

Hospital Management System

Introduction:

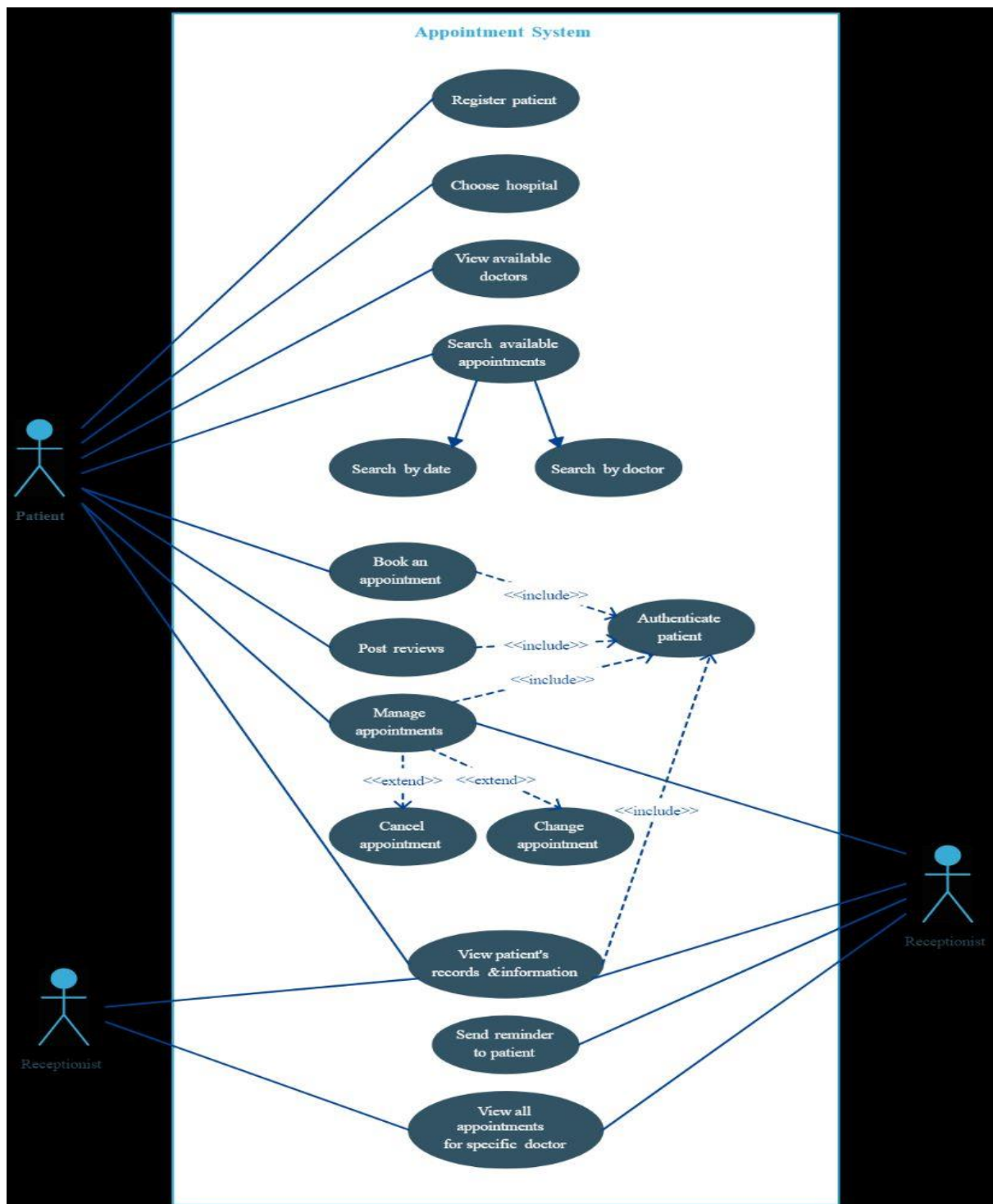
Hospital Management App is a comprehensive software solution designed to streamline and optimize the management and operations of hospitals and healthcare facilities. Our goal is to provide a seamless and efficient system that enhances patient care, simplifies administrative tasks, and improves overall hospital performance.

With our advanced features and user-friendly interface, Hospital Management App empowers hospital administrators, doctors, nurses, and staff to effectively manage appointments, medical records, billing, inventory, and other crucial aspects of healthcare operations. Our solution is tailored to meet the unique needs of hospitals of all sizes, from small clinics to large medical centers.

