Full Stack Development with MERN

1. Introduction

Project Title: DocSpot – Doctor Appointment Booking System

Team ID: LTVIP2025TMID46139

Team Size: 3

Team Leader: Gangisetty Sai Meghana

Team Member: Chandu Sri Nellepalli

Team Member: Shaik Reshma

2. Project Overview

Purpose:

DocSpot is an online doctor appointment booking system built using the MERN stack. It aims to streamline healthcare by allowing patients to book appointments with doctors easily and securely through a web-based platform.

Features:

- Secure role-based login/signup (Patient, Doctor, Admin)

- Doctor registration with admin approval

- Appointment scheduling and management

- Doctor filtering by specialty/location

- Real-time status updates and appointment tracking

- Admin dashboard for managing users and appointments

3. Architecture

Frontend:

Built with React.js using functional components, React Router for navigation, Axios for API calls, and Tailwind CSS or Bootstrap for styling.

Backend:

Developed using Node.js and Express.js. RESTful APIs, middleware for authentication and error handling, and role-based access control.

Database:

Uses MongoDB with Mongoose ODM. Collections include Users, Appointments, Doctors. Includes validation and indexing for performance.

4. Setup Instructions

Prerequisites:

- Node.js and npm

- MongoDB (local or Atlas)

- Git

- VS Code or any code editor

Installation:

```
# Clone the repository
git clone [your-github-repo-link]
cd project-folder

# Install frontend dependencies
cd client
npm install

# Install backend dependencies
cd ../server
npm install

# Add .env files in server and client folders
```

5. Folder Structure

Client:

```
client/
|   ├── public/
| ├── src/
|  ├── components/
|  ├── pages/
|  ├── routes/
|  ├── utils/
|  └── App.js
```

Server:

```
server/
├── controllers/
├── models/
├── routes/
├── middleware/
├── config/
├── .env
└── server.js
```

6. Running the Application

```
# Start backend server
cd server
npm start

# Start frontend server
cd ../client
npm start
```

7. API Documentation

User APIs:
- POST /api/users/register – Register user
- POST /api/users/login – Login user

Doctor APIs:
- POST /api/doctors/register – Register doctor
- GET /api/doctors – Get all approved doctors
- PATCH /api/doctors/approve/:id – Admin approves doctor

Appointment APIs:
- POST /api/appointments – Book appointment
- GET /api/appointments/:userId – Get user appointments
- PATCH /api/appointments/status/:id – Update appointment status

8. Authentication

JWT-based authentication. Tokens stored in HTTP-only cookies or localStorage. Role-based authorization middleware. Passwords hashed using bcrypt.

9. User Interface

Patient UI: Login/Signup, Search doctors, Book/view appointments

Doctor UI: Register profile, View appointments

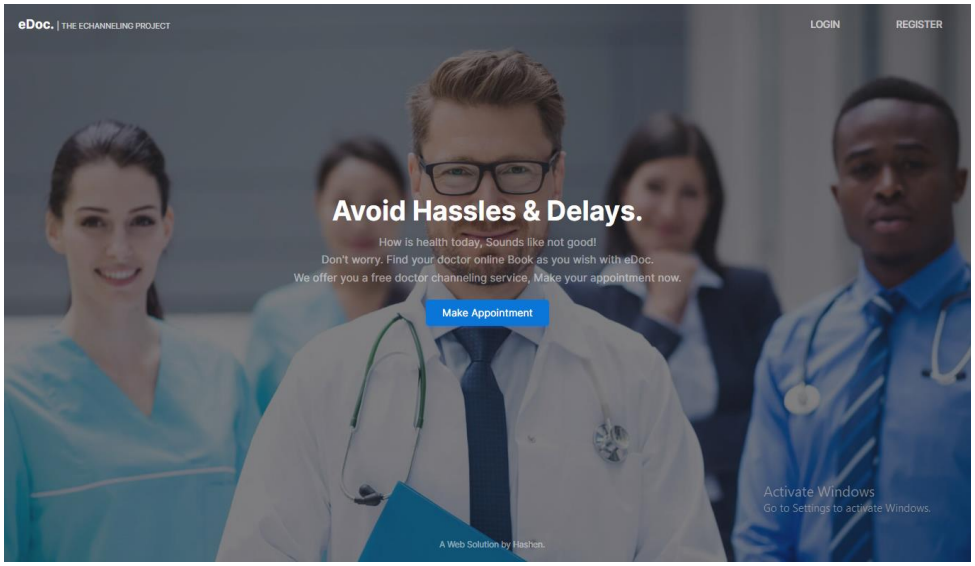
Admin UI: Approve doctors, Manage users and appointments

