Cards & Payment System

Contents

1.0 Problem statement

The purpose of the requirements document is to systematically capture requirements for the project and the system to be developed as “**Card & Payment system**”. Both functional and non-functional requirements are captured in this document. It also serves as the input for the project scoping.

About the System

The client would like to develop an independent API based backend system. Main purpose of this system is expose below operations as set of APIs so that corresponding front end application can consume them

a. Create/Update new customer
b. Insert/Update Card Details detail associated with customer
c. Allow user to initiate/close payment transaction only if customer has already one card and no existing open payment transaction

Scope of the System

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

- Customer API (Customer Creation/Update) → Used by Front End to create/update new user profile into the system. The system stores the details of the customer in the system with below details:
    - Name (20 Alphanumeric)
    - Age (2 digit)
    - Email address (basic email validation)
    - Address (100 Alphanumeric)
    - Phone Number (10 digit)

- Cards API (Insert/Update Card Details) → Used by Front end to insert one Card details at a time. Input would be as below:
    - Card Number (16 digit)
    - Card Type (Visa/Master Card/Amex)
    - Customer Id
    - Expiration date (MM/dd/YYYY)

  Front end can update one card at a time. Before updating and inserting, any card details API need to validate if Customer Id is for valid customer.

- Payment API (Allow user to initiate/close payment transaction)→ This API would be used by front end inputs to initiate/close payment transaction
    - Card Number (16 digit)
    - Card Type (Visa/Master Card/Amex)
    - Customer Id
    - Expiration date (MM/DD/YYYY)
    - Payment amount (5 digit)
    - Status (Open/Closed/Failure)

  Generic Errors message should be returned based to the Front end in following condition:
    - Customer is not present
    - Card details is invalid

- One existing Open transaction is present

Client wants to utilize the micro service capability and want each API to be independent of each other and separately deployable.

Below are the few Non- functional scopes of these APIs:

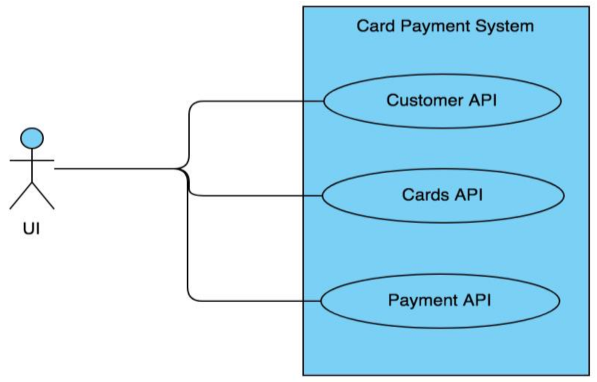
I. Along with updating inserting customer details Customer API would validate the customer details for any other API.
II. Along with updating inserting cards details Cards API would validate the card details for any other API.
III. Payment API need to provide fault tolerance in case any backend failure
IV. All the APIs must be registers in naming server
V. Payment API need to support client-side load balancing for Cards API.

2.0 Skill Tower to develop the project

Tower Name	**Topics**
Backend – Java	Java 8 Oracle Junit, Mockito Spring Boot, Hibernate, Spring Data, Microservices Spring Test Spring Framework, Spring Integration, Spring Cloud Spring Vault
Cloud -Paas – PCF	Cucumber Veracode Rabbit MQ Jenkins PCF CF push, Blue Green Deployment

