Cognizant Academy

Retail Banking System

FSE – Business Aligned Project Case Study Specification

Version 1.0

	Prepared By / Last Updated By	Reviewed By	Approved By
Name	Kumar Mahadevan		
Role	Solution Designer		
Signature			
Date			

Table of Contents

1.0 Important Instructions

1. Associate must adhere to the Design Considerations specific to each Technology 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. A TDD approach need to be followed
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. Design patterns when used need to be documented with their purpose of using it
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 “Retail Banking 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 specified wherever applicable, however the detailed design considerations have to be strictly adhered to during implementation.

2.2 Project Overview

One of the largest and leading Retail Bank within the US, serving millions of customers across the country offering a range of financial products from Credit Cards, Savings & Checking accounts, Auto loans, small business & commercial accounts. The retail bank has historically been served by a large monolith system. This system has Customer information, Transaction information, Account information – Pretty much a ledger generating taxes & statements. The bank is looking for a solution that will provide resilience & scalability for future growth. Following are the required features:

- Highly available
- Highly scalable
- Highly Performant
- Easily built and maintained
- Developed and deployed quickly

2.3 Scope

The retail banking system needs to expose the following resources:

- *Customers:* Handles creation of Customers within the bank and processing Loan requests
- *Accounts:* Handles creation, management & retrieval of a customer’s Banking Account
- *Transactions* : Handles creation and retrieval of transactions made against user’s bank account.
- *Rules*: Banking rules for different entities eg MinAccBalance, ServiceCharges, OverdraftLimits

2.4 Need for an MSA based solution

- The Retail bank requires High Availability for their operations to be available to their clients & bank employees without disruption.
- The Retail bank requires dynamic scaling. The bank periodically launches offers to its customers. On a given day, week, month of the year, the bank may experience heavy traffic. They would like to leverage the elastic features of cloud for saving costs.
- The Retail bank is currently constrained by the legacy system for depicting transaction data which is 80% of the customer traffic to the site & is extremely slow with an increasing customer base. The new system should be extremely performant for this use case
- The bank requires rapid development & deployment. It is also desired to use separate development teams to work in parallel on the core services supporting Account management, Transaction management, Customer Management services and their supporting applications simultaneously. This should accelerate development and delivery times.

2.5 Modules to be developed

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

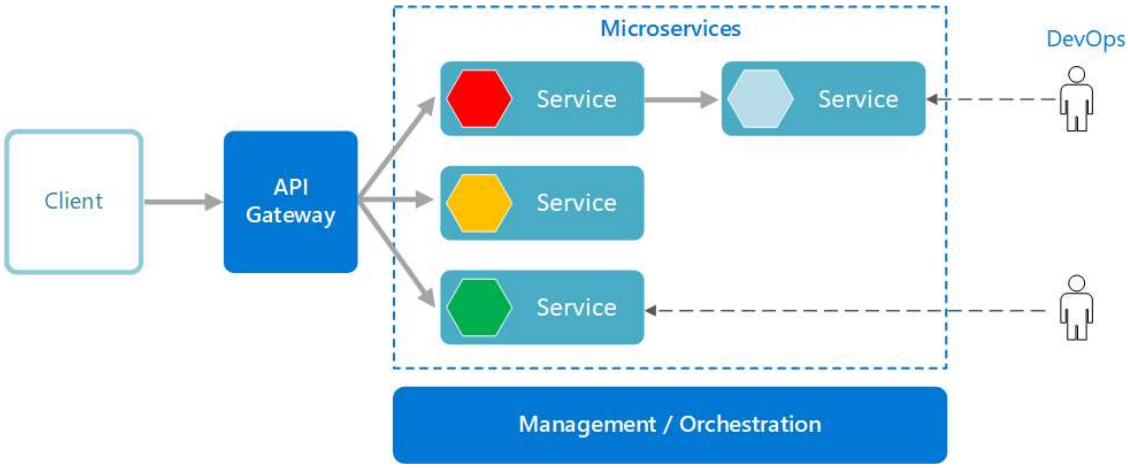
Req. No.	Req. Name	Req. Description
REQ_01	Account Management Module	Account management Module is a Microservice that performs following operations: • Get Customer Account(s) • Create Account • Get Account Statement • Withdraw • Deposit
REQ_02	Customer Module	Customer Module is a Microservice that performs the following operations: • Create Customer • Get Customer Details
REQ_03	Transactions Module	Transactions Module is a Microservice that performs the following operations: • Deposit • Withdraw • Transfer • Get Transactions
REQ_04	UI Portal	Loads the UI and takes care of user sessions. Relies on

