Mini Project Report

1. Title Page

Project Title: Health & Wellness Platform: A Full-Stack Appointment Management System

Team Members:

Student's Name	PRN
Yashika Raj	**1032232524**
Tanushree Dharmavaram	**1032232615**
Vanshika Kaushal	**1032233712**

Panels: TY Btech CSF Panel A

Course Name: Full Stack Development Course

Submission Date: October 31, 2025

2. Abstract

This report details the design, development, and implementation of the "Health & Wellness Platform," a full-stack web application. The primary purpose of this project is to create a multi-user system that seamlessly connects patients, healthcare practitioners, and administrators. The platform implements three distinct roles—User, Practitioner, and Admin—each with a unique dashboard and set of permissions. Key features include role-based token authentication, a module for browsing and booking appointments, comprehensive dashboards for practitioners to manage their schedules and revenue, and a super-admin dashboard for user management and platform-wide analytics. The system is built with a decoupled architecture, featuring a Node.js and Express.js RESTful API on the backend, a MongoDB database, and a dynamic frontend built with Vanilla JavaScript, HTML5, and Bootstrap 5. The project successfully achieves its objective of creating a scalable, secure, and user-friendly platform for managing healthcare appointments.

3. Introduction

Background and Motivation

In today's digital age, the healthcare sector is increasingly migrating online. Patients expect the convenience of finding, booking, and managing their healthcare appointments from their computers or mobile devices. Likewise, practitioners require efficient tools to manage their schedules, patient data, and revenue. This project, the "Health & Wellness Platform," is motivated by the need for a simple, centralized, and scalable solution that bridges this gap between patients and providers.

Problem Statement

The core problem this project addresses is the lack of an integrated system for managing the complete lifecycle of a healthcare appointment.

- **For Patients:** Difficulty in finding verified practitioners and booking appointments without back-and-forth phone calls.
- **For Practitioners:** Inefficiently managing schedules using disparate tools (e.g., paper calendars, simple spreadsheets), leading to difficulties in tracking revenue and patient history.
- **For Administrators:** No centralized oversight to manage the platform's users (both patients and practitioners) or to understand the platform's overall performance.

Project Objectives

The primary objectives for this mini-project are:

1. **To Develop a Secure, Role-Based System:** Implement a secure authentication system for three distinct user roles: User (patient), Practitioner, and Admin.
2. **To Create an End-to-End Booking Workflow:** Allow users to browse a list of available practitioners and book an appointment for a specific date and time.
3. **To Build Practitioner Dashboards:** Provide practitioners with a private dashboard to view upcoming appointments, manage their profile, and track revenue analytics.
4. **To Implement an Admin Panel:** Create a super-admin dashboard for user management (viewing all users/practitioners) and platform-wide analytics (revenue, appointment trends).

5. **To Implement Payment and Reporting:** Develop a (simulated) system for users to view and pay for their appointments and to download receipts.

Scope and Limitations

Scope: The project's scope covers the full-stack implementation of the features mentioned above. This includes the database schema design, backend RESTful API development, and the creation of ten distinct frontend HTML pages to handle all user interactions, from login to booking to dashboard management.

Limitations:

- **Simulated Payments:** The payment system (payments.html) updates the status in the database but does not integrate with a real-world payment gateway like Stripe or Razorpay.
- **No Real-time Features:** The platform does not include real-time chat or video consultations (telehealth).
- **Manual Practitioner Verification:** The process for adding and verifying new practitioners is assumed to be handled manually by the admin, though the admin dashboard has the foundation for this.

4. Literature Review

This project is modeled on existing successful telehealth and appointment booking platforms. A review of market leaders such as **Practo**, **Zocdoc**, and **Setmore** was conducted to understand core industry features.

These platforms commonly share a three-sided marketplace model (patient, doctor, clinic/admin). The key takeaway from this review was the critical importance of a decoupled architecture. Many modern platforms use a "headless" approach, where a central API serves data to multiple clients (a web app, an iOS app, an Android app).