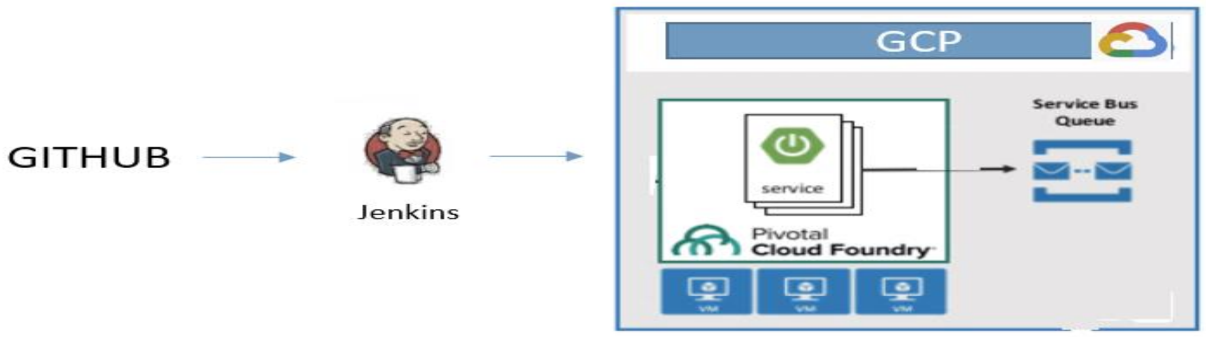
3.0 Use Case/Architecture Diagram

Use Case Diagram

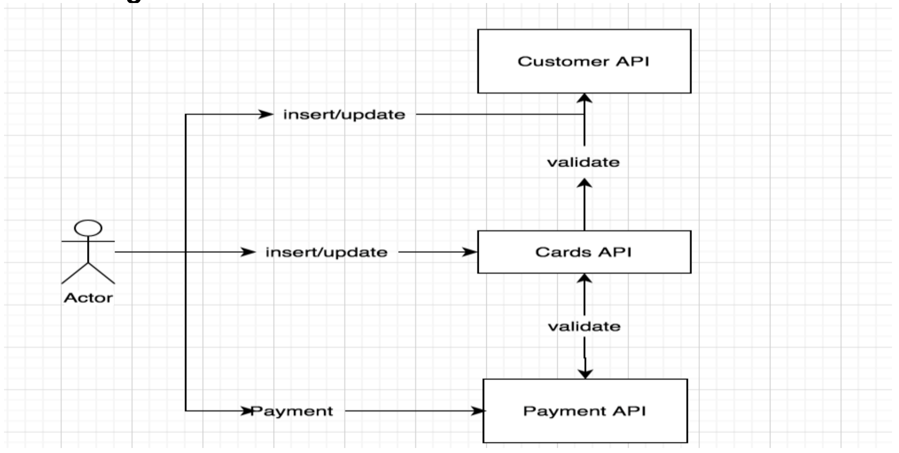


Architecture Diagram

Sample Architecture Diagram given below.



Flow Diagram



4.0 User Stories

User Story #	User Story Name	User Story
US_01	Customer API	As a customer, I should be able to register myself in the system or update my details in the system This API would support following three functionalities: 1. Create/insert customer 2. Update customer 3. Validate customer Acceptance criteria: Following fields must be validated before inserting in Database. o Name (20 Alphanumeric) o Age (2 digit) o Email address (basic email validation) o Address (100 Alphanumeric) o Phone Number (10 digit)
US_02	Cards API	As a customer, I should be able to add my card details or update my card details in the system. This API would support following three functionalities: 1. Insert card details for customer 2. Update card details for customer 3. Validate cards Acceptance criteria: Following fields must be validated before inserting in Database. o Card Number (16 digit) o Card Type (Visa/Master Card/Amex) o Customer Id o Expiration date (MM/dd/YYYY) Validate the customer id via customer API
US_03	Payment API	As a customer, I should be able to provide inputs to initiate/close payment transaction. This API would be used by front end inputs to initiate payment transaction Acceptance criteria: Following fields must be validated before inserting in Database. o Card Number (16 digit) o Card Type (Visa/Master Card/Amex) o Customer Id o Expiration date (MM/dd/YYYY)

		○ Payment amount (5 digit) ○ Status (Open/Closed/Failure) Validate the customer id via customer API Validate the card details via Cards API
US_04	PCF	As a developer I should be able to build and deploy the application using Jenkins pipeline to PCF Acceptance criteria: Create Jenkins deployment job to push the application to PCF. Application should be successfully built, run the Junit test cases, generate reports showing coverage and deployed and running via the Jenkins pipeline.

5.0 Expected Deliverables

Capture the deliverables to be submitted by associate after completing the development.