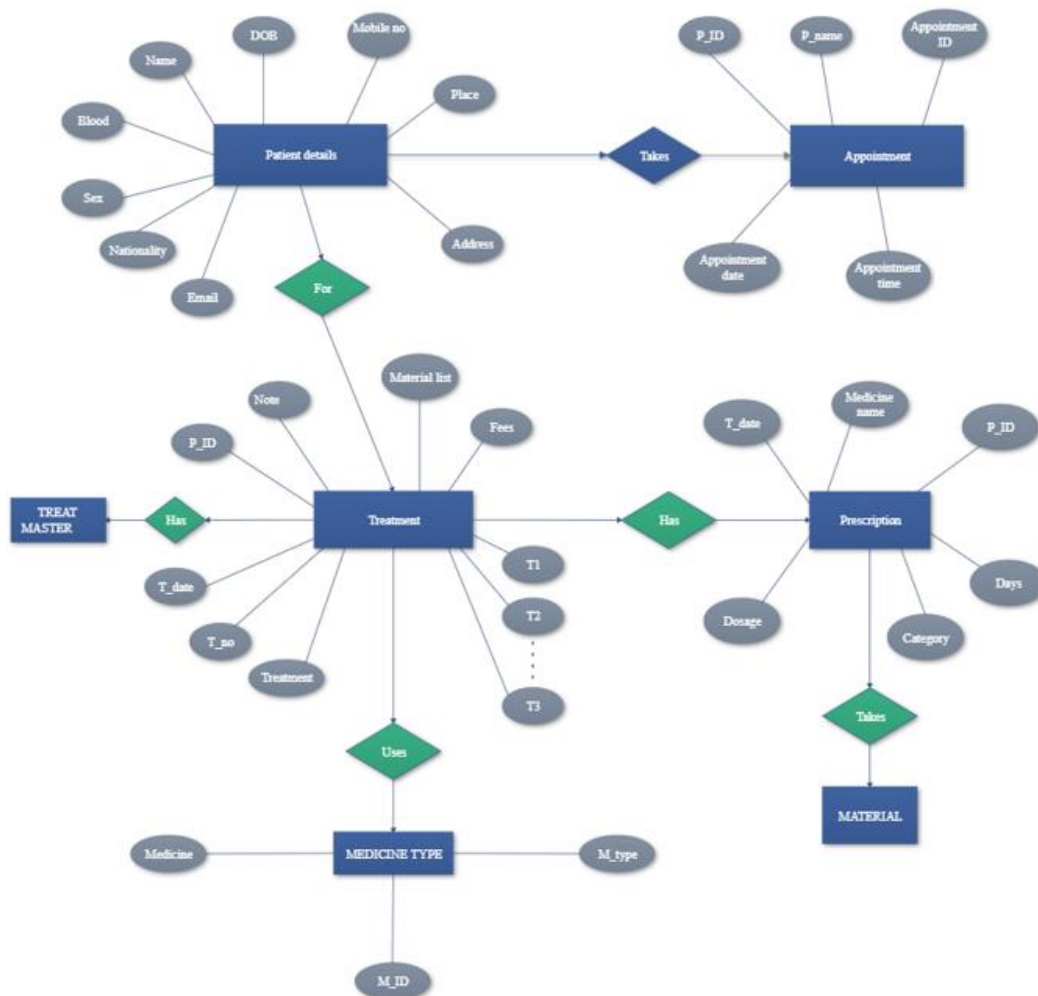
We strive to continuously innovate and evolve our software to keep pace with the rapidly changing healthcare industry. Our team is dedicated to providing exceptional customer support and ensuring a seamless implementation process for our clients. Together, we can transform healthcare management and improve patient outcomes.

Hospital Management System is designed for multi-speciality hospitals, to cover a wide range of hospital administration and management processes. It is an integrated end-to-end Hospital Management System that provides relevant information across the hospital to support effective decision making for patient care, hospital administration and critical financial accounting, in a seamless flow. Hospital Management System is a software product suite designed to improve the quality and management of hospital management in the areas of clinical process analysis and activity-based costing. Hospital Management System enables you to develop your organization and improve its effectiveness and quality of work. Managing the key processes efficiently is critical to the success of the hospital helps you manage your processes.

Technical Architecture:



ER Diagram:



The Entity-Relationship (ER) diagram for a hospital management system visually represents how patients register for an appointment easily and shows the relation between patient, doctor, appointments, medical record, prescriptions. The main entities of the hospital management system are patients and doctors.

Key Features:

Patient management

It is used to control patient flow. It can be used to register them, get the data of the patients' health condition, view the treatment and check the medical history and reports. Appointment module in hospital management arranges the schedule of doctors due to the patients' application. It helps to organize the availability of medical specialists at any convenient time. Some hospital can even offer remote visits when you need immediate assistance.

Facility management

The facility management module is responsible for tracking and maintaining the room availability, the occupancy status as well as various kinds of administrative documentation.

Appointment Management:

Integrate appointment widgets in your online Hospital management system. Besides, enable easy scheduling for patients with the hospital website.

Billing Management

It is difficult to maintain and track separate bills for treatments, testing labs, and diagnostics. Hence, integrate the billing system in your hospital. Besides, get customized alerts for payment dues too.

Prescription Management

Make a note of commonly used medicines. Further, keep track of the availability of frequently used medicines in the pharmacy. On the other hand, switch to digital prescriptions to avoid wrong medication.

Staff management

Staff management module provides the human resources administration. It updates the job description of employees, updates the hospital structure, tracks the recruiting records.

Accounting

Accounting module organizes the financial affairs of both customers and the medical institution. It stores and presents all the patient payment details, hospital financial records on expenses and overall profit.

PRE-REQUISITES:

To develop a MEAN-stack Hospital Management System using AngularJS, Node JS, Express JS and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: <https://nodejs.org/en/download/>
- Installation instructions: <https://nodejs.org/en/download/package-manager/>

MongoDB: Set up a MongoDB database to messages, posts, likes, followers and other information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command: **npm install express**

Angular: Angular is a JavaScript framework for building client-side applications. Install Angular CLI (Command Line Interface) globally to create and manage your Angular project.

Install Angular CLI:

- Angular provides a command-line interface (CLI) tool that helps with project setup and development.
- Install the Angular CLI globally by running the following command: **npm install -g @angular/cli**

Verify the Angular CLI installation:

- Run the following command to verify that the Angular CLI is installed correctly: **ng version**

You should see the version of the Angular CLI printed in the terminal if the installation was successful.