This project adopts this "headless" principle by building a robust backend API (server.js) that is entirely separate from the frontend client. While this project's frontend is a traditional Multi-Page Application (MPA) built with Vanilla JavaScript, the architecture ensures that a React, Vue, or mobile application could be easily built on top of the existing backend in the future. The project also borrows concepts from modern Electronic Medical Record (EMR) systems, specifically in the practitioner's dashboard, which provides a foundation for viewing patient reports and history.

5. Methodology

Choice of Technologies and Tools

The technology stack was chosen to be modern, scalable, and open-source, representing a "MERN-V" (MongoDB, Express, Node, Vanilla JS) stack.

- **Frontend:**
    - **HTML5 & CSS3:** For structuring and styling the web pages.
    - **Bootstrap 5:** Utilized as the primary CSS framework for a responsive, mobile-first design. This is visible in all 10 HTML files for components like cards, modals, navbars, and the grid layout.
    - **Vanilla JavaScript (ES6+):** Used for all client-side logic, including fetch() API calls to the backend, DOM manipulation, and managing user state (via localStorage).
    - **Chart.js:** Implemented in admin_dashboard.html and practitioner_dashboard.html to render visual analytics charts.
    - **Bootstrap Icons:** Used for iconography throughout the application.
- **Backend:**
    - **Node.js:** As the server-side JavaScript runtime environment.
    - **Express.js:** As the web framework for building the RESTful API, managing routes (/api/auth, /api/users, etc.), and handling middleware.
    - **body-parser:** Middleware to parse incoming JSON request bodies.
    - **cors:** Middleware to enable Cross-Origin Resource Sharing, allowing the frontend to make requests to the backend.
- **Database:**
    - **MongoDB:** A NoSQL document database used to store all application data, including users, appointments, and payments, in a flexible JSON-like format.
    - **(Inferred) Mongoose:** Likely used as the Object Data Modeling (ODM) library to manage database connections, schemas, and models (as suggested by the connectDB call).

- **Authentication:**
  - **JSON Web Tokens (JWT):** Used for a stateless authentication strategy. The login.html page sends credentials to /api/auth/login, receives a token, and stores it in localStorage. This token is then sent in the Authorization header of all subsequent authenticated requests.

System Architecture

The project follows a **decoupled 3-Tier Architecture**.

1. **Presentation Tier (Client):** This is the frontend, consisting of the 10 HTML, CSS, and JS files. It runs entirely in the user's browser. It is responsible for rendering the UI and communicating with the API via fetch() requests. It is "dumb" and contains no business logic.
2. **Logic/Application Tier (Server):** This is the Node.js/Express.js backend (server.js). It acts as the "brain" of the application. It handles all business logic, route definitions, user authentication, and request/response processing. It is completely decoupled from the frontend.
3. **Data Tier (Database):** This is the MongoDB database. It is responsible for the persistent storage and retrieval of all data. It only communicates directly with the Application Tier.

Development Process

The development process followed an iterative, Agile-like methodology.

1. **Backend First:** The backend was developed first, starting with the database models (for Users, Appointments, Payments) and the corresponding RESTful API endpoints for basic CRUD (Create, Read, Update, Delete) operations.
2. **Authentication:** The auth routes and JWT strategy were implemented next, as this is central to the application.
3. **Frontend Sprints:** The frontend was built in logical "sprints," focusing on one user role at a time.
   - *Sprint 1:* Login (login.html) and User Dashboard/Workflow (home.html, experts.html, booking.html, payments.html).
   - *Sprint 2:* Practitioner Dashboard (practitioner_dashboard.html).
   - *Sprint 3:* Admin Dashboard (admin_dashboard.html).

4. **Integration & Testing:** After each sprint, the frontend pages were connected to the backend API and tested to ensure the data flow was correct (e.g., "Does a new booking in booking.html show up in practitioner_dashboard.html?").

6. Results and Discussion

Presentation of Outcomes

The project was successfully completed, resulting in a fully functional web platform with three distinct user-facing modules.

