Claim Management System

1.0 Problem statement

The Claim Management System (CMS) application is used to automate the process of managing the activities of the healthcare system, like registering as a member with plan details already purchased, submitting a claim, and updating plan details.

The following section will cover aspects related to Claim Management System.

a) Member Registration
b) Submit Claim
c) Update Plan Details

Scope of the System

The scope of the system is explained through its modules as follows

- Member Login and Registration – used by members to register the details of self-information and health plan details into the system. The system then stores the details of the member in the system along with the health plan details.

  A member should be able to login with a User Id and Password that exists in the database.

  Capture the details like Member Name, Username, Password, Address, State, Country, Email Address, Contact No, DOB, Member ID, Plan ID, Plan Name, Insured amount, Plan start date, Plan end date, etc.

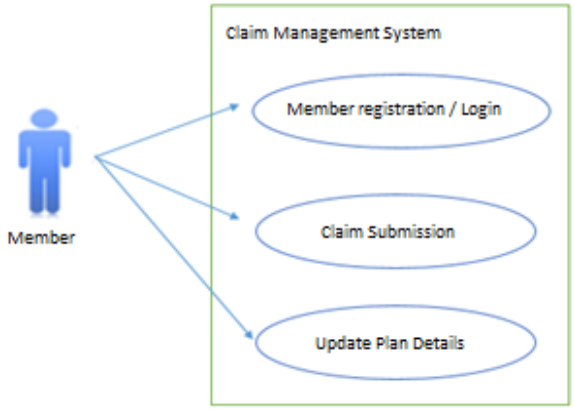
- Submit Claim - will be used by registered members to submit claims into the system. The system validates the claim details with the health plan details of the member and stores the claim details in the system.

  Members should be able to enter new claims into the system, which should be validated against plan details and saved in the database.

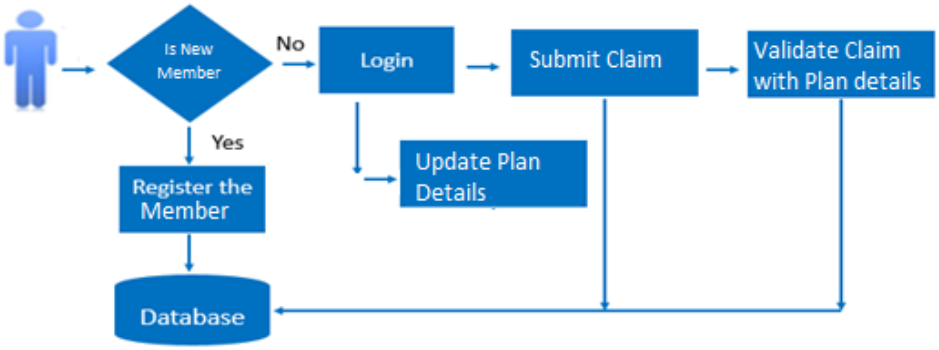
  Capture fields like claim date, claim amount, prescription, or bill details

- Update Plan Details - will be used by registered members to update the plan details in the system. The system should validate the plan details and update them in the system.

2.0 Use Case Diagram



Flow Diagram



3.0 Project Development Guidelines

The project to be developed based on the below design considerations

Backend Development	• Use Rest APIs (Springboot/ASP.Net Core WebAPI to develop the services • Use Java/C# latest features • Use ORM with database • 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 • Follow Coding Standards
Frontend Development	• Use Angular/React to develop the UI • Implement Forms, databinding, validations • 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

4.0 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 Logging
- Implement design patterns
- Use JWT for authentication in SpringBoot/WebApi. A Token must be generated using JWT. Tokens must expire after a definite time interval, and authorization must be handled accordingly based on token expiry
- 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