific workout record.
- **Headers:**
    - Authorization: Bearer <token>
- **Response:**

```
{
  "message": "Workout record deleted successfully"
}
```

Fitness Goals

Set a Fitness Goal

POST /goals

- **Description:** Sets a new fitness goal.
- **Headers:**
    - Authorization: Bearer <token>
- **Request Body:**

```
{
  "goal_type": "Weight Loss",
  "target_value": 5,
  "deadline": "2024-06-01"
}
```

- **Response:**

```
{
  "message": "Goal set successfully",
  "goal_id": "54321"
}
```

Get Fitness Goals

GET /goals

- **Description:** Retrieves the user's fitness goals.
- **Headers:**
  - Authorization: Bearer <token>
- **Response:**

```
[
  {
    "goal_id": "54321",
    "goal_type": "Weight Loss",
    "target_value": 5,
    "progress": 2,
    "deadline": "2024-06-01"
  }
]
```

Delete a Fitness Goal

DELETE /goals/{goal_id}

- **Description:** Deletes a fitness goal.
- **Headers:**
  - Authorization: Bearer <token>
- **Response:**

```
{
  "message": "Goal deleted successfully"
}
```

Error Responses

- **401 Unauthorized:** Invalid or missing authentication token.
- **400 Bad Request:** Missing required fields or invalid data format.
- **404 Not Found:** Resource not found.
- **500 Internal Server Error:** Unexpected server issue.

Notes:

- All API calls require **Bearer Token Authentication**.
- Date format: YYYY-MM-DD.
- Response times are optimized to be **<200ms** under normal load.

API Rate Limits

Endpoint	Rate Limit
/auth/login	5 requests per minute
/fitness/log	20 requests per minute
/goals	10 requests per minute

Conclusion

This API provides secure endpoints to manage **user authentication, fitness tracking, and goal setting** in the Fitness Tracker Dashboard. For integration, use the provided endpoints with proper **authentication headers**.

Testing & Validation Plan

1. Unit Testing

- **Purpose:** Ensure individual components and functions work correctly.
- **Tools:** Jest (for React frontend), Mocha/Chai (for Node.js backend).
- **Key Test Cases:**

Frontend (React)

Component	Test Case	Expected Outcome
SignUpForm	Valid inputs	User registered successfully
SignUpForm	Missing required fields	Error message displayed
SignInForm	Correct credentials	Redirects to dashboard

SignInForm	Incorrect credentials	Shows "Invalid login" message
Dashboard Charts	Data loaded	Charts render correctly
API Calls	Fetch fitness data	Retrieves valid response

Backend (Node.js, MongoDB)

API Endpoint	**Test Case**	**Expected Outcome**
POST /auth/register	Unique email, valid password	201 - User created
POST /auth/login	Correct credentials	200 - Returns token
POST /fitness/log	Valid data	201 - Record created
GET /fitness/history	Authorized user	200 - Returns records
DELETE /fitness/delete/:id	Record exists	200 - Deleted successfully

2. Integration Testing

- **Purpose:** Ensure different modules (frontend-backend, API-DB) work together correctly.
- **Tools:** Postman (API testing), Cypress (end-to-end UI testing).

Integration Test Scenarios

Scenario	**Steps**	**Expected Result**
User logs in	Enter valid credentials	Redirects to dashboard, fetches user data
User logs fitness data	Fill workout form, submit	Data appears in history

User updates profile	Modify weight, save	Profile updates successfully
User sets a goal	Enter goal details, submit	Goal saved and displayed

3. User Acceptance Testing (UAT)

- **Purpose:** Validate the app meets real user needs before deployment.
- **Participants:** Project stakeholders, end-users.
- **Testing Environment:** Staging deployment on Netlify & backend host (e.g., Render).

UAT Test Cases

Feature	Scenario	Expected Outcome
Sign Up	New user registers	Account created successfully
Login	Existing user logs in	Dashboard loads correctly
Logging Workouts	User logs running session	Workout saved, updated on dashboard
Data Visualization	User checks progress chart	Charts display correct stats
Goal Tracking	User adds a goal	Goal appears in goal tracker

Final Validation & Deployment Checklist

✅ All unit and integration tests pass.
✅ No critical bugs in UAT.

✅ Performance tested (API response <200ms).
✅ Deployed to **Netlify (Frontend) & Render/Vercel (Backend)**.
✅ Documentation updated for API and testing logs.

Deployment Strategy for Fitness Tracker Dashboard 🚀

This deployment plan ensures a **scalable, secure, and automated** deployment of the **Fitness Tracker Dashboard** frontend and backend.

---

1. Hosting Environment

Component	**Hosting Platform**	**Tech Used**
Frontend (React)	Netlify / Vercel	React, HTML, CSS, JavaScript
Backend (Node.js + Express)	Render / Vercel / Railway	Node.js, Express.js
Database (MongoDB)	MongoDB Atlas / DigitalOcean / AWS	NoSQL, Cloud-hosted
Storage (if needed for images/files)	AWS S3 / Cloudinary	Cloud-based storage

2. Deployment Pipelines (CI/CD)

A. Frontend (React) – Netlify/Vercel

1. **Git Integration:** Push code to GitHub (main branch).
2. **Automatic Deployment:** Netlify/Vercel builds and deploys React app.
3. **Environment Variables:**
    - REACT_APP_API_BASE_URL = https://api.fitness-tracker.com/v1
4. **Preview Deployments:** Every pull request gets a unique preview URL.
5. **Production Deployment:** Merges to main trigger a live deployment.

B. Backend (Node.js) – Render/Vercel/Railway

1. **GitHub Actions / Render Auto Deploy:**
    - Push to main triggers automated deployment.

2. **Containerization (Optional, for Scaling):**
   - Use **Docker** for a containerized backend.

C. Database – MongoDB Atlas

1. **Cloud Database:** No manual setup, scalable automatically.
2. **Backup Strategy:** MongoDB Atlas provides automatic backups.
3. **Access Control:**
   - Whitelist server IPs for security.
   - Use **MongoDB authentication** instead of public access.

---

3. Scaling Considerations

Component	Scaling Strategy
Frontend	Netlify/Vercel auto-scales with global CDN caching
Backend	Render/Vercel scales automatically (horizontal scaling)
Database	MongoDB Atlas auto-scales storage and connections
Load Balancing	Vercel/Render handle requests efficiently
Rate Limiting	Protects API from abuse (e.g., 100 requests per minute per user)

4. Monitoring & Logging

Tool	Purpose
Netlify Analytics	Tracks frontend traffic
Render Logs / Vercel Logs	Monitors backend performance
MongoDB Atlas Metrics	Checks database queries and load
Sentry / LogRocket	Error tracking for React app
Postman API Monitoring	Ensures API uptime

5. Deployment Workflow Summary

1. **Developer pushes code → GitHub**
2. **CI/CD Pipeline runs:**
    - **Frontend:** Netlify/Vercel builds & deploys
    - **Backend:** Render/Vercel deploys API
3. **Automated Tests Run (Jest/Mocha)**
4. **Deployment to Staging (Test Environment)**
5. **Final Approval & Deployment to Production**

Final Checklist Before Go-Live

✅ **All unit & integration tests pass**
✅ **Performance optimized (API response <200ms, frontend loads <2s)**
✅ **Uptime Monitoring setup**
✅ **Database security configured (no public access)**
✅ **CDN caching enabled for static assets**