- **Outcome 1: Role-Based Access Control (RBAC):** The login system (login.html) successfully authenticates and redirects users based on their role, storing the token and role in localStorage to manage the session.
  - **User:** Redirected to home.html.
  - **Practitioner:** Redirected to practitioner_dashboard.html.
  - **Admin:** Redirected to admin_dashboard.html.
- **Outcome 2: End-to-End Appointment Booking Workflow:** The core user story is fully functional. A user can log in, browse the list of practitioners (experts.html), select one, and be taken to the booking.html page with the practitioner pre-selected. Upon booking, the appointment is created in the database.
- **Outcome 3: Data-Driven Dashboards:** All three dashboards are functional and populate themselves with live data from the backend.
  - The **Practitioner Dashboard** (practitioner_dashboard.html) correctly displays upcoming and past appointments, lists patient reports, and renders revenue analytics using Chart.js.
  - The **Admin Dashboard** (admin_dashboard.html) provides a high-level overview of the entire platform, including user/practitioner counts, total revenue, and appointment trends, also powered by Chart.js.
- **Outcome 4: API-Driven Content:** All dynamic content is loaded from the backend API. For example, the experts.html page does not contain hard-coded data; it fetches the list of practitioners from the /api/users?role=practitioner endpoint.

Discussion of Successes

- **Decoupled Architecture:** The separation of the frontend and backend is the project's biggest success. It makes the system highly maintainable and scalable.

- **Lightweight Frontend:** By using Vanilla JavaScript instead of a large framework, the client-side application is extremely fast and has minimal dependencies.
- **Comprehensive Functionality:** The project successfully models a complex real-world system, including not just one, but three different user perspectives.

Discussion of Challenges

- **Client-Side State Management:** The primary challenge of using a Vanilla JS MPA (Multi-Page Application) was state management. Because each HTML file is a new "page," the user's token, ID, and role had to be persistently stored in localStorage and re-read on every single page. This is less efficient and secure than a Single-Page Application (SPA) framework like React, which can hold state in memory.
- **API Security:** Ensuring all API endpoints were properly secured was a challenge. Logic had to be implemented on the backend to check the user's JWT *and* their role before returning data (e.g., ensuring only a user with the 'admin' role could access the /api/admin/dashboard endpoint).

7. Conclusion

Summary of Findings

The "Health & Wellness Platform" project successfully achieves all the objectives set out at its inception. It is a fully-functional, full-stack application that provides a complete workflow for booking and managing healthcare appointments. The use of a decoupled architecture with a Node.js/Express.js API and a Vanilla JS/Bootstrap frontend proves to be a robust and effective combination. The project demonstrates a clear understanding of full-stack development principles, from database design and API security to responsive frontend UI/UX.

Recommendations for Future Work

While the mini-project is complete, the platform has a strong foundation for future expansion.

1. **Integrate a Real Payment Gateway:** Replace the simulated payment system with a real-world solution like Stripe or Razorpay to handle actual financial transactions.
2. **Add Telehealth Features:** Implement real-time video consultations using WebRTC or a third-party API (like Twilio) to move beyond simple appointment booking.
3. **Migrate Frontend to a SPA:** To solve the state management challenges, the frontend could be progressively migrated to a modern framework like React, Angular, or Vue.js, which would provide a more seamless user experience.

4. **Implement Notifications:** Add an email or SMS notification system (using a service like SendGrid or Twilio) to remind users and practitioners of their upcoming appointments.

8. References

(Note: Please replace these with your actual research sources, formatted in your required style, e.g., APA.)