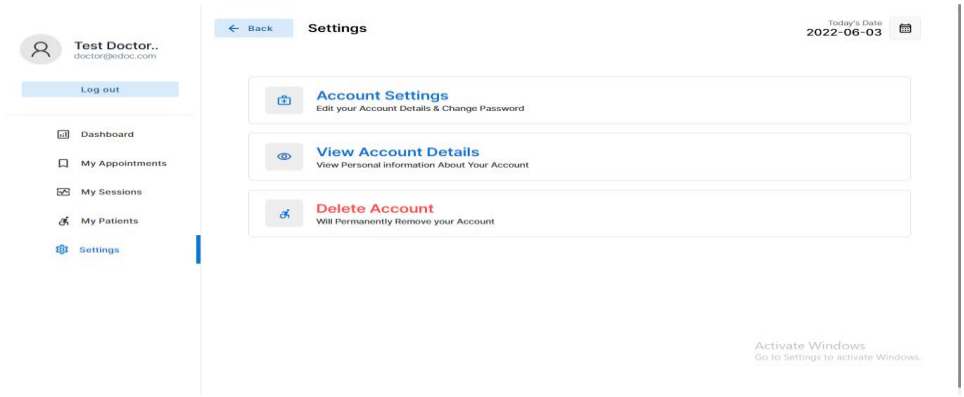
10. Testing

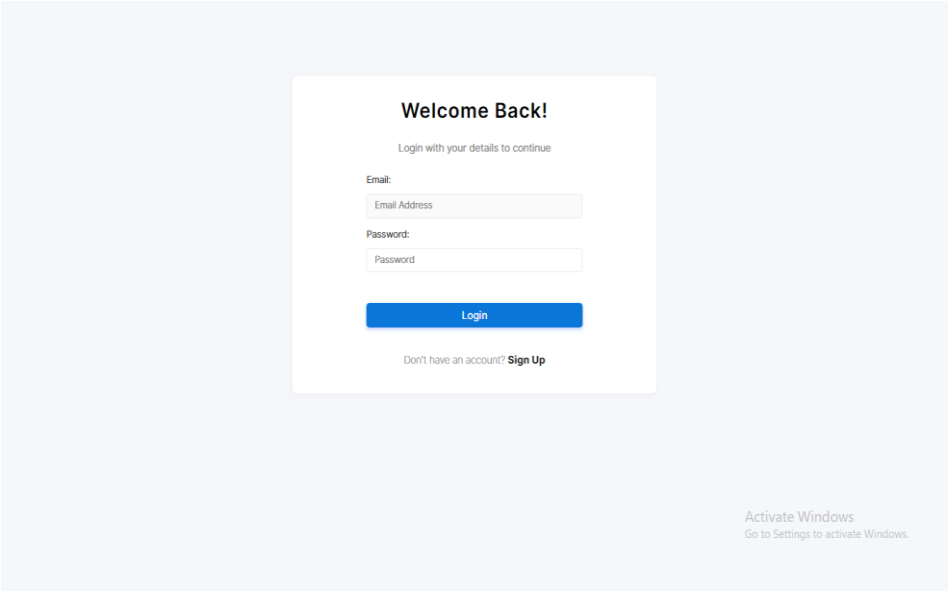
Manual testing using Postman for all APIs. Form validation for frontend. Possible extension: Jest or Mocha for backend unit testing.