- Readme document on the complete application
  - Setup of the application
  - Swagger documentation for the microservices
  - API gateway for orchestration
  - JWT implementation for authentication(Apigee or similar)
  - Exception codes and handling
  - Logging framework that masks names and account numbers
  - High level steps used to convert to server less architecture
  - How to run the application
  - Any inference
  - Snapshot of any implementation
- Reports
  - Code Assessment Report
  - Functional Test Report
  - Vulnerability Assessment Report
  - Performance Test Report
  - Code Profiling Report
- Automation Scripts
  - Script file for Continuous Build, Continuous Integration & Continuous Deployment

6.0 Milestone and duration

As per the project requirement modification can be done in the below table.

Milestone	Topic
Milestone -1	Design and develop Microservice application using Spring Boot and Core Java 8 features.
Milestone -2	Deploy your application in PCF using Jenkins Pipeline and perform the different types of testing. Perform Veracode analysis

7.0 Implementation Notes

As per the project requirement modification can be done in the below table.

Backend - Java	Milestone-1 • Use Domain Driven Design • Use Github as code repository • Use Microservice Architecture(with patterns like service registry, discovery, circuit breaker) • Ensure 12 factor app methodology is followed • Use Spring Boot- Rest APIs to develop the services • Use Core Java 8 features wherever applicable. • Use Spring Data JPA to work with database (MySQL) • Use Gradle to build the application • Implement Spring Integration • Implement Spring Vault • Use OpenAPI to document the REST APIs • User authorization to allow/disallow CRUD operations • Any error message or exception should be logged (and help in refactor) • Unit test the application • All implementation should publish Code Quality Metrics using SonarCloud/SonarQube ○ Technical Debt – lower-the-better ○ Code Smell – lower-the-better ○ Cyclomatic Complexity - lower-the-better ○ Code Coverage – higher-the-better ○ Secure coding practices ○ Follow coding standards

Cloud - PCF	Milestone-3 • Develop pipeline using Jenkins • Perform SAST and DAST analysis using Veracode • Deploy the application in PCF

8.0 Evaluation rubrics

As per the project requirement modification can be done in the below table.

Microservices	• Follow the below basic structure ○ API - Controllers ○ Domain - Model, Events, Business Services Integration ○ Services – API Implementation ○ Infrastructure Project • Associate must have designed/developed Microservices as per the requirement • Associate must implement Spring Integration • Associate must implement Spring Vault • Each of the Microservices need to comprise below functionality, which need to be developed ○ Entity & Model classes, including appropriate relationship (like One-One, Many-One, etc…) between Entity Classes. (Entity and Model classes have been developed in the Previous Phase) ○ In case specific Entity or Model classes are required across multiple Microservices, it is recommended to maintain separate copy of Entity or Model classes for each Microservices. ○ Microservices should interact with corresponding databases it owns. ○ Microservice need to interact with other Microservice ○ Usage of Postman to test the Microservices by directly passing requests to each REST end Point, of each Microservice ○ Circuit Breaker, Service Registry, Service Discovery should be implemented ○ Consumer Driven Contract (CDC)
Rest API	• Associate must have used REST API for exposing resources • Associate must have used HTTP GET/PUT/POST request method designators for the business methods which is to be exposed

	• Associate must have customized the request and response formats according to the requirement • Associates must have used appropriate RETURN CODES based on the service outcome • Associates must have extracted query/form/header parameters from the input • Associate must have built a custom response based on the input
Core Java 8	• Associate should have used appropriate Java 8 features • Associate should have used appropriate Base class Libraries, Control Statements and Operators, File Handling and Java 8 features for implementing the functionalities.
Unit Testing	• Test cases covers the functionality of API with custom inputs • Good test Coverage
Common	• Code Smell • Technical Debt • Secured Coding • Coding Standards
DevOps on Cloud	• DevOps pipeline for each microservices which uses cloud PaaS services to trigger a CI/CD pipeline when code is checked-in to GIT • The check-in process should trigger unit tests with mocked dependencies • Unit tests should not alter persistent data • DevOps dashboard should show status of CI/CD pipeline • DevOps pipeline should support manual approval for rollout, gradual traffic shifting and rollback to earlier version • Checked-in code should meet 75%+ code coverage in unit testing