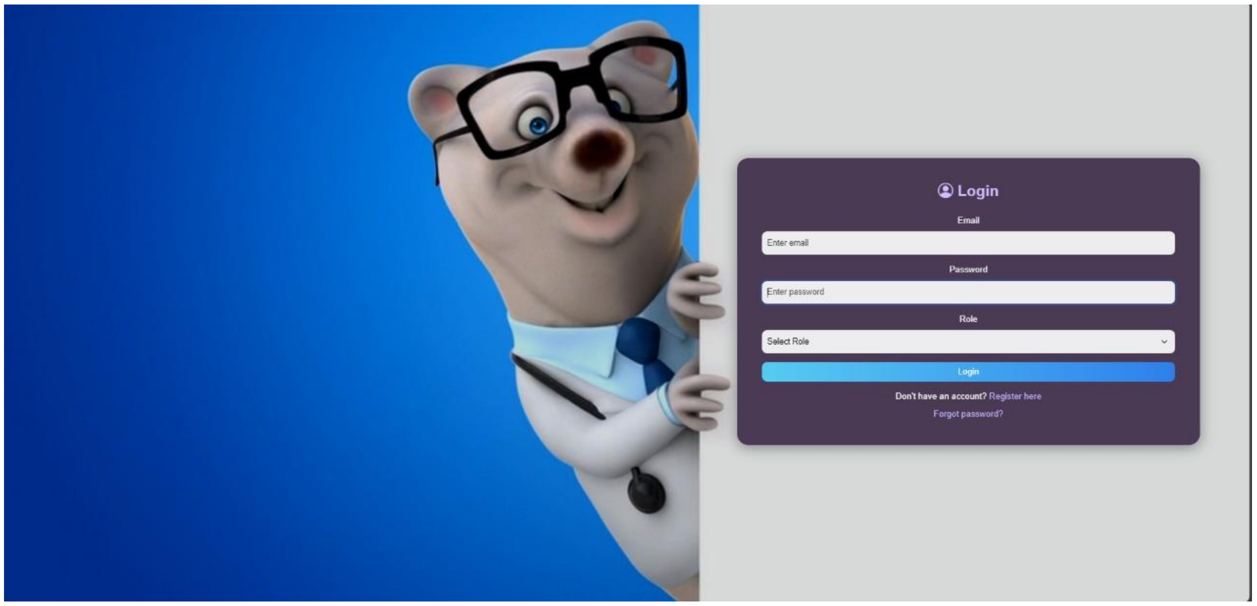
- Bootstrap 5 Documentation. (2025). *Get started with Bootstrap*. Retrieved from https://getbootstrap.com/docs/5.3/getting-started/introduction/
- Chart.js Organization. (2025). *Chart.js Documentation*. Retrieved from https://www.chartjs.org/docs/latest/
- Mozilla Developer Network (MDN). (2025). *Fetch API*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- Node.js Foundation. (2025). *Node.js Documentation*. Retrieved from https://nodejs.org/en/docs/

