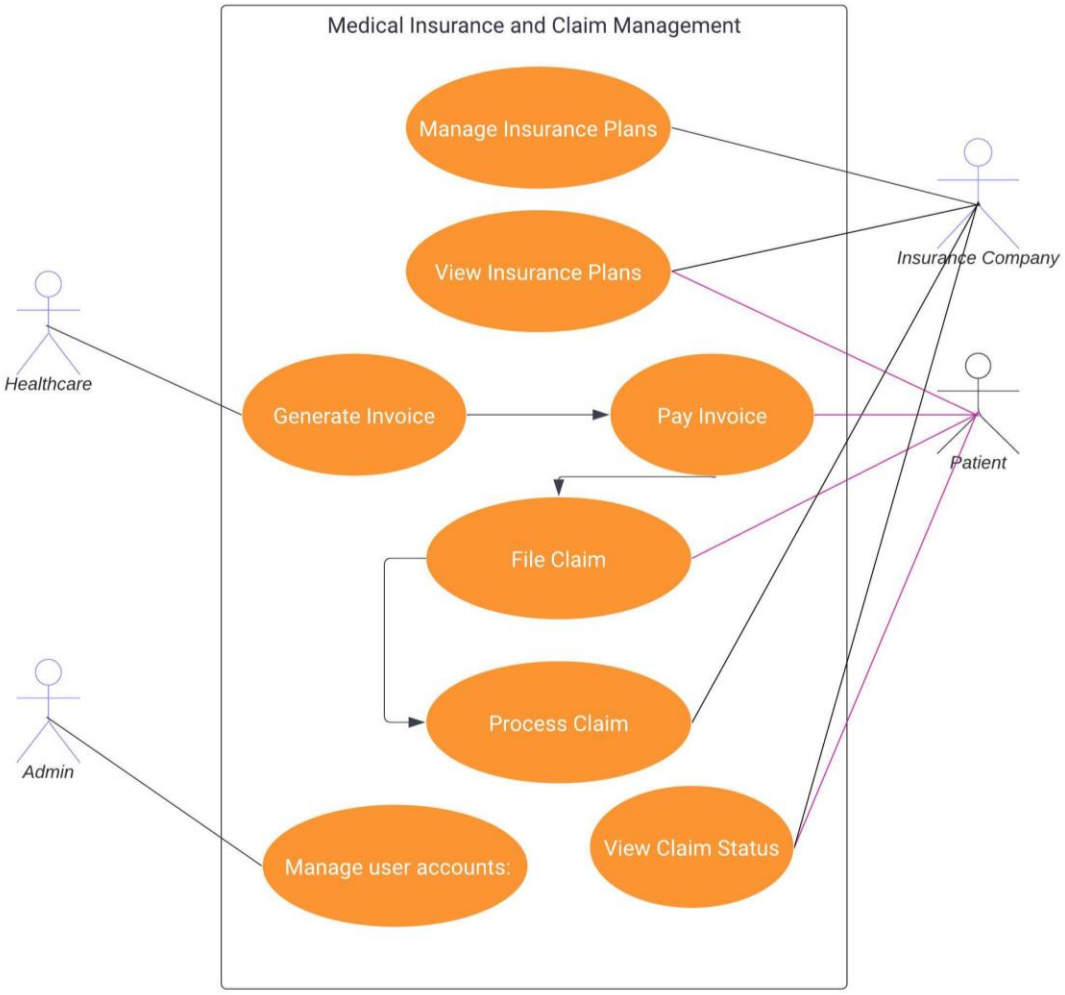
Medical Billing and Claims Management System

Problem Statement: The Medical Billing and Claims Management System is a comprehensive application designed to streamline the complex process of medical billing, claims submission, and reimbursement for healthcare providers, insurance companies, and patients. The system aims to reduce administrative burdens, improve accuracy, and enhance communication between stakeholders.

Scope:

1. **Efficient Claims Submission:** Develop a system for healthcare providers to submit accurate and timely insurance claims.
2. **Claims Processing:** Enable insurance companies to process claims, validate coverage, and reimburse healthcare providers.
3. **Patient Invoicing:** Generate and manage invoices for patients based on their medical services and insurance coverage.

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: view all insurance plan details.

Use Case: Select an insurance plan.

Use Case: View invoice.

Use Case: Submit a claim.

Actor - Healthcare Provider:

Use Case: Log in as a healthcare provider.

Use Case: Generate an invoice.

Use Case: Notify patient of the invoice.

Actor - Insurance Company:

Use Case: Review and process claims.

Use Case: Approve claims.

Use Case: Process claim payment.

Use Case: View patient claim history.

Actor - Administrator:

Use Case: Log in as an administrator.

Use Case: View system dashboard(claims, payments)

Use Case: Manage user accounts.

Development Process:

1. Patient Enrollment and Insurance Management

1. The patient accesses the application, registers, and creates a profile.
    - Personal Information:
    - Full Name
    - Date of Birth
    - Gender
    - Contact Information (Mobile Number)
    - Brief description of symptoms, Treatment
2. The patient can search available insurance plans.
    - Refer policy bazaar for more plans. https://health.policybazaar.com/
3. The patient selects a suitable insurance plan and updates their profile.
4. The patient can raise the request to generate invoice for claim.
5. On successful invoice generation patient can raise the request for claim.

2. Healthcare Service and Billing

1. The healthcare provider logs into the system.
2. The provider adds details of services rendered and generates an electronic invoice with following details.
    - Invoice Number: INV2023-001
    - Invoice Date: August 15, 2023
    - Due Date: September 15, 2023
    - Patient ID: PAT123456
    - Patient Name: John Doe
    - Patient Address: 123 Main Street, City, State, ZIP
    - Consultation Fee: 5675
    - Diagnostic Tests Fee: 2000
    - Diagnostic Scan Fee: 6000
    - Prescribed Medications Bill Amount: 13675
    - Tax (8%): for bill amount

- find total due amount = bill amount + tax.
- Fill 0(zero) for fee if not applicable.

3. The patient accesses the invoice simply update status by paid.
4. The healthcare provider submits the claim form from the patient.

3. Claim Submission and Processing

1. The patient submits a claim electronically from their dashboard, attaching relevant medical documents.
    - The patient fills out the necessary claim details, which may include:
        - Patient information (name, date of birth, address, insurance information).
        - Medical service details (diagnosis, treatment, date of service, claim amount, invoice amount by healthcare provider).
2. The insurance company reviews the claim.
3. They review incoming claims, verifying patient insurance plan coverage.
4. Claim can be rejected or approved, if approved the insurance company processes the claim and initiates payment.
5. Insurance company can view patient claim history with invoice amount, claim amount, date claim submission and date claim approved.

4. Administrator Dashboard

1. The administrator logs into the system and accesses the dashboard.
2. The dashboard displays user accounts, claims, payments.
    - history of all claim details (user name, claim amount, invoice amount, status of claim) should be displayed as table with above mentioned details.
3. The administrator manages user accounts.
    - registered user details should be displayed in table.
    - registered health care, insurance company details should be displayed in table.

5. 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.

4. User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
5. Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
6. Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.
7. Token Transmission: Send the signed token back to the client as a response to the login request.
8. 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.

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

Logout:

13. 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 Constraints**	• 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 Development 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, Invoice Generation page, Claim submission 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.