Sunday, 18 August 2024

Final Project Details

---

1. Can you briefly explain your project?

- The project is a Hospital Management System designed for a small firm. It has three main roles: Admin, Doctor, and Patient. The Admin can manage doctors, departments, and monitor the overall platform, including appointments and doctor availability. Doctors can manage their schedules, interact with patients via chat and video calls, and send prescriptions. Patients can search for doctors, book appointments, and manage their profiles. The system uses role-based authentication to ensure secure access.

2. What technologies did you use in this project?

- The project was built using the MERN stack. For the backend, we used Node.js, Express.js, and MongoDB for data storage. AWS S3 was used for storing files, and Multer for handling file uploads. For real-time communication, we used Socket.io and Zego Cloud for video calls. On the frontend, we used React.js, Redux, and Context API for state management. Chart.js was used for displaying charts.

3. Why did you choose these technologies?

- We chose the MERN stack because it allows for a full JavaScript-based development environment, making it easier to manage and integrate both the frontend and backend. MongoDB is a flexible NoSQL database that fits well with the dynamic nature of the application. React.js offers efficient rendering and a great user experience, while Redux and Context API help in managing the application state effectively. AWS S3 is reliable for file storage, and Socket.io, along with Zego Cloud, provides robust real-time communication features.

4. What were some challenges you faced during development?

- Some challenges included implementing role-based authentication to ensure secure access based on user roles, handling real-time communication for chat and video calls, and managing file uploads to AWS S3. Additionally, integrating various services like Zego Cloud for video calls required careful coordination to ensure a seamless user experience.

5. How does role-based authentication work in your system?

- Role-based authentication ensures that users can only access features that are relevant to their role. For example, only Admins can manage doctors and departments, while Doctors can only access patient information and manage appointments. Patients have access to search and book appointments. This is implemented both on the backend, using middleware in Node.js, and on the frontend, by conditionally rendering components based on the user's role.

6. How does the chat and video call feature work?

- The chat feature is implemented using Socket.io, which allows real-time messaging between doctors and patients. When a patient books a slot, they can initiate a video call with the doctor at the scheduled time. Zego Cloud is integrated for video calling, providing a reliable and high-quality video communication service. The chat feature is embedded within the video call interface for convenience.

7. What is the purpose of using AWS S3 and Multer?

- AWS S3 is used for securely storing files, such as profile pictures or documents related to the patients. Multer is a middleware used in Node.js to handle file uploads, ensuring that the files are processed and sent to AWS S3 for storage. This setup allows for efficient and scalable file management.

8. How does the appointment booking and scheduling work?

- Patients can search for doctors by name or department and view available slots. They can then book a slot, which is immediately reflected on the doctor's schedule. The doctor can see all upcoming appointments in their dashboard and can initiate a video call at the scheduled time. The scheduling and booking logic is handled in the backend using MongoDB to store and manage the appointments.

9. How do you manage state in your frontend application?

- State management is handled using both Redux and Context API. Redux is used for managing global states that need to be accessed across various components, such as user authentication status or appointment data. Context API is used for smaller, localized state management, allowing components to share data without prop drilling.

Make sure to add questions and problems that you felt during project.