Policy Management System

1.0 Problem statement

Policy Management System (PMS) is basically a system that helps to maintain the Policies and Payments. It also allows to search a Policy and allow end users to manage policies with ease.

The following section will cover aspects related to the Policy Management System.

a) User Registration.

b) Policy Registration.

c) Search Policies.

Scope of the System

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

- User Registration - User should be able register the details in the system, and it should be saved in the database. Capture the details below

1. First name - Name of the user
2. Last name -Name of the father
3. Date Of Birth- User DOB in DD/MM/YYYY format
4. Address- Address of the User
5. Contact no- Contact number of the Applicant
6. Email Address- Email Address of the User
7. Salary- Salary per month
8. PAN No- Pan Card Number
9. Employer type- Type of the Employer
10. Employer- Name of the Employer if Salaried or Optional

For failed scenarios, the system should display appropriate user-friendly error messages. Use the below table to capture the type of the employer

Income per year (in lakhs)	User type
5	A
10	B
15	C
30	D
More than 30	E

- Policy Registration- Policy Registration is used by the admin to register the policies. The system will generate the policy id and if it is a valid policy the system will store the details in the database.

  Capture fields like

  1. Policy name- Name of the Policy
  2. Start Date- Start date in DD/MM/YYYY format
  3. Duration in years- Duration in years
  4. Company- Company name
  5. Initial deposit- Amount to be deposited at start
  6. Policy Type- Type of the policy
  7. User Types- User types available for policy
  8. Terms per year- Number of Payment Terms per year
  9. Term amount- One term's payment amount
  10. Interest- Interest for the policy

  The system should generate the policy id and calculate the end date based on the duration provided.

  Policy types can be one of the below

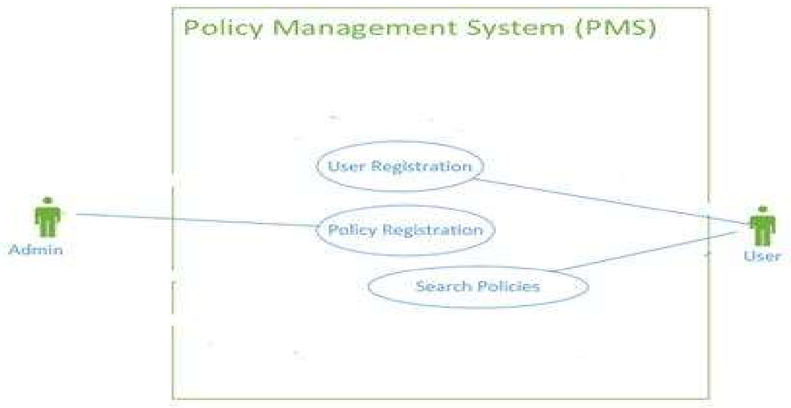
  i. Vehicle Insurance
  ii. Travel Insurance
  iii. Life insurance
  iv. Health Insurance
  v. Child Plans
  vi. Retirement Plans

  Generate the maturity amount as follows.

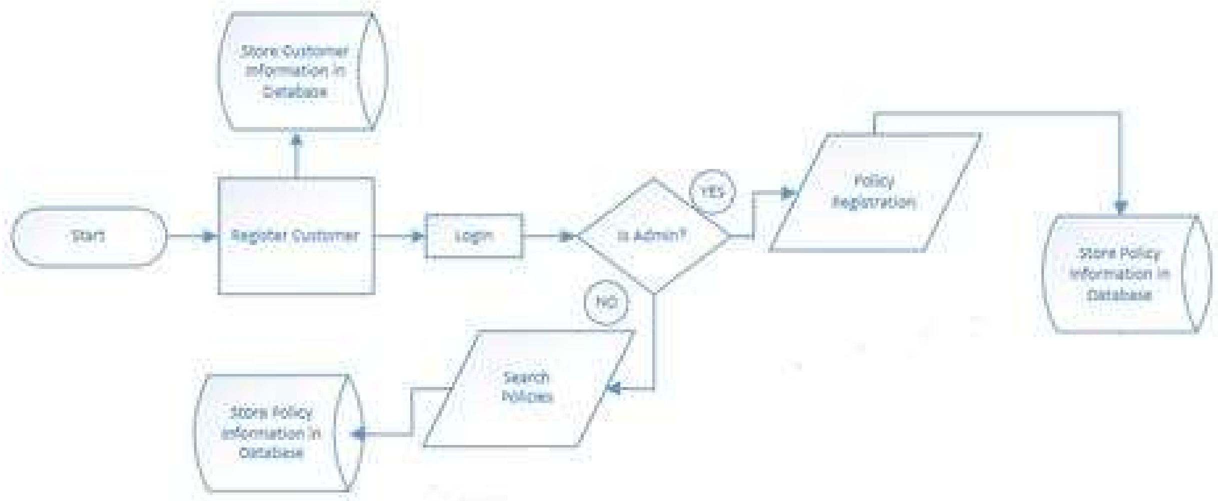
  Maturity amount = (initial deposit) + (duration * term_in_years * term amount) + ((duration * term_in_years * term amount) * (interest /100))

- Search Policies - The system will provide a search engine mechanism to find the policies. This will access the database by getting the details based on the user's search criteria

2.0 Use Case Diagram



Flow Diagram


Store Customer information in Database
Start
Register Customer
Login
Is Admin?
YES
NO
Policy Registration
Store Policy information in Database
Search Policies
Store Policy information in Database

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