		all other microservices for core functionality.
REQ_05	Authentication Module	Authentication Module is a Microservice that performs the following operations: • Login • Logout
REQ_06	Rules Module	The rules module is a microservce that will be responsible for evaluating rules while performing transactions like withdrawals, deposits. It will also return values based on the rules eg MinAccBal, ServiceCharges for non-maintenance of min balance. The following operations are performed: • Evaluate Min Balance • Get Service Charges

2.6 Hardware and Software Requirement

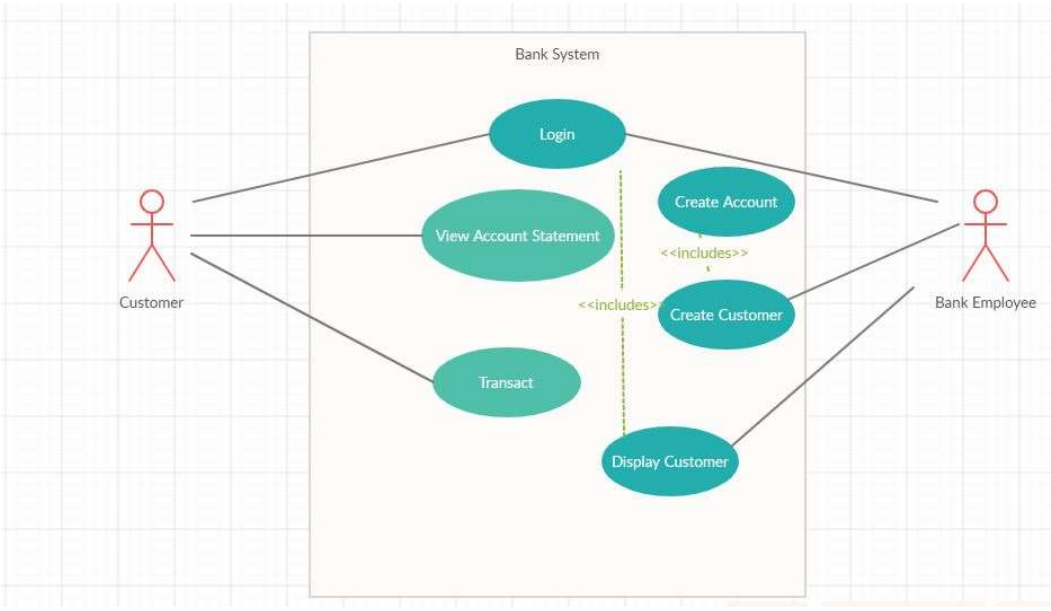
1. Hardware Requirement:
    a. Developer Desktop PC with 8GB RAM
2. Software Requirement (Java)
    a. Spring Tool Suite (STS) Or any Latest Eclipse
        i. Have PMD Plugin, EclEmma Code Coverage Plugin and AWS Code Commit Enabled
        ii. Configure Maven in Eclipse
    b. Maven
    c. Docker (Optional)
    d. Postman Client in Chrome
    e. AWS Cloud Account
3. Software Requirement (Dotnet)
    a. Visual studio 2017 enterprise edition
    b. SQL Server 2014
    c. Postman Client in Chrome
    d. Azure cloud access

