Cognizant Academy
Loan Management System

FSE – Business Aligned Project
Case Study Specification

Version 1.0

	Prepared By / Last Updated By	Reviewed By	Approved By
Name	Khaleelullah Hussaini Syed		
Role	Trainer		
Signature	t-syed8		
Date	26 October 2022		

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 Unit testing and 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 code level data as specified, for the Microservices that should be deployed in cloud. No Physical database is suggested for Microservice.

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 "Loan 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

XYZ bank has started a loan division in the banks. They need an application to manage the entire loan process for their bank. The application will be used by the bank and it's customers related to various loan related activities like viewing loan plan, applying for loans, paying EMI and managing any disputes related to loans.

2.3 Scope

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

Req. No.	Req. Name	Req. Description
REQ_01	**Loan Plans** module	• Using the loan plans module bank manager and customers can perform the following operations • Bank manager will be able to create a new loan plan in the system • The customers and bank managers can view the existing loan plans present in the system • Bank manager can update the existing loan plans
REQ_02	**Loan applications** module	• This module will provide the following functionalities to the application users • Customers can submit a loan application • A bank manager will be able to view the loan applications • A bank manager can approve or reject a loan application

REQ_03	**EMI Management** module	• This module will allow the features to the users • The bank manager can create an EMI plan • Customers can view their EMI plans • A customer can also pay their EMIs
REQ_04	**EMI extensions management** module	• This module supports the following operations in the system • The customers can request for extension of EMI payments • The branch manager can view all the pending requests for the extensions • The branch manager can either approve or reject the request by providing the sufficient remarks

Table 1 : Application Modules

3.0 Use Case Diagram

The following use case diagram shows various users of the system and their responsibilities.