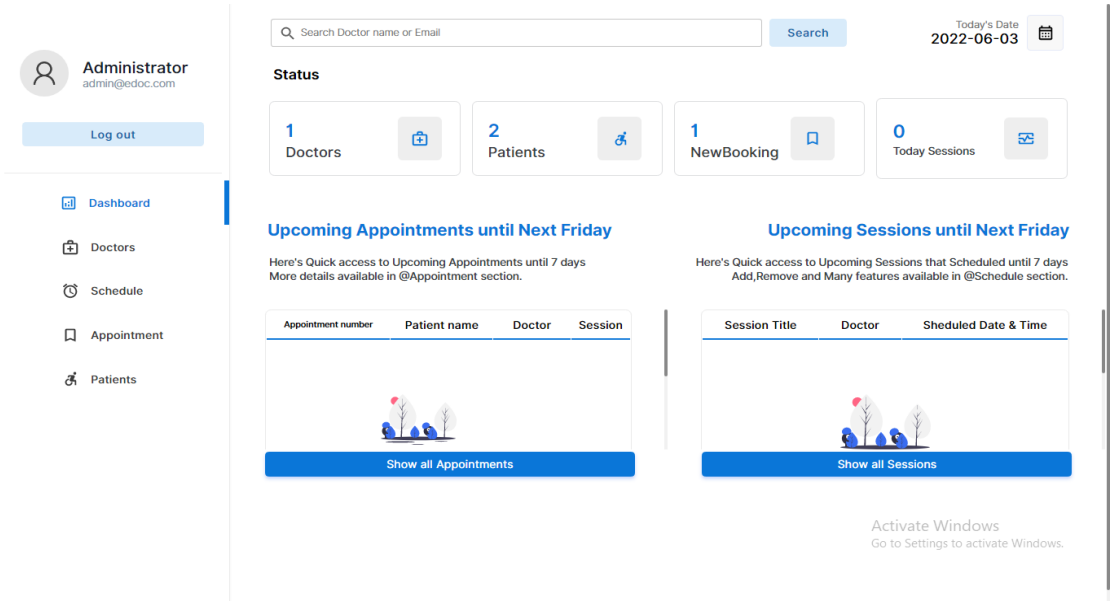
11. Screenshots or Demo

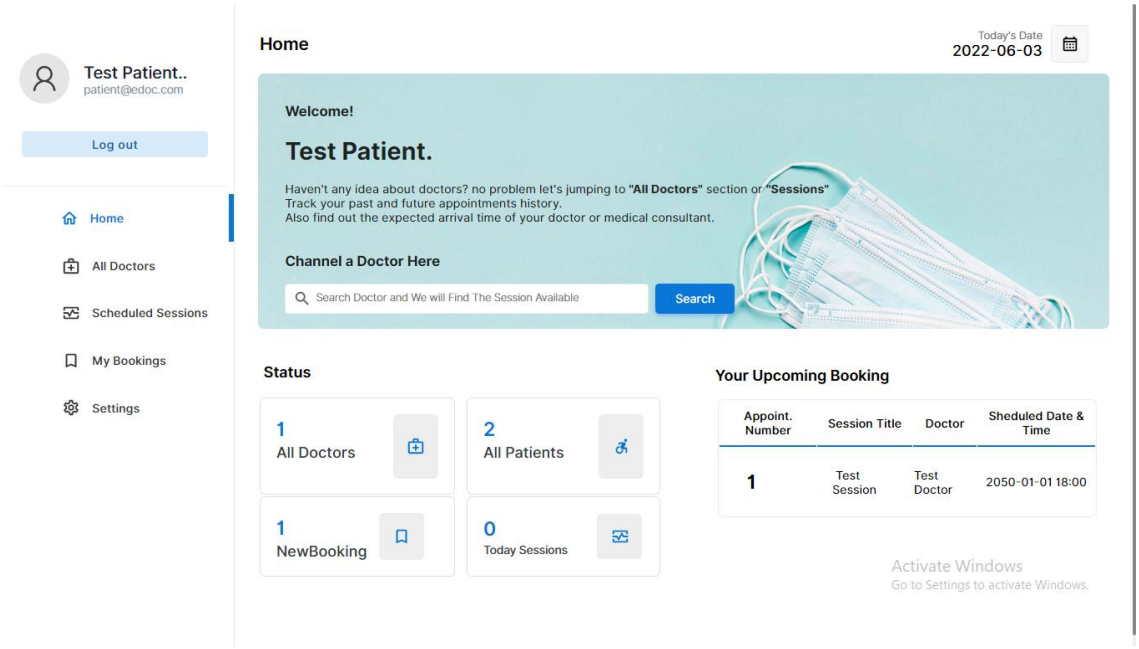
Screenshots regarding project


eDoc. | THE ECHANNELING PROJECT
LOGIN
REGISTER
Avoid Hassles & Delays.
How is health today, Sounds like not good!
Don't worry. Find your doctor online Book as you wish with eDoc.
We offer you a free doctor channeling service, Make your appointment now.
Make Appointment
A Web Solution by Hashen.
Activate Windows
Go to Settings to activate Windows.


Test Doctor..
doctor@edoc.com
Log out
Dashboard
My Appointments
My Sessions
My Patients
Settings
Back
Settings
Today's Date
2022-06-03
Account Settings
Edit your Account Details & Change Password
View Account Details
View Personal information About Your Account
Delete Account
Will Permanently Remove your Account
Activate Windows
Go to Settings to activate Windows.


Welcome Back!
Login with your details to continue
Email:
Email Address
Password:
Password
Login
Don't have an account? Sign Up
Activate Windows
Go to Settings to activate Windows.