Create a new Angular project:

- Choose or create a directory where you want to set up your Angular project.
- Open your terminal or command prompt.
- Navigate to the selected directory using the `cd` command.
- Create a new Angular project by running the following command: `ng new client` Wait for the project to be created:
- The Angular CLI will generate the basic project structure and install the necessary dependencies

Navigate into the project directory:

- After the project creation is complete, navigate into the project directory by running the following command: `cd client`

Start the development server:

- To launch the development server and see your Angular app in the browser, run the following command: `ng serve / npm start`
- The Angular CLI will compile your app and start the development server.
- Open your web browser and navigate to `http://localhost:4200` to see your Angular app running.

You have successfully set up Angular on your machine and created a new Angular project. You can now start building your app by modifying the generated project files in the `src` directory.

Please note that these instructions provide a basic setup for Angular. You can explore more advanced configurations and features by referring to the official Angular documentation:

<https://angular.io>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

To Connect the Database with Node JS, go through the below provided link:

Link: <https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/>

To run the existing Hospital-Management-System project download from GitHub:

Follow below steps:

1. Clone the Repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the Hospital Management system app.
- Execute the following command to clone the repository:
- git clone: <https://github.com/Yarra-Deepthi/Hospital-Management-System.git>

2. Install Dependencies:

- Navigate into the cloned repository directory:
cd Hospital-Management-System
- Install the required dependencies by running the following command:
npm install

3. Start the Development server:

- To start the development server, execute the following command:
npm run dev or npm run start
- The Hospital-Management-System app will be accessible at `http://localhost:5100` by default. You can change the port configuration in the `.env` file if needed.

4. Access the App:

- Open your web browser and navigate to `http://localhost:5100`.
- You should see the Hospital-Management-System's homepage, indicating that the installation and setup were successful.

Video Tutorial Link to clone the project: -

Link:

Project Repository Link:

<https://github.com/Yarra-Deepthi/Hospital-Management-System.git>

Congratulations! You have successfully installed and set up the Hospital Management System on your local machine. You can now proceed with further customization, development, and testing as needed.

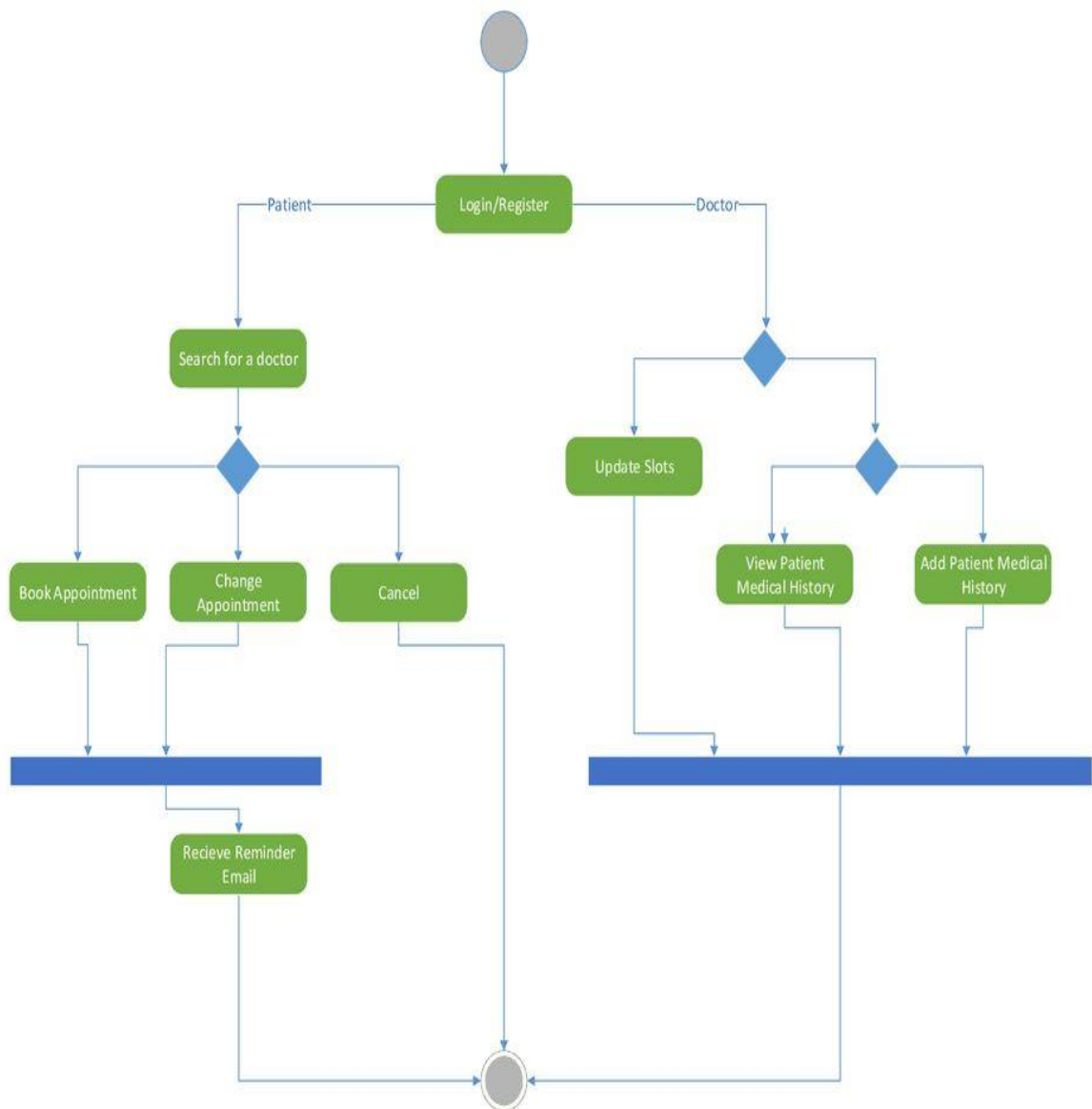
Roles and Responsibilities:

