AmazeCare

Problem statement:

In this case study, we will design and develop a fullstack healthcare application called " AmazeCare" which aims to streamline patient management and improve healthcare services in a medical facility. The application will facilitate the interaction between patients, doctors, and administrative staff, ensuring efficient appointment scheduling, medical record management, and communication.

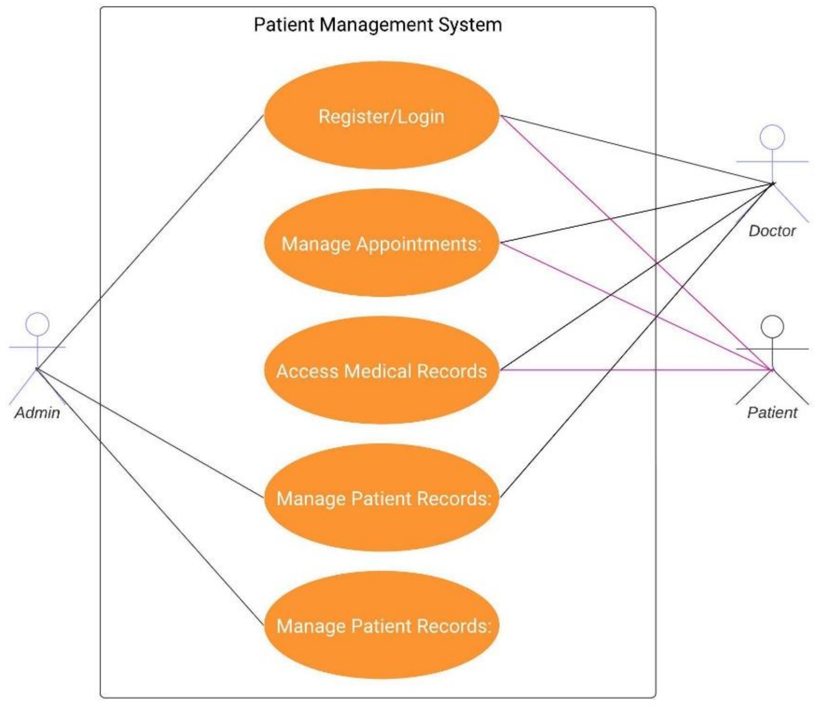
Scope

- **Efficient Appointment Booking:** Enable patients to schedule appointments online, reducing waiting times and improving overall patient experience.
- **Comprehensive Medical Records:** Store and manage patient medical records electronically for easy access by healthcare providers.
- **Real-time Communication:** Allow patients and doctors to communicate securely through the platform, enabling follow-up discussions and quick responses to queries.
- **Administrative Automation:** Simplify administrative tasks such as handling medical records, staff records.

Technologies:

- Frontend: React.js / Angular Js.
- Backend: Java, Spring Boot/C#, .Net / Python Djnago for API development.
- Database: MySql / Sql Server.
- Authentication: JSON Web Tokens (JWT) for secure user authentication.

Use case Diagram



Use Cases:

Actor: Patient

- Use Case: Register as a new patient
- Use Case: Log in as an existing patient
- Use Case: Update personal information
- Use Case: Schedule Appointment
- Use Case: View Medical History
- Use Case: View Appointments
- Use Case: Cancel Appointment

Actor: Doctor

- Use Case: Log In as an existing Doctor
- Use Case: View Appointments
- Use Case: Conduct Consultation
- Use Case: Update Medical Records
- Use Case: Prescribe Medications

Actor: Administrator

- Use Case: Log In
- Use Case: Manage Appointments
- Use Case: Manage Doctors
- Use Case: Generate Reports(As simple list view)

System: Security and Authentication

- Use Case: Authenticate User

System: Database Management

- Use Case: Store Patient Information
- Use Case: Store Doctor Information
- Use Case: Store Appointment Data
- Use Case: Store Medical Records

Associations:

- Patient initiates the Register Account and Log In use cases.
- Patient uses View Patient Profile, Manage Appointments, View Medical Records, and Communicate with Doctor.
- Doctor uses View Patient Profile, Manage Appointments, View Medical Records, and Communicate with Doctor.
- Administrator uses Manage Doctor Schedule, Handle Billing, and Generate Reports.
- Use cases can have associations with multiple actors.