12. Known Issues

- Limited testing coverage

- No email/SMS notification system yet

- UI not fully responsive on all devices

13. Future Enhancements

- Integrate email/SMS notifications

- Add chat functionality between patient and doctor

- Create mobile app using React Native

- Admin analytics dashboard

- Payment gateway integration

Additional Project Information

FRONTEND TECHNOLOGIES

Bootstrap and Material UI: Provide a responsive and modern UI that adapts to various devices, ensuring a user-friendly experience.
Axios: A promise-based HTTP client for making requests to the backend, ensuring smooth data communication between the frontend and server.

BACKEND FRAMEWORK

Express.js: A lightweight Node.js framework used to handle server-side logic, API routing, and HTTP request/response management, making the backend scalable and easy to maintain.

DATABASE AND AUTHENTICATION

MongoDB: A NoSQL database used for flexible and scalable storage of user data, doctor profiles, and appointment records. It supports fast querying and large data volumes.
JWT (JSON Web Tokens): Used for secure, stateless authentication, allowing users to remain logged in without requiring session storage on the server.
Bcrypt: A library for hashing passwords, ensuring that sensitive data is securely stored in the database.

ADMIN PANEL & GOVERNANCE

Admin Interface: Provides functionality for platform admins to approve doctor registrations, manage platform settings, and oversee day-to-day operations.
Role-based Access Control (RBAC): Ensures different users (patients, doctors, admins) have appropriate access levels to the system's features and data, maintaining privacy and security.

SCALABILITY AND PERFORMANCE

MongoDB: Scales horizontally, supporting increased data storage and high user traffic as the platform grows.
Load Balancing: Ensures traffic is evenly distributed across servers to optimise performance, especially during high traffic periods.
Caching: Reduces database load by storing frequently requested data temporarily, speeding up response times and improving user experience.

TIME MANAGEMENT AND SCHEDULING

Moment.js: Utilised for handling date and time operations, ensuring precise appointment scheduling, time zone handling, and formatting.

SECURITY FEATURES

HTTPS: The platform uses SSL/TLS encryption to secure data transmission between the client and server.
Data Encryption: Sensitive user information, such as medical records, is encrypted both in transit and at rest, ensuring privacy and compliance with data protection regulations.

NOTIFICATIONS AND REMINDERS

Email/SMS Integration: Notifications for appointment confirmations, reminders, cancellations, and updates are sent to users via email or SMS, ensuring timely communication.

PRE-REQUISITES

NODE.JS AND NPM: Node.js is a JavaScript runtime for server-side scripting. npm is used to install libraries and manage dependencies.
EXPRESS.JS: Framework to build web APIs. Install via npm install express.
MONGODB: NoSQL database for storing doctor, patient, and appointment data. Set up locally or via MongoDB Atlas.
MOMENT.JS: JavaScript library for managing dates and times.
REACT.JS: Frontend library for creating dynamic web UIs.
ANTD (Ant Design): React UI library for components like tables, forms, and buttons.
HTML, CSS, JS: Essential for structuring and styling web pages.
MONGOOSE: ODM to connect Node.js with MongoDB and manage CRUD operations.

SETUP AND INSTALLATION INSTRUCTIONS

CLONE THE PROJECT REPOSITORY: git clone [your-repo-link]
INSTALL DEPENDENCIES:
- cd frontend && npm install
- cd backend && npm install
START THE DEVELOPMENT SERVER:
- Frontend: npm start (localhost:3000)
- Backend: npm start (localhost:5000 or 8001)

ACCESS THE APPLICATION: Visit the respective local URLs to view frontend and test backend APIs.

DATABASE CONFIGURATION (MONGODB)

Install MongoDB or use MongoDB Atlas. Configure the connection string in the backend .env file.
Run mongod locally for local databases or use MongoDB Atlas connection URI for cloud access.

FINAL CONFIGURATION & RUNNING THE APP

Install concurrently: npm install concurrently --save-dev
Add this to package.json:

```
"scripts": {
  "start": "concurrently \"npm run server\" \"npm run client\"",
  "server": "node backend/server.js",
  "client": "npm start --prefix frontend"
}
```

Run npm start to launch both servers.

VERIFYING THE APP

Check Frontend: Open http://localhost:3000
Check Backend: Use Postman to test APIs like login, register, and appointment booking.

ADDITIONAL SETUP

Version Control: git init, git add ., git commit
Deployment: Use Heroku, AWS, or Vercel for live hosting of frontend/backend.