9. Appendices

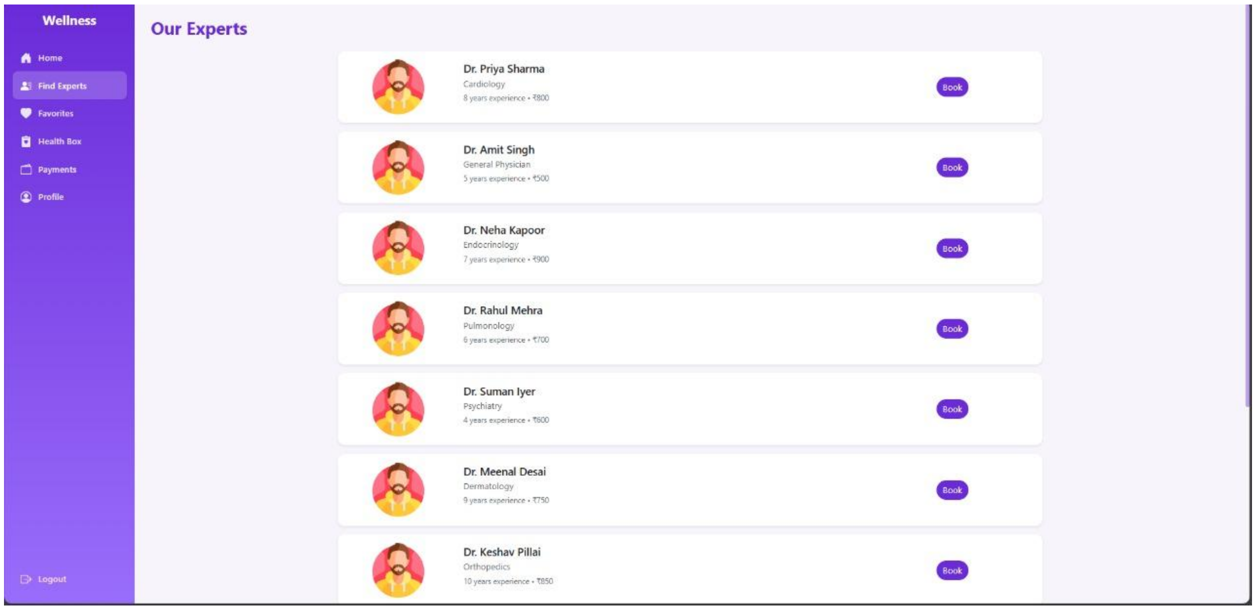
Appendix A: Key Screenshots

(Please insert your own screenshots here. I recommend including the following):

- **Screenshot 1:** The Login Page (login.html) showing the role selection.



- **Screenshot 2:** The User Dashboard (home.html) showing upcoming appointments.

- 

Our Experts

Dr. Priya Sharma
Cardiology
8 years experience • ₹800

Dr. Amit Singh
General Physician
5 years experience • ₹500

Dr. Neha Kapoor
Endocrinology
7 years experience • ₹900

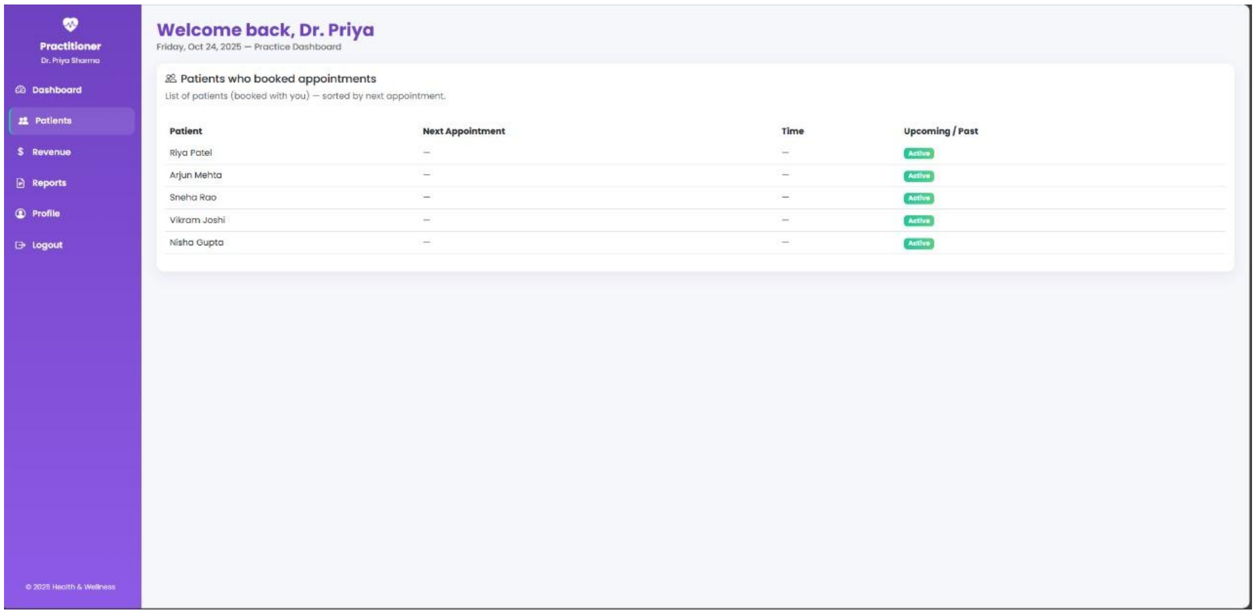
Dr. Rahul Mehra
Pulmonology
6 years experience • ₹700

Dr. Suman Iyer
Psychiatry
4 years experience • ₹600

Dr. Meenal Desai
Dermatology
9 years experience • ₹750

Dr. Keshav Pillai
Orthopedics
10 years experience • ₹850

- **Screenshot 3:** The Find Experts Page (experts.html) showing the list of practitioners.

