

1. Introduction

Project Title:

DocSpot — Seamless Appointment Booking for Health

Team Members:

- Ponnada Jaya Chaitanya – Team Leader
- Velpuri Posibhavani – Team Member
- Kone Veera Sravani – Team Member
- Gurugubelli Abhiram – Team Member
- Polisetti Ambica Prasanna – Team Member

Team ID: LTVIP2026TMIDS35287

Team Size: 5

2. Project Overview

Purpose

DocSpot is an online healthcare appointment booking platform that connects patients with doctors. It simplifies booking, managing schedules, and receiving notifications in a secure and user-friendly way.

Key Features

- Patient Registration & Profile Management
- Doctor Search & Filtering
- Appointment Booking & Management
- Doctor Dashboard
- Admin Panel
- Email & SMS Notifications
- Secure Authentication (JWT & bcrypt)

3. Architecture

Frontend:

- Built using React.js
- React Router for navigation
- Bootstrap / Material UI / Ant Design for styling
- Axios for API communication

Backend :

- Node.js with Express.js
- RESTful API architecture
- JWT for authentication
- bcrypt for password hashing

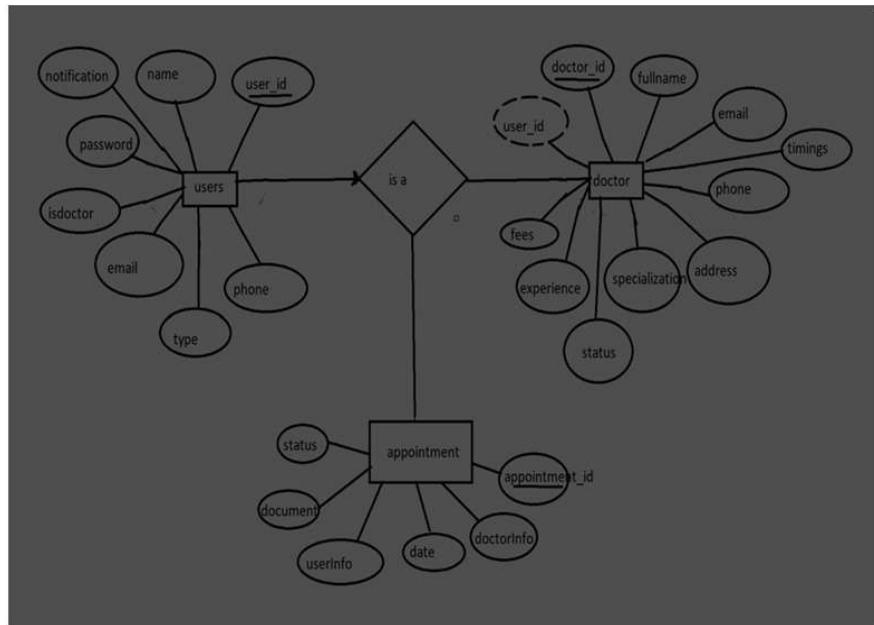
Database :

- MongoDB (NoSQL database)
- Collections:
 - Users
 - Doctors
 - Appointments

System Architecture Diagram of DocSpot



ER Diagram of DocSpot System



4. Setup Instructions

Prerequisites

- Node.js
- MongoDB
- Git

Installation Steps

Clone Repository

```
cd docspot git clone https://github.com/docspot/docspot.git
```