2.7 System Architecture Diagram

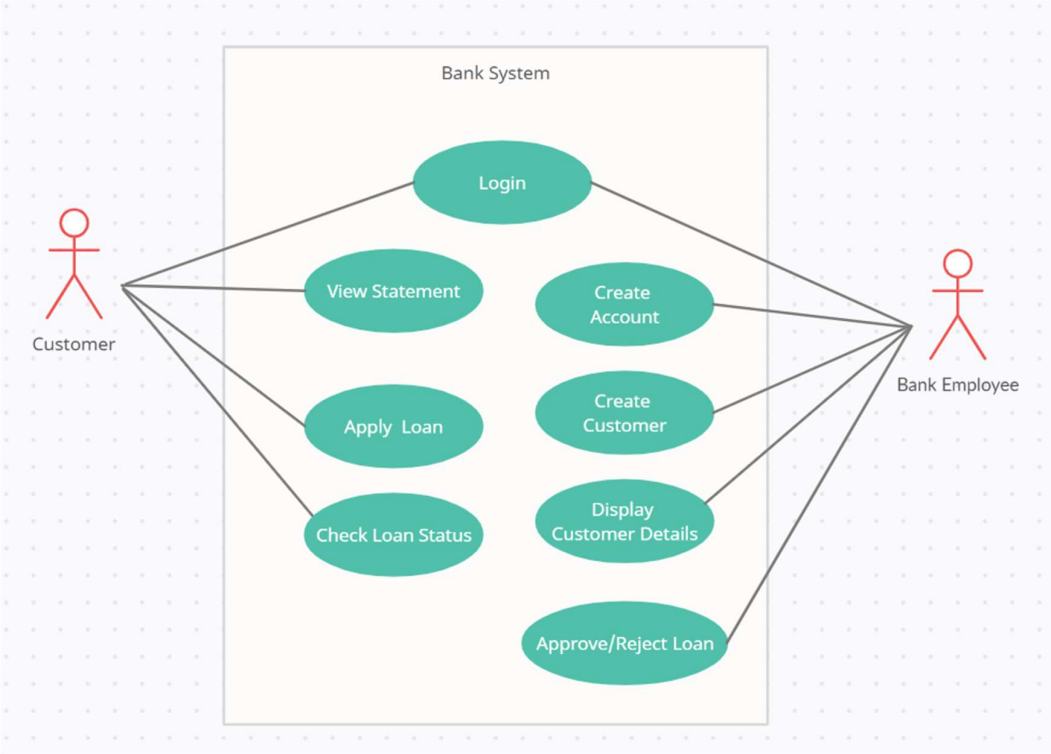


3.0 Functional Requirements and High Level Design

3.1 Use Case Diagram



Updated Use case diagram



3.2 Individual Components of the System

3.2.1 Customer Microservice

Retail Banking System	Customer Microservice
Functional Requirements The Customer microservice will perform the following tasks 1. Creating a customer profile based on the details provided by the customer 2. Displaying customer's profile	
Entities **1. Customer** <Details of Customer: Name, Address, DOB, PAN no …> **2. CustomerCreationStatus** <Details of Customer Creation: Message, CustomerId> **REST End Points** **Customer Microservice** ○ POST: /createCustomer (Input: Customer | Output: CustomerCreationStatus) ○ GET: /getCustomerDetails (Input: Customer_Id | Output: Customer)	
Trigger – Customer creation, Approve or Reject the Loan will be done by the Bank employee from the Banking Portal after inputting in his details. Customer can apply for Loan from Banking portal.	

Steps and Actions

1. The Bank employee logs in to the Portal
2. The Bank employee inputs Customer information within the Portal
3. createCustomer on the Customer Microservice will be called by passing the customer information.
4. After successful creation of customer, the Customer service will interact with the Account Service to create the customer's account. Account types of Savings & Current will be created by default. AccId will be generated by the Account Service
5. Bank employee may query on the customer to get information about the customer
    - Customer Details
    - Summary of all the accounts he holds. CustomerService will interact with the AccountService for fetching these details

Non-Functional Requirement:

- Only Authorized Members can access these REST End Points

3.2.2 Account Microservice

Retail Banking System	Account Microservice

Functional Requirements

The Account Microservice will perform the following tasks:

1. Creating an Account for a Customer
2. Fetching Accounts related to a Customer as a summary information
3. Fetching account statement for a particular customer based on a date range
4. Depositing to a customer's account
5. Withdrawing from a customer's account