The project has only user role and the responsibilities of the users can be inferred from the API endpoints defined in the code. Here is a summary:

User:

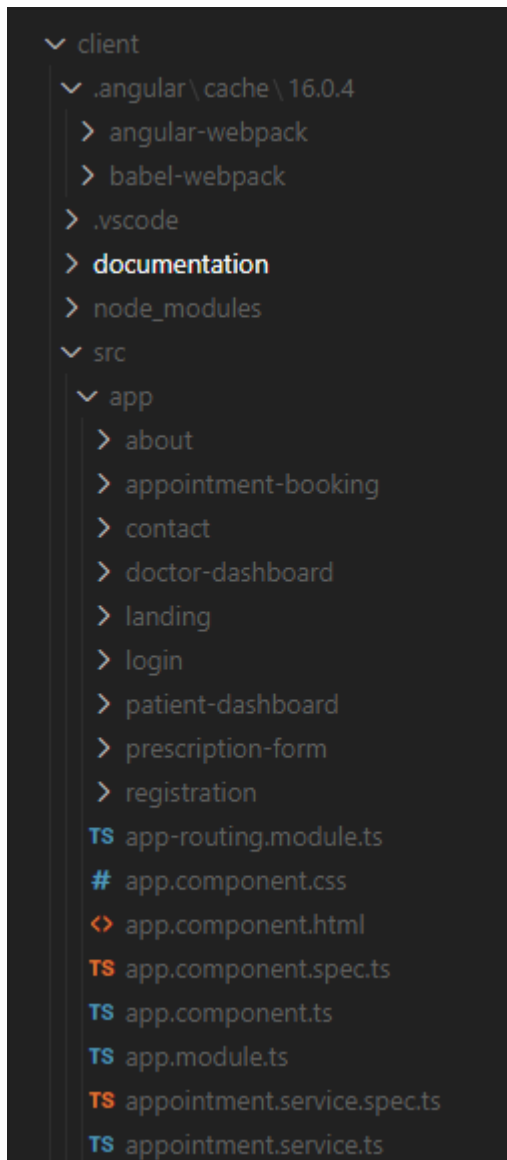
1. Patient has to login and register into the portal using their email and password.
2. Managing appointments for patients to see doctors or specialists efficiently.
3. Managing and generating prescriptions for patients.
4. Handling billing and invoicing for medical services.
5. Maintaining EHR of patients, treatments ,and test results

Admin and User Flow:



The project flow for a hospital management system involves login/register, search for a doctor, add patient medical history, book appointment, change appointment, cancel, add patient medical history .

Project Structure:



This structure assumes an Angular app and follows a modular approach. Here's a brief explanation of the main directories and files:

- `src/app`: contains components related to the hospital management system, such as about, appointment booking, contact, doctor-dashboard, landing, login, patient-dashboard, prescription-form, registration.
- `src/db/connect.js`: contains the connection url of the angular and it performs the routing operations through mongo db connection.
- `src/app.js`: Defines the routing configuration for the app, specifying which components should be loaded for each route.

Project Flow:

Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

- Node.js.
- MongoDB.
- Angular CLI.

2. Create project folders and files:

- Client folders.
- Server folders

Milestone 2: Backend Development:

Setup express server:

- Install express.
- Create app.js file.
- Define API's

Configure MongoDB:

- Install Mongoose.
- Create database connection.
- Create Models.

Implement API end points:

- Implement CRUD operations.
- Test API end points.

Milestone 3: Web Development:

i. Setup Angular Application:

- Create Angular application using angular CLI.
- Configure Routing.
- Install required libraries.

ii. Design UI components:

- Create Components.
- Implement layout and styling.
- Add navigation.

iii. Implement frontend logic:

- Integration with API end points.
- Implement data binding.

Create database in cloud video link:-

To Setup the frontend development and to connect node.js with MongoDB Database Go through this video link: -

https://drive.google.com/file/d/1b5bMvnqmASXLnSZ74B2t3EzNjuWHj63g/view?usp=drive_link

Backend:

1) Set up project structure:

- Create a new directory for your project and set up a package.json file using npm init command.
- Install necessary dependencies such as Express.js, Mongoose, and other required packages.

2) Database Configuration:

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas.
- Create a database and define the necessary collections for doctors, appointments, prescriptions, patients and other relevant data.

3) Create Express.js Server:

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

4) Define API Routes:

- Create separate route files for different API functionalities such as users, patients, appointments, prescriptions and payments.
- Define the necessary routes for handling user registration and login etc.
- Implement route handlers using Express.js to handle requests and interact with the database.

5) Implement Data Models:

- Define Mongoose schemas for the different data entities like patients, users, and appointments.
- Create corresponding Mongoose models to interact with the MongoDB database.
- Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

API Design and Development:

- Identify the necessary functionality and data required by the frontend.
- Design a set of RESTful APIs using a framework like Express.js or Django REST Framework.
- Define API endpoints for appointments, patients, payments, prescriptions etc.
- Implement the API routes, controllers, and data models to handle the corresponding operations.
- Ensure that the APIs follow best practices, are secure, and provide appropriate responses.

User Management and Authentication:

- Implement user registration and login functionality.
- Choose an authentication mechanism like session-based authentication or token-based authentication (e.g., JWT).
- Store user credentials securely.
- Implement middleware to authenticate API requests and authorize access to protected routes.