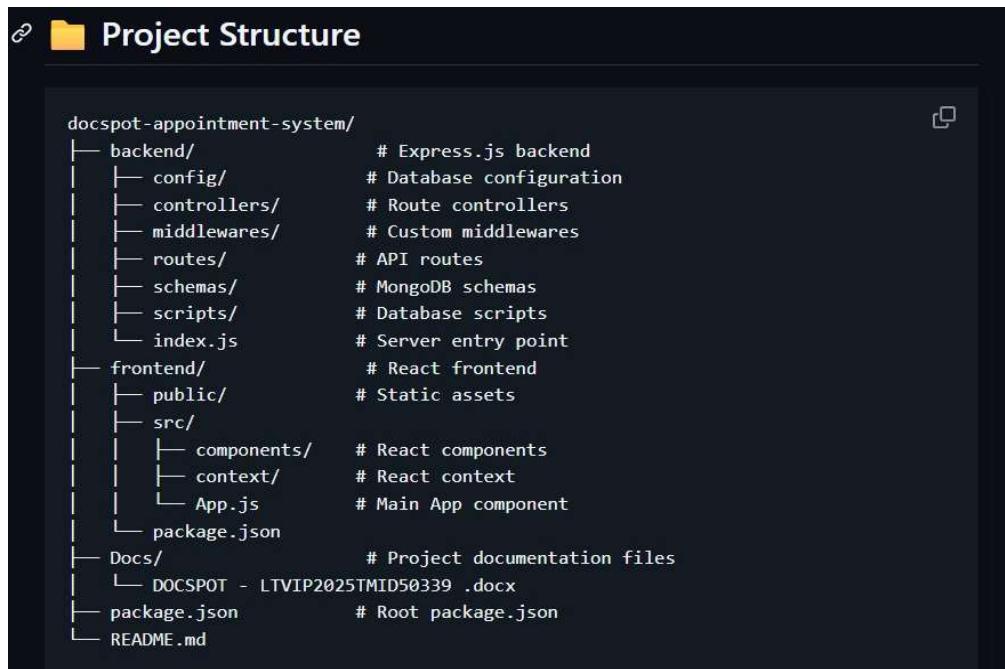
Backend Setup

```
cd backend  
npm install  
npm start
```

Frontend Setup

```
cd frontend  
npm install  
npm start
```

5. Folder Structure



```
docspot-appointment-system/
├── backend/          # Express.js backend
│   ├── config/        # Database configuration
│   ├── controllers/   # Route controllers
│   ├── middlewares/   # Custom middlewares
│   ├── routes/         # API routes
│   ├── schemas/        # MongoDB schemas
│   ├── scripts/        # Database scripts
│   └── index.js        # Server entry point
├── frontend/          # React frontend
│   ├── public/         # Static assets
│   ├── src/
│   │   ├── components/ # React components
│   │   ├── context/    # React context
│   │   └── App.js      # Main App component
│   └── package.json
├── Docs/              # Project documentation files
│   └── DOCS - LTVIP2025TMID50339.docx
└── package.json        # Root package.json
└── README.md
```

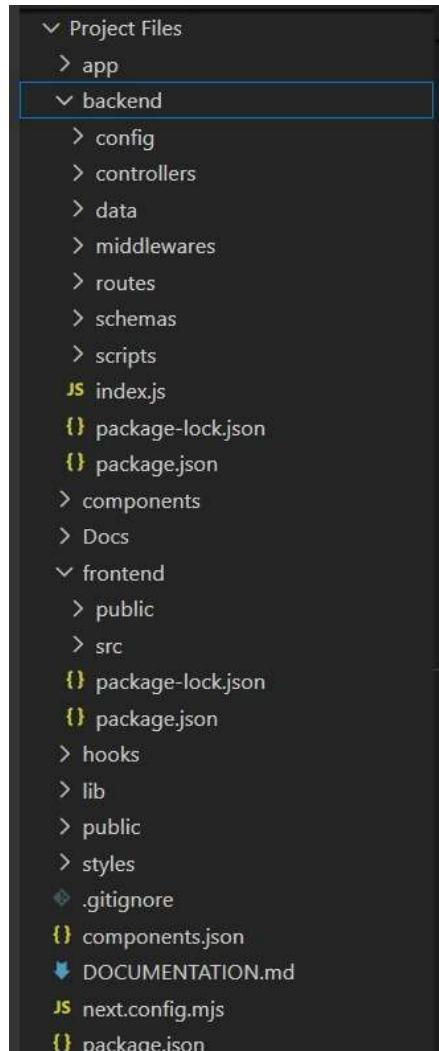
Client (Frontend)

