Cognizant Academy

Claims Management System

FSE – Business Aligned Project Case Study Specification

Version 1.0

	Prepared By / Last Updated By	Reviewed By	Approved By
Name	Srilakshmi Jayaraman		
Role	Solution Designer		
Signature			
Date			

Table of Contents

1.0 Important Instructions

1. Associate must adhere to the Design Considerations specific to each Technolgy Track
2. Associate must not submit project with compile-time or build-time errors
3. Being a Full-Stack Developer Project, you must focus on ALL layers of the application development
4. Unit Testing is Mandatory, and we expect a code coverage of 100%. Use Mocking Frameworks wherever applicable.
5. All the Microservices, Client Application, DB Scripts, have to be packaged together in a single ZIP file. Associate must submit the solution file in ZIP format only
6. If backend has to be set up manually, appropriate DB scripts have to be provided along with the solution ZIP file
7. A READ ME has to be provided with steps to execute the submitted solution, the Launch URLs of the Microservices in cloud must be specified.

   (Importantly, the READ ME should contain the steps to execute DB scripts, the LAUNCH URL of the application)
8. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable
9. You are supposed to use an In-memory database or sessions as specified, for the Microservices that will be deployed in cloud. No Physical database is suggested.

2.0 Introduction

2.1 Purpose of this document

The purpose of the software requirement document is to systematically capture requirements for the project and the system “Claims Management System” that has to be developed. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

The scope of this document is limited to addressing the requirements from a user, quality, and non-functional perspective.

High Level Design considerations are also specificed wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

2.2 Project Overview

A leading HealthCare Management Organization wants to strengthen its Middleware by exposing the core logic related to Claims Management as Microservices. This middle ware Microservices will be hosted on Cloud so that all the up/downstream applications can get an access to this for performing business transactions.

There will also be a Member Portal to be developed part of this scope that consumes these Microservices and responses back to members who are in need of Claim related information.

2.3 Scope

Below are the modules that needs to be developed part of the Project:

Req. No.	Req. Name	Req. Description
REQ_01	Claims Module	Claims Module is a Middleware Microservice that performs following operations: • Get Claim Status • Validate Eligibility of Claim and Action Settlement
REQ_02	Member Module	Member Module is a Middleware Microservice that performs the following operations: • View Bills • Submit Claim • View Claim Status
REQ_03	Policy Module	Policy Module is a Middleware Microservice that performs the following operations:

		• Get Chain of Permissible Providers (Hospitals) • Get Benefits permissible under a policy • Get Acceptable Claim Amount per benefit, per policy
REQ_04	Member Portal	An Web Portal that allows a member to Login and allows to do following operations: • Login • View Current Bill Status, Next Due • Verify Claim Status • Submit a Claim

Note: The project phase is for 2 weeks. The first week is to be developed on local machine and the second week deals with Cloud deployment.

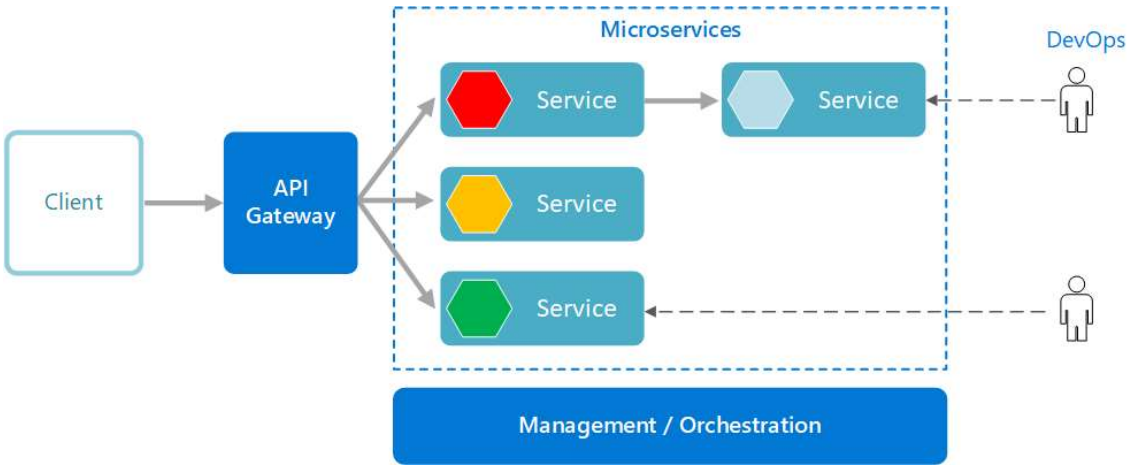
The requirement details given below states in-memory database usage. **The first phase of the development which is done in the first week, SHOULD use the Database for related activities and NOT the in-memory database.**