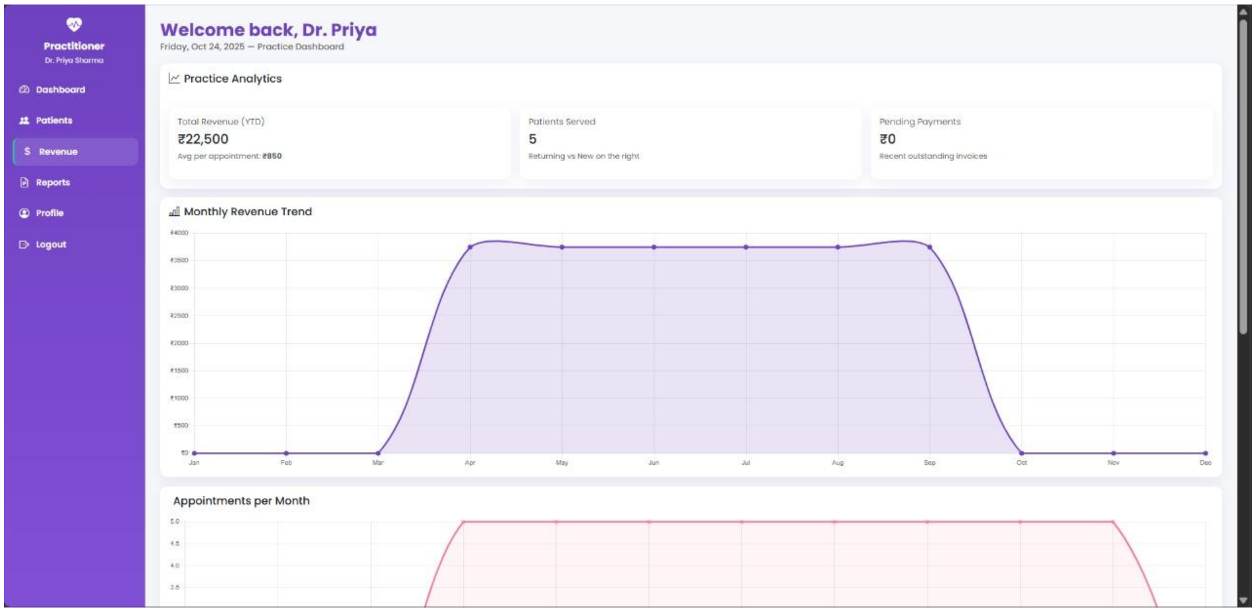
- 

Welcome back, Dr. Priya
Friday, Oct 24, 2025 — Practice Dashboard

Patients who booked appointments
List of patients (booked with you) — sorted by next appointment.

Patient	Next Appointment	Time	Upcoming / Past
Riya Patel	—	—	Active
Arjun Mehta	—	—	Active
Sneha Rao	—	—	Active
Vikram Joshi	—	—	Active
Nisha Gupta	—	—	Active

- **Screenshot 4:** The Practitioner Dashboard (practitioner_dashboard.html) showing analytics charts.