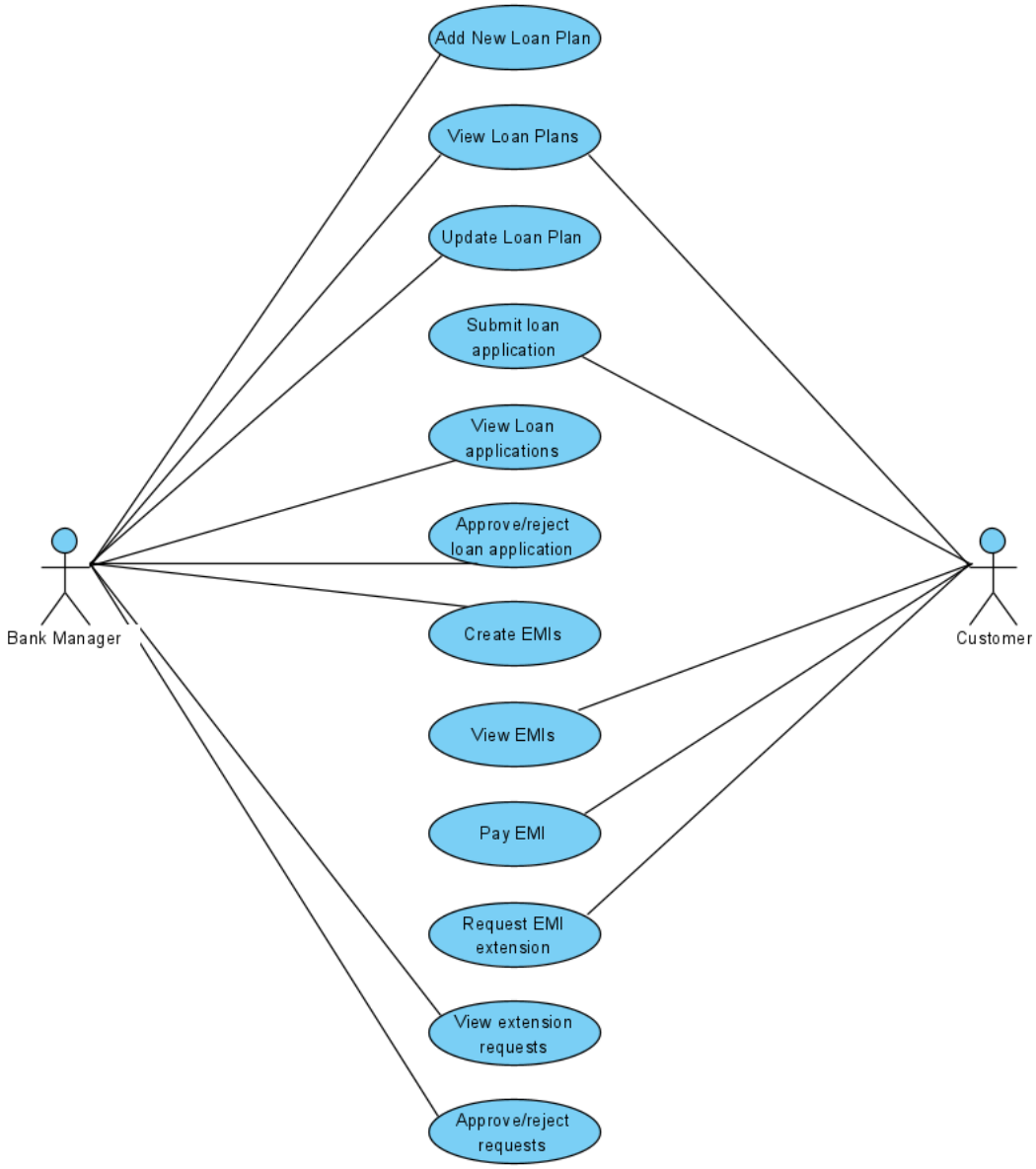


Figure 1 : Use case diagram

4.0 System Architecture Diagram

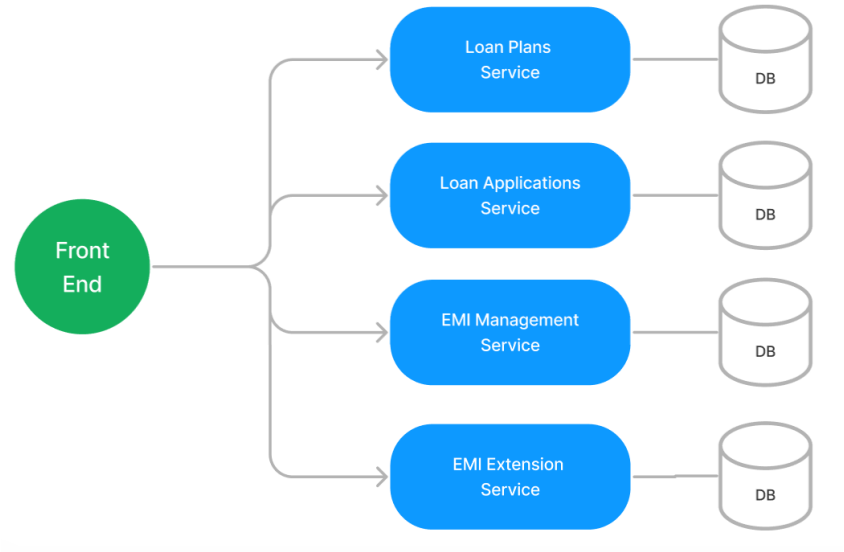
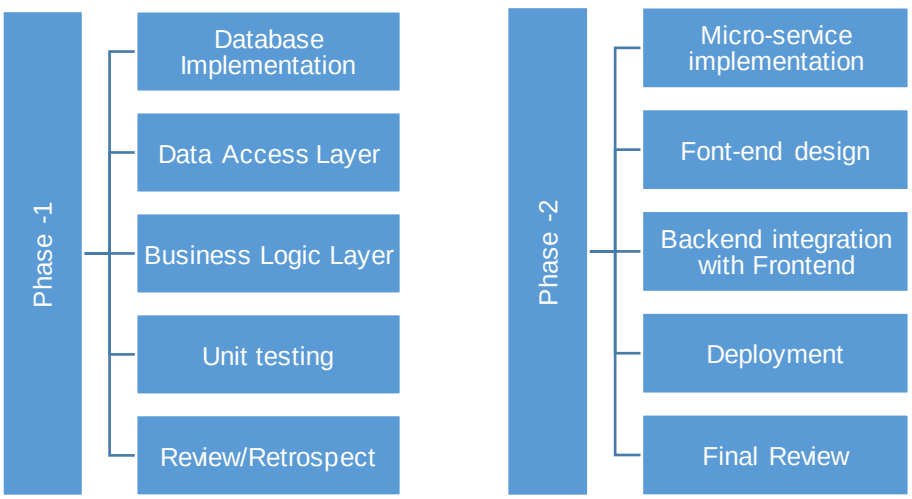


