FITNESSO- TECHNICAL DESIGN

TABLE OF CONTENTS

---

1. Introduction and System Overview
2. Architecture
    - Components Interaction
    - High-level Architecture
3. Detailed Design
    - Frontend Design
        - Components
        - State Management
        - UI/UX
    - Backend Design
        - API Endpoints
        - Database Schema
        - Technologies used
        - Project Structure
4. Middleware
5. Backend Design Diagram
6. Security Considerations
7. API Design
8. Data Flow and Error Handling
9. Deployment
10. Future Enhancements
11. Conclusion

INTRODUCTION AND SYSTEM OVERVIEW

Introduction

The Fitnesso website is a modern web application built with the MERN stack (MongoDB, Express, React, Node.js). It facilitates seamless interaction between yoga tutors and students by offering an online platform for appointment bookings, tutor registrations, and payment processing. Designed with a user-friendly interface, the system supports both tutors and users, enabling

- tutors to manage their profiles, schedules, and sessions.
- Users to find tutors, book appointments, cancel appointments and make payments.

This fully responsive application ensures smooth performance on desktops, tablets, and mobile devices, making it accessible to a broad audience.

System Overview

Fitnesso follows a client-server architecture consisting of:

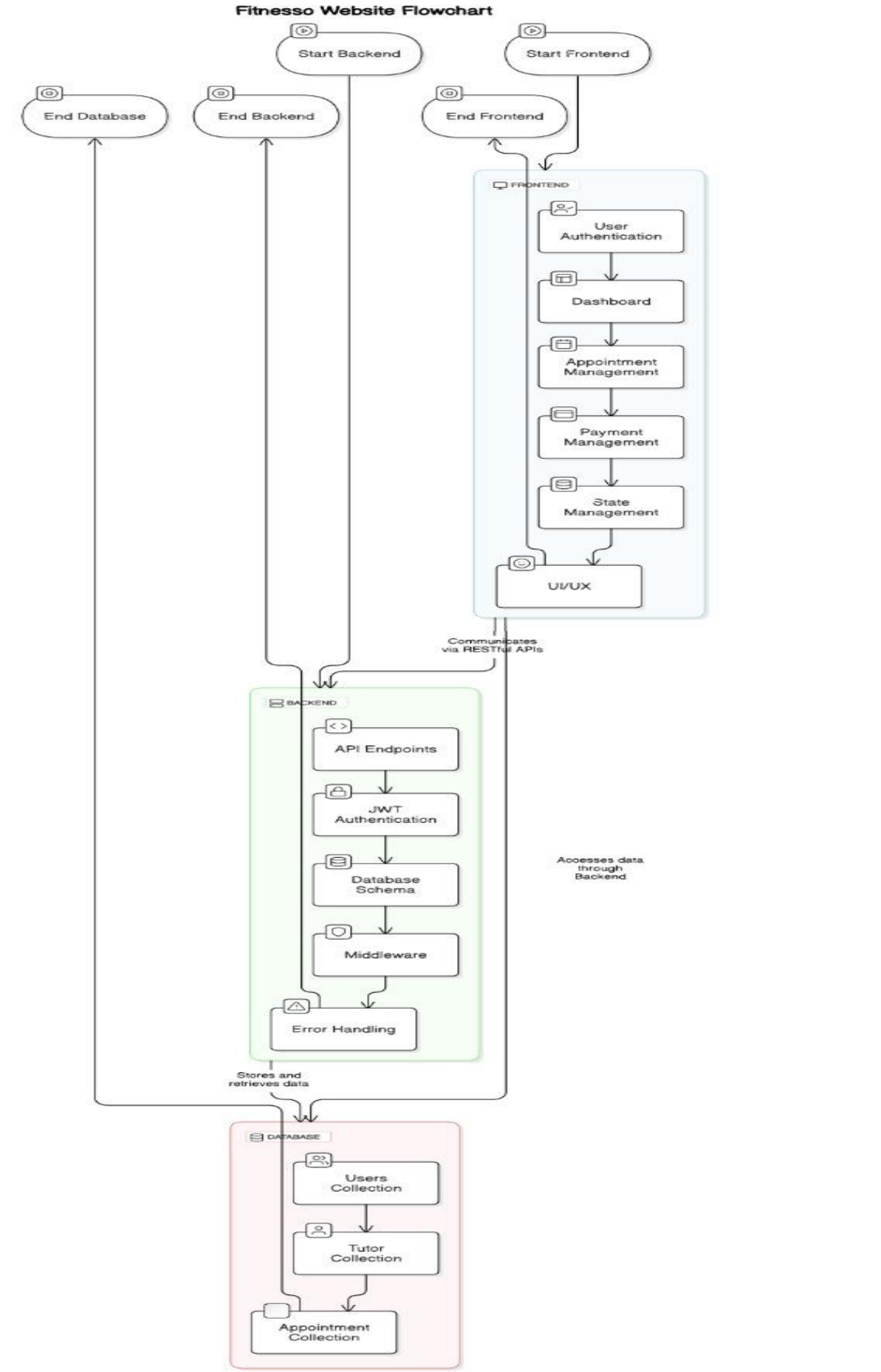
- Frontend (React.js): A dynamic, responsive interface for managing profiles, bookings, and payments. Communicates with the backend through RESTful APIs.
- Backend (Node.js with Express): Handles API requests, business logic, and secure user authentication using Json Web Token (JWT).
- Database (MongoDB): Stores user profiles, appointments, and payment records for scalable and efficient data management.