The second phase of the development which is done in the second week, can use the in-memory database as mentioned in the requirement, with appropriate code modifications.

2.4 Hardware and Software Requirement

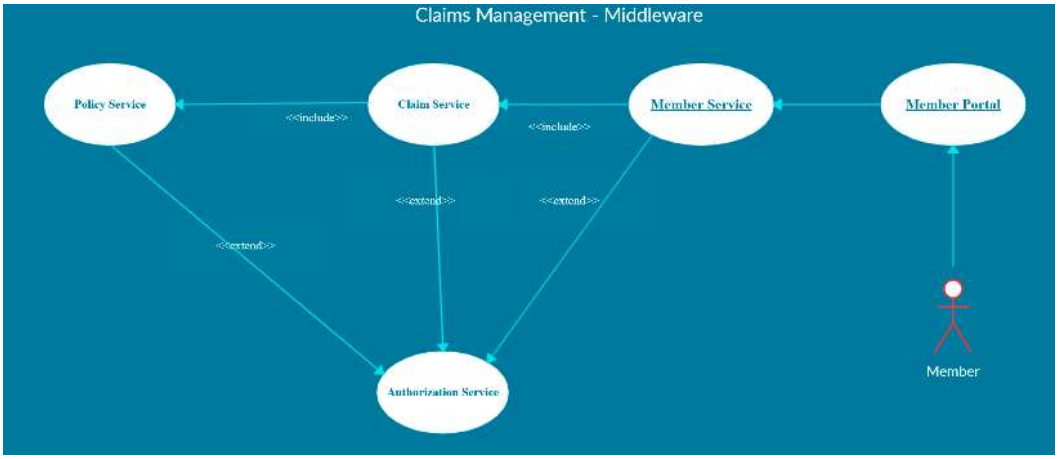
1. Hardware Requirement:
    a. Developer Desktop PC with 8GB RAM
2. Software Requirement (Java)
    a. Spring Tool Suite (STS) Or any Latest Eclipse
        i. Have PMD Plugin, EclEmma Code Coverage Plugin and AWS Code Commit Enabled
        ii. Configure Maven in Eclipse
    b. Maven
    c. Docker (Optional)
    d. Postman Client in Chrome

2.5 System Architecture Diagram



3.0 Functional Requirements and High Level Design

3.1 Use Case Diagram



3.2 Individual Components of the System

3.2.1 Member Microservice

Claims Management System	Member Microservice

Functional Requirements

Can assume that Member Portal App is the only client to this Microservice. An authorized member view the premium bills, submit the claim, can view the claim status, which is already submitted.

Post Authorization, basic Member based validation are performed in this Microservice and then it communicates to the below Microservices for retrieving necessary information.

Member Microservice will interact with the Claim Microservice for the following functionalities:

- the Microservice will interact with Claims Module, to check the eligibility of the member and the claim, and then action the claim to set for processing.
- To get the status of an already submitted claim

Entities

1. **Member**
   <Details of Member>
2. **Member_Policy**
   <Details of Policy subscribed by every member, Premium Details, Top-up Summary>
3. **Member_Claim**
   <Details of Member, Policy, Claim Status, Claim requested and settled details>
4. **Member_Premium**
   <Details of Primary Member, Policy Details, Premium Due and Payment Details>

REST End Points

Claim Microservice

- GET: /viewBills (Input: Member_ID, Policy_ID | Output: Last Premium Paid Date, Premium_Amount_Due, Details of Late Payment Charges if applicable, Due Date etc.)
- GET: /getClaimStatus (Input: Claim_ID, Policy_ID, Member_ID | Output: Claim Status, Claim Status Description)
- POST: /submitClaim (Input: Policy_ID, Member_ID, Claim_Details (Hospital ID, Benefits Availed, Total Billed Amount, Total Claimed Amount) | Output: Claim Status, Claim Status Description)

Trigger – Can be invoked from Member Portal (local MVC app)

Steps and Actions

1. Member Portal will request for any of the 4 operations as per the business logic
2. For all the 3 operations basic Member Profile will be verified before interacting with other Microservices.