- 
Practitioner
Dr. Priya Sharma

Dashboard
Patients
$ Revenue
Reports
Profile
Logout

Welcome back, Dr. Priya
Friday, Oct 24, 2025 — Practice Dashboard

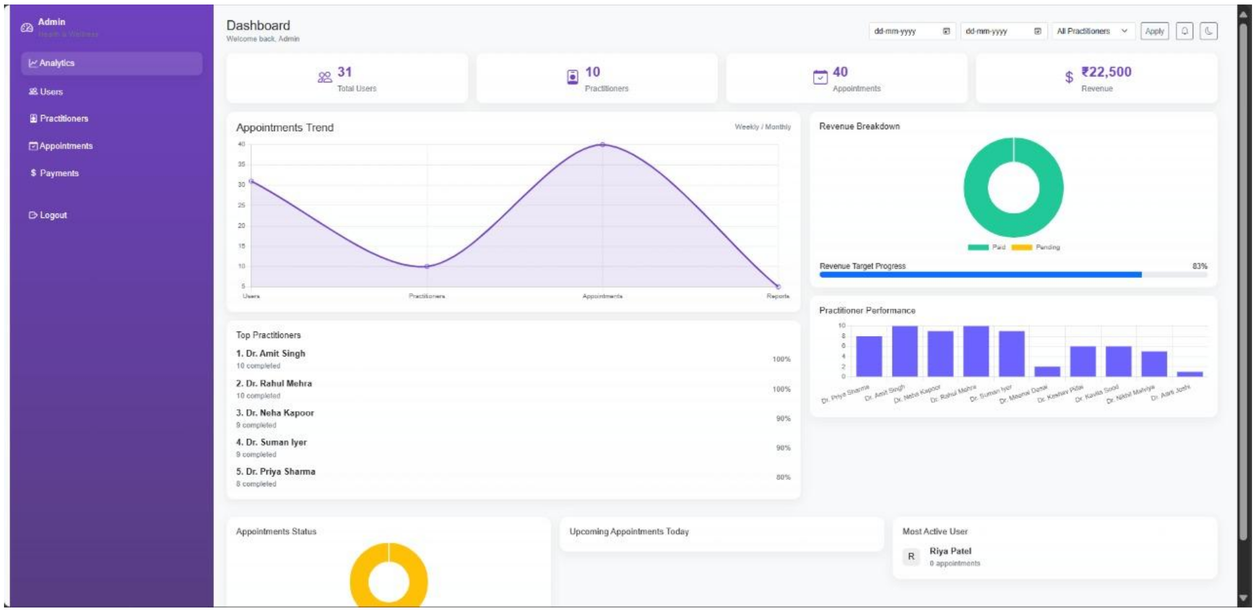
Practice Analytics

Total Revenue (YTD)
₹22,500
Avg per appointment: ₹850

Patients Served
5
Returning vs New on the right

Pending Payments
₹0
Recent outstanding invoices

- **Screenshot 5:** The Admin Dashboard (admin_dashboard.html) showing user management and platform analytics.

- 
Admin

Analytics
Users
Practitioners
Appointments
$ Payments
Logout

Dashboard
Welcome back, Admin

dd-mm-yyyy dd-mm-yyyy All Practitioners Apply

31 Total Users
10 Practitioners
40 Appointments
₹22,500 Revenue

Revenue Target Progress 83%

Top Practitioners
1. Dr. Amit Singh
10 completed 100%
2. Dr. Rahul Mehra
10 completed 100%
3. Dr. Neha Kapoor
9 completed 90%
4. Dr. Suman Iyer
9 completed 90%
5. Dr. Priya Sharma
8 completed 80%

Appointments Status

Upcoming Appointments Today

Most Active User
R Riya Patel
0 appointments

Appendix B: Core Code Snippets

(Please insert key code snippets here. I recommend including):

- **Snippet 1: Backend API Routing (server.js)**

  // backend/server.js

  ...

```
app.use('/api/auth', authRoutes);

app.use('/api/users', userRoutes);

app.use('/api/appointments', apptRoutes);

app.use('/api/payments', paymentRoutes);

app.use('/api/practitioner', require('./routes/practitioner'));

app.use('/api/admin', require('./routes/admin'));

...
```

- **Snippet 2: Client-Side Authentication (login.html)**

```
  // login.html
 ...
 document.getElementById("loginForm").addEventListener("submit", async (e) => {
 ...
 const res = await fetch(`${API_BASE}/login`, {
 method: "POST",
 headers: { "Content-Type": "application/json" },
body: JSON.stringify({ email, password, role })
});


const data = await res.json();
...
// Save user session
localStorage.setItem("token", data.token);
localStorage.setItem("user_id", data.user._id);
localStorage.setItem("role", data.user.role);
```

```
  ...
 // Redirect based on role
 if (data.user.role === "admin") {
  window.location.href = "admin_dashboard.html";
 } ...
});
```

- **Snippet 3: Authenticated API Call (booking.html)**

```
 // booking.html
...
const res = await fetch("http://localhost:3000/api/appointments", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`, // Token from localStorage
  },
  body: JSON.stringify({ practitionerId, date, time, type }),
});
...
```