Entities

1. **Account**
   <Account details like AccoutId.>

2. **AccountCreationStatus**
   <Message, AccountId.>

3. **Statement**
   <date, Narration, Chq/refno, ValueDate, Withdrawal, Deposit, ClosingBalance>

4. **TransactionStatus**
   <message, source_balance, destination_balance>

REST End Points

Account Microservice

- POST: /createAccount (Input: CustomerId) | Output: AccountCreationStatus
- GET: /getCustomerAccounts (Input:CustomerId | Output: Array of Account(id, Balance)
- GET: /getAccount(Input:AccountId | Output: Account(id, balance)
- GET:/getAccountStatement(Input: AccountId, from_date, to_date | Output: Statement
- POST:/deposit(Input: AccountId, amount | Output: TransactionStatus)
- POST:/withdraw(Input: AccountId, amount | Output: TransactionStatus)

Trigger – Invoked when a customer is created, customer logs in to the Portal, Bank employee queries customer details, customer views statements

Steps and Actions

1. Creation of an account will be triggered while creating a customer.
2. Two types of accounts will be created for a customer Savings, Current
3. Fetching Customer accounts will display a summary of all the accounts held by the customer
    a. Total account holdings value
    b. Individual account balance value
4. Account statement will display Statement details based on the date range. If no date range is provided, it will display Statement for a single month
5. Deposit & withdraw URI's can be invoked only by the transaction service. It cannot be directly invoked on the AccountService. So the AccountService needs to check for a token parameter prior to executing the service. This token will be attached by the transaction service
6. Transaction Status will show source & destination balances for Transfer only. For withdraw & deposit it will show only the current account's balance

Assumptions: Minimum balance to be maintained in the account will be derived from a business rule. The minimum balance will be assumed to have come from another transaction. It will be just added to the account balance while creating an account after querying the business rule service

Non-Functional Requirement:

- Only Authorized Members can access these REST End Points

3.2.3 Transactions Microservice

Retail Banking System	Transaction Microservice

Functional Requirements

The Transactions microservice will be responsible for performing all transactions within the Retal bank like Deposits, Withdrawal, Transfers. The service is responsible for checking business rules & propagating transaction contexts to Entities participating in the transaction. This service in turn will

invoke behavior on the Account service

Entities

TransactionsHistory
<< Refer to Section 4 for the Transaction History Data model >>

REST End Points

Transaction Microservice

- POST: /deposit (Input: AccountId, amount) | Output (TransactionStatus)
- POST: /withdraw (Input: AccountId, amount | Output: TransactionStatus)
- POST: /transfer (Input: Source_AccountId, Targer_AccountId, amount | Output: TransactionStatus)
- GET: /getTransactions (Input: CustomerId | Output:TransactionsHistory)

Trigger – Invoked whenever an user attempts to perform any transactions like deposit, withdraw, transfer amounts, get the transaction history on his accounts

Steps and Actions

1. Transactions microservice will be invoked whenever a user performs any transactions within the system like deposit, withdrawal, transfers or sees his transcation history
2. Transactions microservice will interact with Account microservice to actually complete the request.
3. For withdrawals, the service will initially check whether the withdrawal will result in non maintainence of min balance. If so the transaction will be declined. The rules microservices will be used for evaluating the business rules.
4. For transfers, the source account should maintain min withdrawal balance after the transfer. Else the transcation is declined. Rules microservice will be used for evalusting rules
5. All transactions need to be logged into a Transaction history DB and updated to the cache (Shown in figure 4.0).
6. Transaction history will retrieve transaction details of all accounts the customer has

NOTES:

- Transactions history information needs to be cached for good performance

Assumptions:

- Assume that the customer holds multiple accounts withn the same bank during Transfer