3. At any point in time, the premium bills must be viewed by the Member Portal Client. Hence the Member Microservice must expose the Premium Details through /viewBills REST End Point. (Assume that the Premium Payment is not done in the portal, wherein it gets updated to the system as flat file).
4. If /getClaimStatus end point is invoked the Claims Microservice has to be invoked to get the claim status. The response returned by Claims has to be cascaded as the response in this end point.
5. If /submitClaim end point is invoked, then Claims Microservice has to be invoked with Claim Details. The new Claim submission status returned by Claims Microservice will be forwarded to the Member Portal by the Member Microservice.

Non-Functional Requirement:

- Only Authorized Member can access these REST End Points
- If the viewBill request is received multiple times for the same member and policy, there must not be multiple database hits leading to performance issues.

3.2.2 Claims Microservice

Claims Management System	Claims Microservice

Functional Requirements

Member Microservice interacts with Claims Microservice. Post authorization of request, Claim Microservice allows the following operations:

To view the status of submitted claim:

- Retrieve the claim status from database and return

To verify claim eligibility by interacting with Policy Microservice and action settlement:

- View the Claim details and check the following:
  i. If the Claimed Amount is applicable under the subscribed policy
  ii. If the Claimed benefit is applicable under the subscribed policy
  iii. If the Hospital in which benefits are availed is a permissible Health Care Provider (Hospital).
- If the above 3 conditions are satisfied, update the claim as “Pending Action” else “Claim Rejected”. If any information is not available or found to be invalid, then update status as “Insufficient Claim Details”. If any contradictory details found, update status as “Under Dispute”.

Entities

1. **Claim**

   <Claim details like Claim Number, Status, Remarks, Policy/Benefit Details, Hospital Details, Benefits Availed, Amount Claimed, Settled etc.>

REST End Points

Claims Microservice

- GET: /getClaimStatus (Input: Claim_ID, Policy_ID, Member_ID | Output: Claim Status, Claim Status Description)
- POST: /submitClaim (Input: Policy_ID, Member_ID, Claim_Details (Hospital ID,

Benefits Availed, Total Billed Amount, Total Claimed Amount) | Output: Claim Status, Claim Status Description)
Trigger – Can be invoked from Member Microservice
Steps and Actions 1. Claims Microservice will have 2 End Points exposed to Member Microservice 2. If /getClaimStatus end point is invoked by Member Microservice, the Claims Microservice will check the status in database and will return the response back to Member Microservice. 3. If /submitClaim end point is invoked, then the Claims Microservice will invoke the Policy Microservice for retrieving the permissible Provider Details (Hospital), eligible benefits of a policy and the eligible claim amoun for the benefits. o Based on the details retrieved, the Claims Microservice will decide on any of the following actions: sanctioning / rejecting / requesting further information / raising a dispute.
Non-Functional Requirement: • Hitting the Policy Microservice for 3 different details must happen in parallel to improve efficiency

3.2.3 Policy Microservice

Claims Management System	Policy Microservice
Functional Requirements Claims Microservice interacts with Policy Microservice. Post authorization of request, Claim Microservice allows the following operations: o Provide the permissible providers in which healthcare services can be offered. (Return the chain of hospitals) o Provide the list of benefits which the member is eligible to, under a subscribed policy. o To provide the eligible claim amount, for the given benefit under a subscribed policy.	
Entities **1. Policy** <Policy details like Policy Number, Benefits, Premium, Tenure etc.> **2. Member_Policy** <Member Policy details like Policy Number, Member Details, Subscription Date, Tenure, Benefits, Cap Amount for benefits etc.> **3. Provider_Policy** <Provider/Hospital details which is allowed to avail benefits from, for each policy in each location.> **REST End Points** **Policy Microservice** o GET: /getChainOfProviders (Input: Policy_ID | Output (Provider List, location wise) o GET: /getEligibleBenefits (Input: Policy_ID, Member_ID | Output: Benefits List for the	