ARCHITECTURE

Components Interaction and High-level Architecture:

- **Frontend**: React.js (Client-side)
- **Backend**: Node.js with Express (Server-side)
- **Database**: MongoDB (For persistent storage)
- **Authentication**: JWT (JSON Web Tokens for secure login),
- **Authorization**: Access control, roles and permissions.

ARCHITECTURE



DETAILED DESIGN

Frontend Design(React.js)

- **Components**:
    1. Authentication:
        - Login: to input username/password. Successful login provides a JWT token.
        - Register: Form to create a new account for students or tutors.

    2. Dashboard:
        - User dashboard: displays a list of available fitness tutors for user.
        - Tutor dashboard: shows tutor's schedule, profile management tools, and session details for tutors.
        - Admin dashboard: shows the list of all appointments of all tutors and manage permissions to create, read, update and delete(CRUD) operations.

    3. Appointment Management:
        - Booking: enables user to book sessions with available tutors.
        - Schedule Viewer: Allows tutors to view and manage their schedules.

    4. Payment Management:

        - Checkout: handles payment processing using integrated payment gateways.

    5. State Management:
        - Context API : to provides a straightforward way to share state across the application.

- **UI/UX:**

    - Toastify: provides smooth, non-intrusive notifications to deliver real-time feedback to users. It enhances the user experience by offering clear, subtle alerts for actions such as errors, success messages, or confirmations.

Backend Design(API, Database Schema and Middlewares)

- **API Endpoints:**

  a) User:
  - `POST /user/login`: Login endpoint to authenticate users.
  - `POST /login`: Register endpoint for new users.
  - `POST /api/register`: registers a new user (user or tutor).
  - `GET/api/user`: retrieves list of currently logged-in user.

  b) Admin:
  - `POST /api/add-tutor`: to add a tutor in database.
  - `GET /api/login`: login for admin.
  - `DELETE /tutor`: Delete a tutor from database.

- **Middlewares**

Authentication Middleware(authAdmin.js)

- Validates JWT tokens sent in the Authorization header.
- Protects routes that require authentication.

Error handling middlewares catches and handles errors in API request

Database Schema:

- **Users**: Store user profile information.
- **Tutor**: Store tutor details and information.