3.2.4 Rules Microservice

Retail Banking System	Rules Microservice

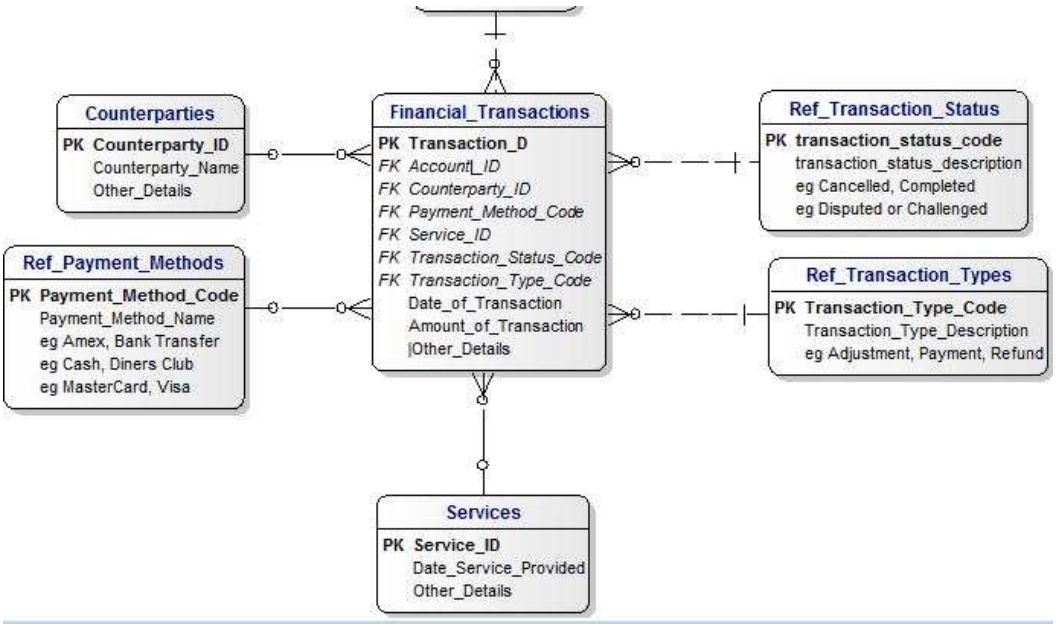
Functional Requirements The Rules microservice will be responsible for interacting with a rules engine to evaluate certain rules that is applicable prior to performing transactions	
Entities RuleStatus <status: denied, allowed, NA> **REST End Points** **Rules Microservice** ○ GET: /evaluateMinBal (Input: balance, AccountId) | Output (RuleStatus) ○ GET: /getServiceCharges (Input: | Output: float(indicating the service charge applicable)	
Trigger – Invoked from Transaction service while performing a withdrawal or transfer, System generated event for applying service charges on accounts that do not comply with the min bal criteria. Invoked from a monthly running batch job	
Steps and Actions • Whenever a transaction for Withdrawal or transfer happens, the transaction service will interact with the Rules service to check whether the account has a min balance criteria & whether the account maintains the criteria after the withdrawal or transfer has been performed. It will do so by checking against a business rules engine. • It will then return the evaluation status for the Transaction service which will help it to proceed with the transaction • The system will run a monthly batch job checking accounts that belong to the min balance criteria. When they don't meet the criteria, it will apply a service charge defined by the business rules. getServiceCharge will return the current service charge applicable to min balance accounts	

3.2.5 Bank Portal (MVC)

Retail Banking System	Bank Portal
Client Portal Requirements ○ Bank Portal must allow a customer to Login. Once successfully logged in, the customer can perform the following operations: ○ View his accounts and balances ○ Transact (Withdraw(Online transaction), Transfer) ○ View statements ○ View Transactions ○ Bank portal must allow bank employees to login. On successful login the bank employee can perform the following operations: ○ View customers account details ○ Create customers	

- For the first two steps, two roles ‘Customer’ & ‘Employee’ should exist & their screens should be customized according to the roles

4.0 Transactions history data model



5.0 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 Azure Kubernetes or AWS ECS Services.
- These services must be consumed from an MVC app running in a local environment.

6.0 Design Considerations

Java and Dotnet specific design considerations are attached here. These design specifications, technology features have to be strictly adhered to.

CDE-Project-Design Considerations.pptx

7.0 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 .Netcore 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

8.0 Change Log

	Changes Made			
V1.0.0	Initial baseline created on <21-Jul-2020> by <Srilakshmi Jayaraman>			
	Section No.	**Changed By**	**Effective Date**	**Changes Effected**