Member, for the policy) ○ GET: /getEligibleClaimAmount (Input: Policy_ID, Member_ID, Benefit_ID | Output: Eligible Amount that can be claimed)
Trigger – Can be invoked from Claims Microservice
Steps and Actions 1. Claims Microservice will have 3 End Points exposed to Claims Microservice ○ For all the 3 End Points, straight forward details will be retrieved from the database

3.2.4 Authorization Microservice

Claims Management System	Authorization Microservice
Security Requirements ○ Service to Service communication has to happen using JWT ○ Pass End User Context across Microservices ○ Have the token expired after specific amount of time say 15 minutes. ○ Have this service configured in the cloud along with other services	

3.2.5 Swagger

Claims Management System	Swagger
Documentation Requirements (Java) ○ All the Microservices must be configured with Swagger for documentation ○ Register the swagger resources in the Swagger Microservice and enable them as REST end points ○ Configure this service along with other services in the cloud	

3.2.6 Member Portal (MVC)

Claims Management System	Member Portal
Client Portal Requirements ○ Member Portal must allow a member to Login. Once successfully logged in, the member do the following operations: ○ View Claim Status ○ Submit a Claim	

- View Bill Status of the subscribed policies
- Each of the above operations will reach out to the middleware Microservices that are hosted in cloud.

4.0 Cloud Deployment requirements

- All the Microservices must be deployed in Cloud
- All the Microservices must be independently deployable. They have to use In-memory database or user sessions wherever applicable
- The Microservices has to be dockerized and these containers must be hosted in Cloud using CI/CD pipelines
- The containers have to be orchestrated using AWS/Azure Kubernetes Services.
- These services must be consumed from an MVC app running in a local environment.

5.0 Design Considerations

These design specifications, technology features have to be strictly adhered to.

CDE-Project-Design Considerations.pptx

6.0 Reference learning

Please go through all of these k-point videos for Microservices deployment into AWS.

https://cognizant.kpoint.com/app/video/gcc-6e36500f-c1af-42c1-a6c7-ed8aac53ab22
https://cognizant.kpoint.com/app/video/gcc-92f246c9-024a-40b7-8bfc-96b3ce7c1a39
https://cognizant.kpoint.com/app/video/gcc-cfedd9c1-e29e-4e3e-b3e2-1960277f72a3

https://cognizant.kpoint.com/app/video/gcc-900a7172-43b7-42f3-a6cc-e301bd9cc9b3

Microservices deployment into Azure Kubernetes Service.

AzureWithCICD-1
AzureWithCICD-2
AzureWithCICD-3
AzureWithCICD-4

Other References:

Java 8 Parallel Programming	https://dzone.com/articles/parallel-and-asynchronous-programming-in-java-8
Feign client	https://dzone.com/articles/Microservices-communication-feign-as-rest-client
Swagger (Optional)	https://dzone.com/articles/centralized-documentation-in-Microservice-spring-b
ECL Emma Code Coverage	https://www.eclipse.org/community/eclipse_newsletter/2015/august/article1.php
Lombok Logging	https://javabydeveloper.com/lombok-slf4j-examples/
Spring Security	https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world
H2 In-memory Database	https://dzone.com/articles/spring-data-jpa-with-an-embedded-database-and-spring-boot https://www.baeldung.com/spring-boot-h2-database
AppInsights logging	https://www.codeproject.com/Tips/1044948/Logging-with-ApplicationInsights
Error response in WebApi	https://stackoverflow.com/questions/10732644/best-practice-to-return-errors-in-asp-net-web-api
Read content from CSV	https://stackoverflow.com/questions/26790477/read-csv-to-list-of-objects

Access app settings key from appSettings .json in a .Net core application	https://www.c-sharpcorner.com/article/reading-values-from-appsettings-json-in-asp-net-core/ https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-3.1

7.0 Change Log

	Changes Made			
V1.0.0	Initial baseline created on <24-Jul-2020> by <Srilakshmi Jayaraman>			
	Section No.	**Changed By**	**Effective Date**	**Changes Effected**