Technologies Used:

- **Node.js:** JavaScript runtime environment.
- **Express.js:** Web application framework for building APIs.
- **MongoDB:** NoSQL database for data storage.

Backend Design(API, Database Schema and Middlewares)

- **Mongoose:** Object Data Modeling(ODM) library for MongoDB.
- **JSON Web Token(JWT):** token-based authentication and authorization.
- **bcrypt:** Library for hashing passwords.
- **toastify**: displays non-intrusive, real-time notifications.
- **Nodemon**: automatically restarts server during development
- **Browser Router**: enables routing which allows single-page experience without requiring full page reloads.

Project Structure:

- **index.js:** Entry point of the server application.
- **middleware/:** Contains middleware functions, including authentication.
- **models/:** Mongoose schemas for user and tutor models.
- **routes/:** defines API endpoints related to authentication, user management, admin panel and tutor functionality.
- **connect.db.js:** Controller to connect with database.

DATABASE SCHEMA EXPLANATION

- User Model (UserModel.js):

  Fields:

  - name(String, required): User's full name
  - email(String, required, unique): User's email address
  - password(String, required): hashed password
  - gender(String): User’s gender
  - dob(String): stores user’s date of birth
  - phone(number):stores contact number.

  Notes

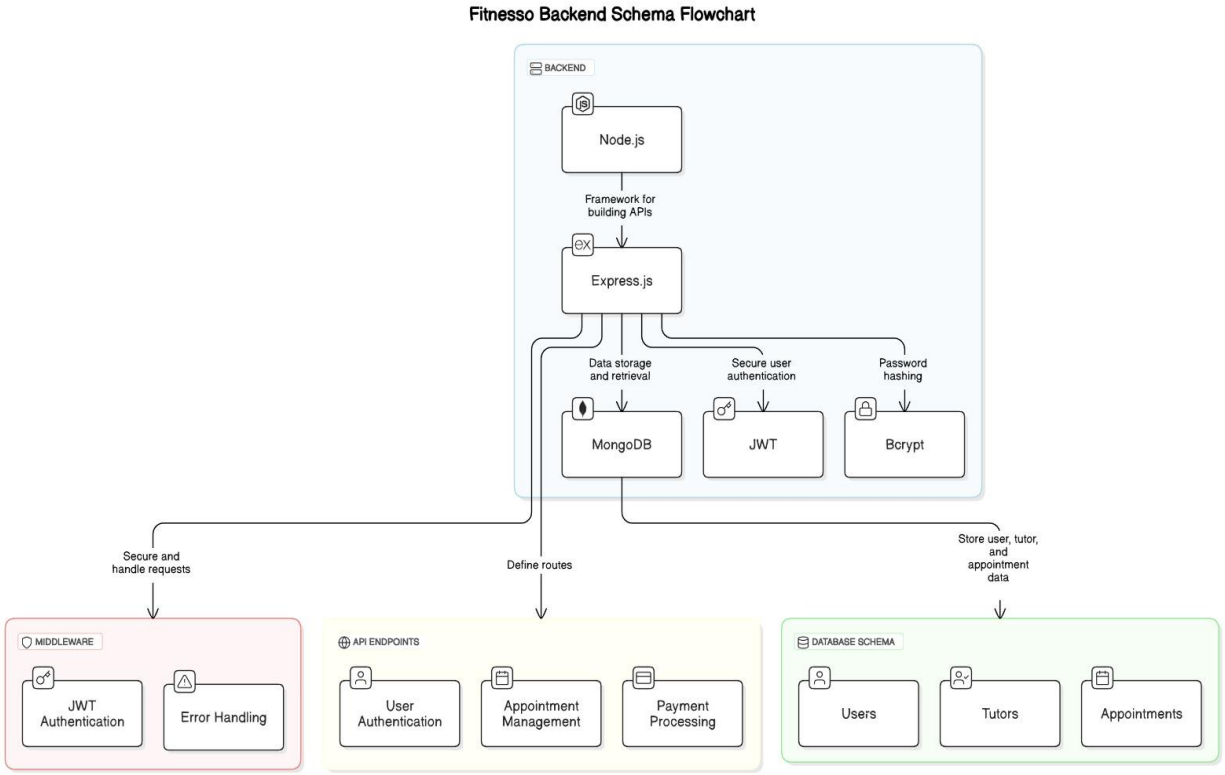
  - Passwords are hashed using bcrypt before being saved.