Access Settings:

- After you can book an appointments
- Then a list of components home, about contacts, register, login etc.

Schema use case:

1.Users:

- Schema : user Schema
- Model: 'user'
- Purpose: Represents the schema and model for user data, including information like name, email, password, and other relevant details. It is used for user registration, authentication, and managing user-related functionalities.

2.Appointments :

- Schema : appointment Schema
- Model : 'appointment'
- Purpose: Represents the schema and model to book and appointment or cancel the appointments or get an appointments

3.Patient:

- Schema : patient Schema
- Model : 'patient'
- Purpose: Represents the schema and model to get the patients or add ,update ,delete the patients

4.prescriptions:

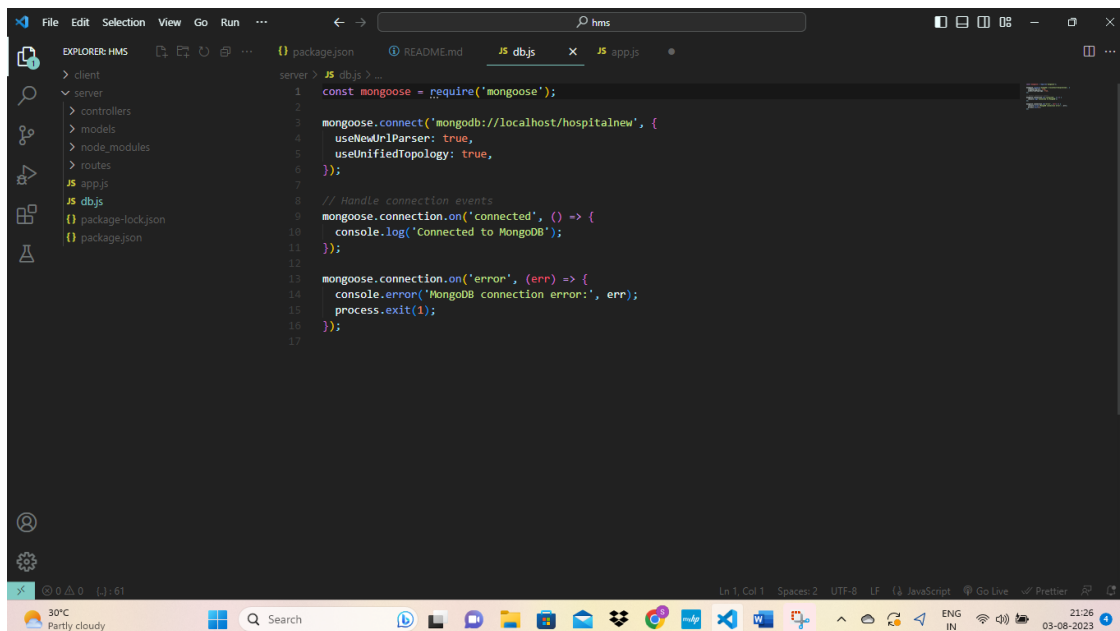
- Schema: prescription Schema
- Model: 'prescription'
- Purpose: Represents the schema and model for get all the prescriptions or get prescriptions by id, name.

5.Payments:

- Schema: payment Schema
- Model: 'payment'
- Purpose: Represents the schema and model to get all the payments, save the payments or we can create the new payment.

Backend Explanation with code snippets:

Database Connection:

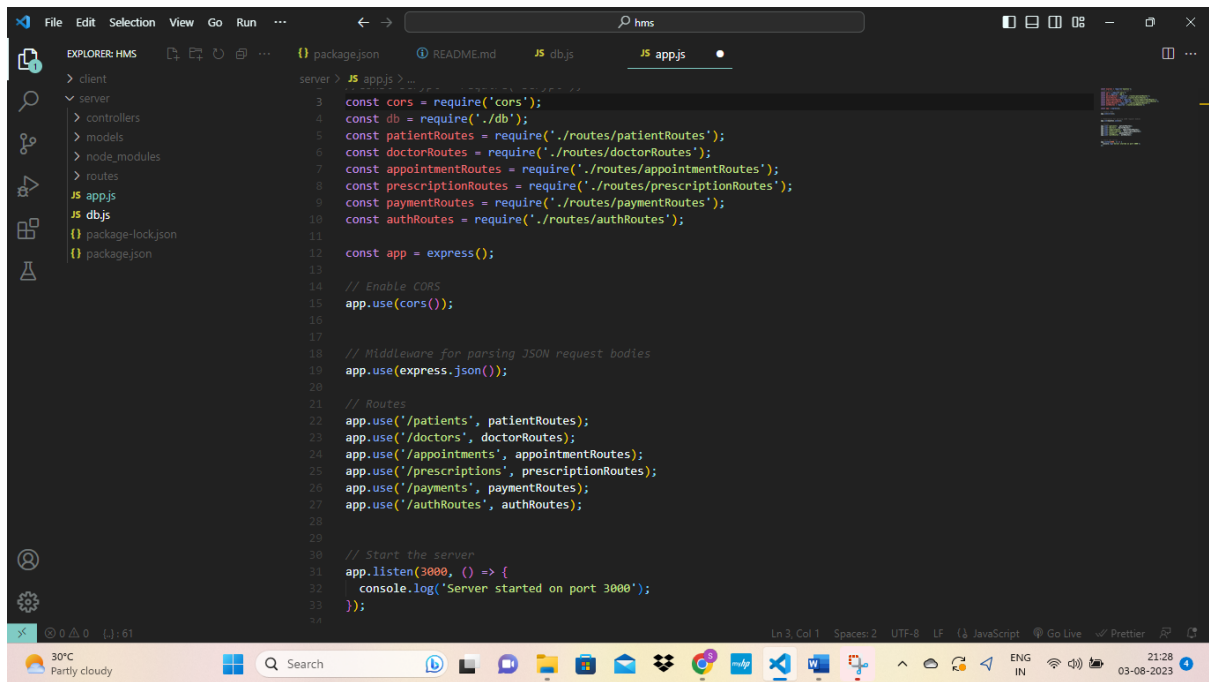
A screenshot of a Visual Studio Code editor window. The Explorer sidebar on the left shows a project structure with folders like 'client', 'server', 'controllers', 'models', 'node_modules', 'routes', and files like 'app.js', 'db.js', 'package-lock.json', and 'package.json'. The main editor area is open to 'db.js' and contains the following JavaScript code:

```
1 const mongoose = require('mongoose');
2
3 mongoose.connect('mongodb://localhost/hospitalnew', {
4   useNewUrlParser: true,
5   useUnifiedTopology: true,
6 });
7
8 // Handle connection events
9 mongoose.connection.on('connected', () => {
10   console.log('Connected to MongoDB');
11 });
12
13 mongoose.connection.on('error', (err) => {
14   console.error('MongoDB connection error:', err);
15   process.exit(1);
16 });
17
```