- public/
- src/
- components/
- pages/
- services/
- App.js
- index.js

Server (Backend)

- config/
- controllers/
- models/
- routes/
- middleware/
- uploads/
- server.js

Detailed Project Structure (VS Code View)



6. Running the Application

Frontend:

npm start

Runs on: <http://localhost:3000>

Backend:

npm start

Runs on: <http://localhost:5000>

7. API Documentation

The backend exposes RESTful API endpoints to support frontend operations. Key endpoints include:

The screenshot shows a dark-themed API documentation interface titled "API Endpoints". It lists several categories of routes:

- Authentication**:
 - POST /api/auth/register - User registration
 - POST /api/auth/login - User login
- User Routes**:
 - GET /api/users/profile - Get user profile
 - PUT /api/users/profile - Update user profile
 - POST /api/users/apply-doctor - Apply to become a doctor
- Doctor Routes**:
 - GET /api/doctors - Get all approved doctors
 - GET /api/doctors/appointments - Get doctor's appointments
 - PUT /api/doctors/appointments/:id/status - Update appointment status
- Admin Routes**:
 - GET /api/admin/users - Get all users
 - GET /api/admin/doctors - Get all doctors
 - PUT /api/admin/doctors/:id/approve - Approve doctor application

User Registration:

- POST /api/users/register

Request Body:

```
{  
  "name": "John Doe",  
  "email": "john@example.com",  
  "password": "password123",  
  "type": "patient"  
}
```

Response:

```
{  
  "success": true,  
  "message": "User registered successfully."  
}
```

User Login:

- POST /api/users/login

Request Body:

```
{  
  "email": "john@example.com",  
  "password": "password123"  
}
```

Response:

```
{  
  "token": "<jwt_token>",  
  "user": {  
    "_id": "user_id",  
    "name": "John Doe",  
    "email": "john@example.com",  
    "type": "patient"  
  }  
}
```

Fetch Doctors List:

- GET /api/doctors
- Query Parameters: Optional filters (e.g., specialization, location)

Response:

```
[{  
  "_id": "doctor_id",  
  "fullname": "Dr. Smith",  
  "specialisation": "Cardiology",  
  "experience": "10 years",  
  "fees": 500  
}, ...]
```

Book Appointment

- POST /api/appointments

Request Body:

```
{  
  "doctorId": "doctor_id",  
  "userId": "user_id",  
  "date": "2025-07-01T10:00:00Z",  
  "documents": ["document_url_1", "document_url_2"],  
  "status": "pending"  
}
```

Response:

```
{  
  "success": true,  
  "message": "Appointment booked successfully."  
}
```

Other Endpoints: Include routes for appointment status updates, doctor registration approval, and user profile management.

8. Authentication

- JWT Token-based Authentication
- Password hashing using bcrypt
- Role-based Authorization (Admin, Doctor, Patient)
- Stateless session handling

9. User Interface

Includes:

- Landing Page
- Login & Register Page
- Patient & Doctor Dashboard
- Admin Dashboard
- Booking History

10. Testing

- Unit Testing
- Integration Testing
- Performance Testing
- Security Testing
- User Acceptance Testing

11. Screenshots / Demo

Demo Video: <https://drive.google.com/file/d/18C61itAUcULcNt8OR3e2nthy2zoJcSr/view>

12. Known Issues

- No direct appointment rescheduling
- Minor notification delays
- Mobile UI needs optimization
- File upload size warning missing

13. Future Enhancements

- Direct Rescheduling Feature
- Real-time Chat
- Payment Integration
- Multi-language Support
- AI Doctor Recommendation
- Mobile Application