Development Process:

1. **Patient Registration and Appointment Booking:**
    - Patients can create accounts, providing personal details and medical history.
    - Patients can search for available doctors, view their profiles, and schedule appointments.
2. **Doctor's Dashboard:**
    - Doctors can view their appointment schedule, patient details, and medical history.
        1. doctor can list their upcoming and completed appointment with following details in table.
            1. name of patient, contact no, symptoms.
        2. upcoming appointments should be highlighted in green color and should add the consulting details with prescription as discussed below.
        3. upcoming appointments have an option to reject/cancel appointments.
        4. Doctor can view completed appointment with prescription.
    - Doctor should capture from patient as consulting details in corresponding appointment:
        1. Current Symptoms and Concerns:
        2. Physical Examination: Observe and assess the patient's vital signs, appearance, and any visible symptoms.
        3. Treatment Plan: Recommend any medical test to be taken.
        4. Recommend tests: refer the excel file.
        5. Prescription: include medicine name with 0-0-1 AF(after food)/BF(before food)
    - Doctor can update patient prescribe medications and recommend tests. refer excel for list of medicine and medical test.
3. **Patient Dashboard:**
    - Patient can make new appointments with doctors and providing following information.
        1. Personal Information:
        2. Full Name
        3. Date of Birth
        4. Gender
        5. Contact Information (Mobile Number)
        6. Brief description of symptoms or health concerns

7. Nature of the visit (e.g., general check-up, specific medical issue)

8. Preferred date and time

- Patients can view upcoming appointments with doctor name and date of appointment and can reschedule appointments. all the upcoming appointments should display in table.
- Patient can access completed consulting details with doctor (completed appointment) in table with following details date of appointment, treatment or diagnosis and doctor name.

4. **Admin Dashboard:**

- Admin can manage doctors like add, update delete the doctor details.

  To add new doctor details, capture details like:

1.	Name	POORNIMA C
2.	Specialty	Obstetrics and Gynecology
3.	Experience	18 Years
4.	Qualification	MS (OG)
5.	Designation	Associate Professor/Consultant

- Admin can manage Patient details like add, update delete the patient details.
- manage Appointment details like reschedule and view the Appointment details.

5. **Appointment Management:**

- Patients can search for available doctors based on specialization, and availability. refer end of document for list of specialization.
- Patients can request appointments with doctors. refer the below link.
- Doctors receive appointment requests and can confirm, reschedule, or reject appointments.
- reference link https://www.psghospitals.com/find-a-doctor/

6. Security and Compliance:

- User authentication and authorization are enforced to ensure data privacy.

1. JWT Authentication:

JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

- User Login: Upon successful login (using valid credentials), generate a JWT token on the server.

- Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
- Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.
- Token Transmission: Send the signed token back to the client as a response to the login request.
- Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

2. JWT Authorization:

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

- Protected Routes: Define routes that require authentication and authorization.
- Token Verification:
    1. Extract the token from the request header.
    2. Verify the token's signature using the server's secret key.
- Payload Verification:
    1. Decode the token and extract user information.
    2. Check user roles or permissions to determine access rights.
- Access Control: Grant or deny access based on the user's roles and permissions.

Logout:

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

Project Development Guidelines

The project to be developed based on the below design considerations.

1	**Backend Development**	• Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services • Use Java/C# latest features • Use ORM with database • perform backend data validation • Use Swagger to invoke APIs • Implement API Versioning • Implement security to allow/disallow CRUD operations

		• Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. • Any error message or exception should be logged and should be user-readable (not technical) • Database connections and web service URLs should be configurable • Implement Unit Test Project for testing the API • Implement JWT for Security • Implement Logging • Follow Coding Standards with proper project structure.

Frontend Constraints

1.	**Layout and Structure**	Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes.
2	**Visual Elements**	**Logo:** Place your application's logo at the top of the page to establish brand identity.
		Form Fields: Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field.
		Buttons: Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable).
		Error Messages: Provide clear error messages for incorrect login attempts or registration errors.
		Background Image: Consider using a relevant background image to add visual appeal.
		Hover Effects: Change the appearance of buttons and links when users hover over them.
		Focus Styles: Apply focus styles to form fields when they are selected
3.	**Color Scheme and Typography**	Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for readability. Select a legible and consistent typography for headings and body text.
4.	**Registration Page, Doctor Consultation Page, Patient Appointment Booking Page, Add New Doctor Admin update details page**	**Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users.
		Validation: Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors. **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission.
	Registration Page	**Password Strength:** Provide real-time feedback on password strength using indicators or text.

		Password Requirements: Clearly indicate password requirements (e.g., minimum length, special characters) to help users create strong passwords.
		Registration Success: Upon successful registration, redirect users to the login page.
5.	**Login Page**	**Form Fields:** Provide fields for users to enter their email and password.
		Password Recovery: Include a "Forgot Password?" link that allows users to reset their password.
6.	**Common to React/Angular**	• Use Angular/React to develop the UI. • Implement Forms, databinding, validations, error message in required pages. • Implement Routing and navigations. • Use JavaScript to enhance functionalities. • Implement External and Custom JavaScript files. • Implement Typescript for Functions Operators. • Any error message or exception should be logged and should be user-readable (and not technical). • Follow coding standards. • Follow Standard project structure. • Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets.

Good to have implementation features

- Generate a SonarQube report and fix the required vulnerability.
- Use the Moq framework as applicable.
- Create a Docker image for the frontend and backend of the application .
- Implement OAuth Security.
- Implement design patterns.
- Deploy the docker image in AWS EC2 or Azure VM.
- Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies.
- Use AWS RDS or Azure SQL DB to store the data.