The status bar at the bottom indicates 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'LF', 'JavaScript', 'Go Live', 'Prettier', and the date '03-08-2023'.

API's for Authentication:

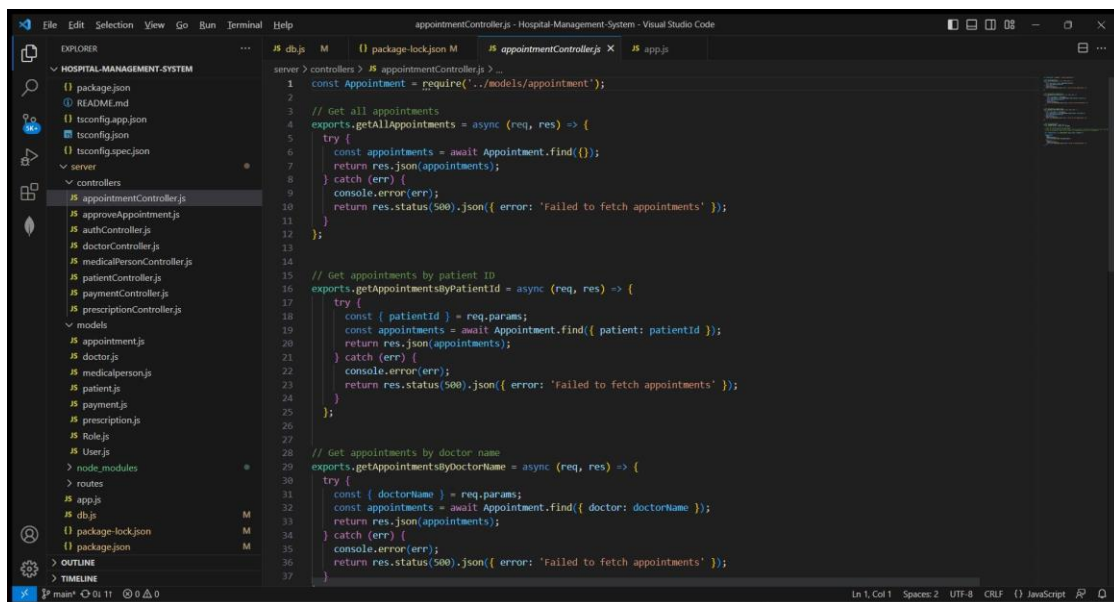
the admin Authentication and user Authentication API enables users to create and connect to the database.



```
1 // Importing modules
2 const express = require('express');
3 const cors = require('cors');
4 const db = require('./db');
5 const patientRoutes = require('./routes/patientRoutes');
6 const doctorRoutes = require('./routes/doctorRoutes');
7 const appointmentRoutes = require('./routes/appointmentRoutes');
8 const prescriptionRoutes = require('./routes/prescriptionRoutes');
9 const paymentRoutes = require('./routes/paymentRoutes');
10 const authRoutes = require('./routes/authRoutes');
11
12 const app = express();
13
14 // Enable CORS
15 app.use(cors());
16
17 // Middleware for parsing JSON request bodies
18 app.use(express.json());
19
20 // Routes
21 app.use('/patients', patientRoutes);
22 app.use('/doctors', doctorRoutes);
23 app.use('/appointments', appointmentRoutes);
24 app.use('/prescriptions', prescriptionRoutes);
25 app.use('/payments', paymentRoutes);
26 app.use('/authRoutes', authRoutes);
27
28 // Start the server
29 app.listen(3000, () => {
30   console.log('Server started on port 3000');
31 });
```

API's for appointments:

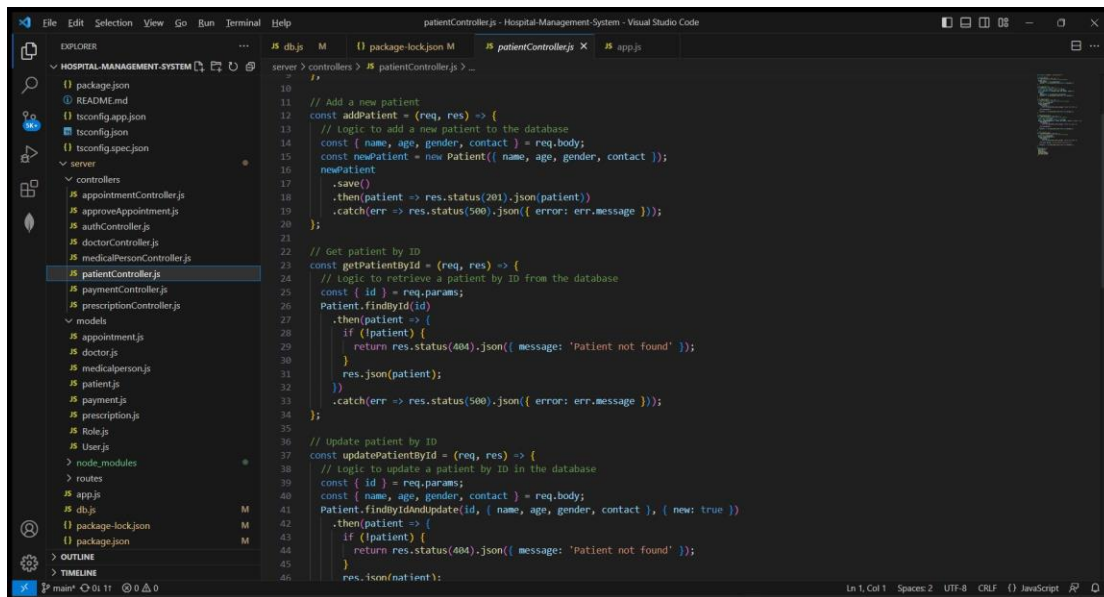
This API is used to book an appointments .so we can book an appointment if the appointment is not booked it shows failed to book appointments. And we can also get an appointment by patient id, doctor name etc.



```
1 // Importing modules
2 const Appointment = require('../models/appointment');
3
4 // Get all appointments
5 exports.getAllAppointments = async (req, res) => {
6   try {
7     const appointments = await Appointment.find({});
8     return res.json(appointments);
9   } catch (err) {
10     console.error(err);
11     return res.status(500).json({ error: 'Failed to fetch appointments' });
12   }
13 };
14
15 // Get appointments by patient ID
16 exports.getAppointmentsByPatientId = async (req, res) => {
17   try {
18     const { patientId } = req.params;
19     const appointments = await Appointment.find({ patient: patientId });
20     return res.json(appointments);
21   } catch (err) {
22     console.error(err);
23     return res.status(500).json({ error: 'Failed to fetch appointments' });
24   }
25 };
26
27 // Get appointments by doctor name
28 exports.getAppointmentsByDoctorName = async (req, res) => {
29   try {
30     const { doctorName } = req.params;
31     const appointments = await Appointment.find({ doctor: doctorName });
32     return res.json(appointments);
33   } catch (err) {
34     console.error(err);
35     return res.status(500).json({ error: 'Failed to fetch appointments' });
36   }
37 };
```

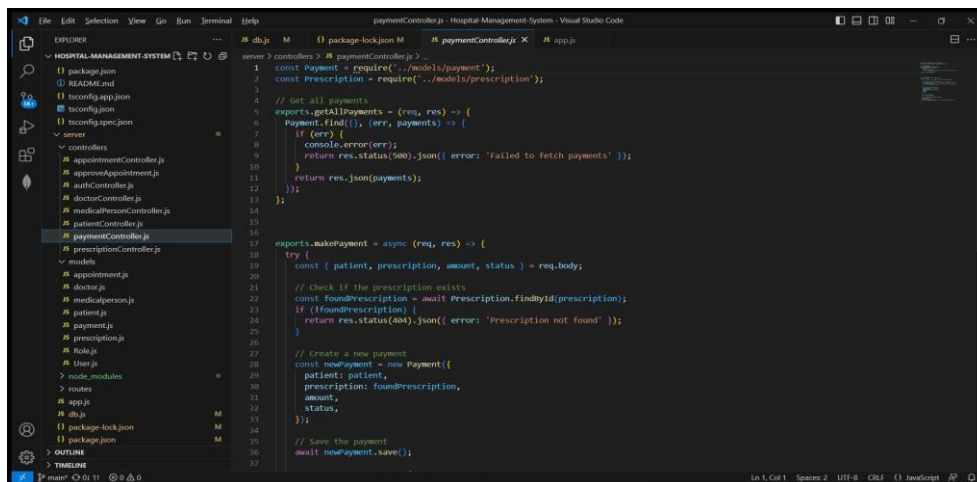
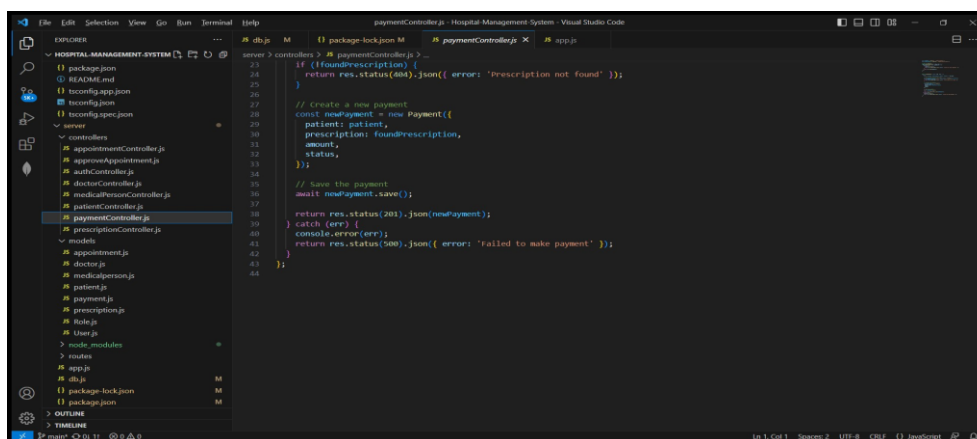
APIs for patients:

This API is used to get all the patients,also we can get patients by id and also we can update patients or delete patients.