- Tutor Model(TutorModel.js)

  Fields:

  - userId(String, required): user id.
  - docId(String, required): stores tutor id
  - slotDate(String, required): slot date
  - userData(String, required): user information
  - docData(String, required): tutor information
  - amount(Number, required): payment fee
  - date(Number, required): date of session
  - cancelled(Boolean): whether session cancelled or not
  - payment(Boolean): whether payment completed or not
  - isCompleted(Boolean): is completed or not.

BACKEND DESIGN DIAGRAM



SECURITY CONSIDERATIONS:

Authentication:

- **JWT Tokens:**
    - Securely generated and signed with a secret key.
    - Secure login and token storage in the client (preferably in **localStorage** or **sessionStorage**).
- **Password Security**:
    - Passwords hashed using bcrypt before storing in the database.
    - Plain passwords are never stored or logged.

Authentication:

- **Protected Routes:**
    - Backend routes require valid JWT tokens.
    - Frontend routes use higher-order components to restrict access.
- **Input Validation**:
    - Validation rules applied on both client and server sides.

CORS Configuration:

- **Access-Control Policies:**
    - Configured to allow requests from trusted origins.
    - Proper headers set for **Access-Control-Allow-Origin**, **Methods**, and **Header**.

API DESIGN (RESTful)

- **User Authentication and Management**:
    - POST **/register**: registers a new user (user or tutor).
    - POST **/login**: logs in a user and returns a token (likely JWT for authentication).
    - GET **/user**: retrieves the details of the currently logged-in user.

- **Appointment Management**:
    - POST **/appointments**: books a new appointment between user and tutor.
    - GET **/appointments**: retrieves all appointments for the logged-in user (user or tutor).
    - GET /**appointments/**:**id** : retrieves details of a specific appointment.
    - DELETE **/appointments/:id** : cancels or deletes an existing appointment

- **Admin authentication**:
    - POST **/add-tutor**: to add tutor to the database.
    - POST /**login**: for the admin to login
    - GET **/api/admin**: to redirect to admin route
    - GET **/api/user**: to redirect to user route

- **Data Flow**:

    1. Users log in and receive a JWT token.
    2. Users browse tutors and book appointments.
    3. Payments are processed and stored in the database.
    4. Tutors and students can fetch or update their data via API calls.

- **Error Handling**:

    1 **Backend**: Implement middleware to handle database errors, authentication failures, etc.

    2 **Frontend**: Display user-friendly error messages.

DEPLOYMENT

- **Frontend:**
  - Deployed on Vercel for easy React.js hosting.
- **Backend:**
  - Deployed on AWS
- **Database:**
  - MongoDB Compass for MongoDB.

FUTURE ENHANCEMENTS

- **Automated Reminders**: Email or SMS reminders for upcoming appointments.

- **Advanced Search**: Filter tutors by specialization, rating, or availability.

Google Meet Integration: Automatically generate a Google Meet link for each appointment. The link will be sent via email or SMS, along with reminders.

CONCLUSION

The website for booking fitness classes has been successfully developed with a user-friendly interface and robust functionality for both users and tutors. With seamless registration and login features, users can easily access the platform to book appointments, make payments, and cancel bookings

The website is fully integrated with a backend, ensuring smooth data management and storage, enhancing the user experience. With the integration of both frontend and backend technologies, the platform provides a reliable and efficient way for users to engage with fitness instructors, making it a valuable tool in the wellness space.