A screenshot of the Visual Studio Code editor showing the file explorer on the left with the 'controllers' folder expanded. The 'patientController.js' file is selected and its code is displayed in the main editor. The code includes functions for adding a new patient, getting a patient by ID, and updating a patient by ID. The status bar at the bottom indicates 'Ln 1, Col 1' and 'Spaces: 2'.

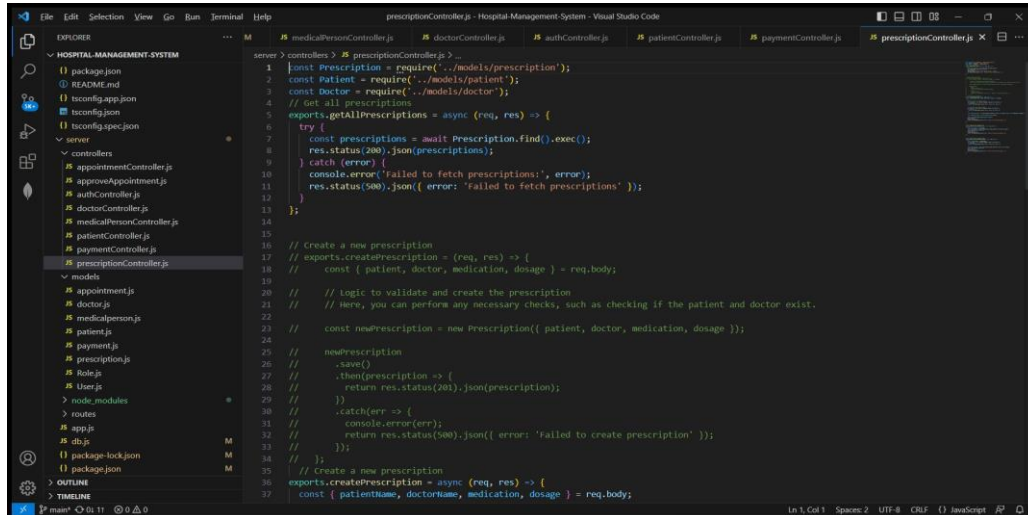
API for payments:

This API is used for to get all payments and also used to save the payments and create new payment.

A screenshot of the Visual Studio Code editor showing the file explorer on the left with the 'controllers' folder expanded. The 'paymentController.js' file is selected and its code is displayed in the main editor. The code includes functions for getting all payments and making a payment. The status bar at the bottom indicates 'Ln 1, Col 1' and 'Spaces: 2'.A screenshot of the Visual Studio Code editor showing the file explorer on the left with the 'controllers' folder expanded. The 'paymentController.js' file is selected and its code is displayed in the main editor. The code includes functions for getting all payments and making a payment. The status bar at the bottom indicates 'Ln 1, Col 1' and 'Spaces: 2'.

APIs for prescriptions:

This API is used for prescriptions, get prescriptions by patient id, name. if its not found then it shows failed to fetch prescriptions.



```
server > controllers > prescriptionController.js
1 const prescription = require('../models/prescription');
2 const patient = require('../models/patient');
3 const doctor = require('../models/doctor');
4 // Get all prescriptions
5 exports.getAllPrescriptions = async (req, res) => {
6   try {
7     const prescriptions = await Prescription.find().exec();
8     res.status(200).json(prescriptions);
9   } catch (error) {
10    console.error('Failed to fetch prescriptions', error);
11    res.status(500).json({ error: 'Failed to fetch prescriptions' });
12   }
13 }
14
15 // Create a new prescription
16 exports.createPrescription = (req, res) => {
17   const { patient, doctor, medication, dosage } = req.body;
18   // Logic to validate and create the prescription
19   // Here, you can perform any necessary checks, such as checking if the patient and doctor exist.
20   const newPrescription = new Prescription({ patient, doctor, medication, dosage });
21   newPrescription.save()
22     .then(prescription => {
23       return res.status(201).json(prescription);
24     })
25     .catch(err => {
26       console.error(err);
27       return res.status(500).json({ error: 'Failed to create prescription' });
28     });
29 }
30
31 // create a new prescription
32 exports.createPrescription = async (req, res) => {
33   const { patientName, doctorName, medication, dosage } = req.body;
```

Frontend:

User Interface(UI) Design:

- Create a visually appealing and consistent design using modern design principles.
- Pay attention to typography, colour schemes, spacing, and visual hierarchy.
- Use responsive design techniques to ensure the app looks great on different devices.

Responsive Design:

- Utilize CSS media queries and responsive design frameworks like Bootstrap or Tailwind CSS to create a responsive layout.
- Test your app on various devices and screen sizes to ensure a seamless user experience.

User Authentication and Account Management:

- Design and implement a user registration and login system.
- Implement authentication guards to restrict access to certain pages or features.

Home component:

- Design and develop a home page component to allow users to book an appointment.
- User can book an appointment by register/login.

About component:

- It tells about the use of this app and uses of this app .
- You can find about the information like appointments, medical record, inventory etc.

Contact:

- If you have any questions, feedback, or inquiries about our Hospital Management App, you can reach out using the contact details .

Register/Login:

- User can able register/login to continue into the app like booking an appointments, payments, prescriptions etc.