Figure 2 : Application Architecture Diagram

5.0 Development Phases

- The application will be developed in 2 phase.
- Each phase will have 4 stages followed by a review at the end.
- The phase-1 output will be unit tested core business logic of the application.
- In phase-2 the output will be a functional application with micro-service and the Front end.
- Each stage of the development phase must be completed alongside the learning milestone



6.0 System Requirements

6.1.1 Module – Loan Plans

Using the loan plans module bank manager and customers can perform the following operations.

1. Bank manager will be able to create a new loan plan in the system
2. The customers and bank managers can view the existing loan plans present in the system
3. Bank manager can update the existing loan plans

Stage: Database Implementation

1. Design a data base as per the following ER diagram provided.

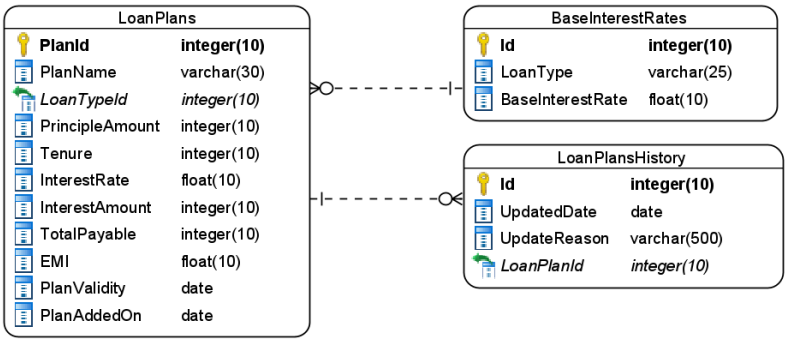


Figure 3 : ER Diagram – Loan Plans

2. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
    a. TotalPayable amount should be calculated based on Principle amount and interest amount
    b. Plan validity must be a future date
    c. Plan added on should be taken by default as date of insert operation
    d. Tenures should accept a maximum of 3 digits only

Note: Seed the data into the BaseInterestRates table with interest rates for Home, Personal, Medical, Vehicle as 8.5, 10, 7.5, 8.0 respectively

Stage: Data Access Layer Design

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
    a. Return list of interest rates
    b. Insert a new loan plan
    c. Return list of loan plans
    d. Update a loan plan
    e. Return a loan plan by ID

Stage: Business Logic Layer Development

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
    a. Fetch the list of interest rates
    b. Fetch the list of loan plans
    c. Fetch a loan plan by ID
    d. Add a new loan plan
    e. Update a loan plan
5. Following business rules must be implemented as part of the business service class
    a. Implement the rule to calculate interest amount as below

Type	Principle Amount	Tenure	Interest rate
All	5 Lakhs	5 years	Base rate
Personal	5-20 lakhs	5-20 years	Base rate + 0.2% for each year
Home	5-1 crore	5-30 years	Base rate + 0.3% for each year
Vehicle	5-30 lakhs	5-15 years	Base rate + 0.25% for each year
Medical	5-30 lakhs	5-10 years	Base rate + 0.25% for each year

Stage: Unit Testing

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

Stage: Micro-service implementation

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 2 : Loan Plans - Endpoint - 1

URL	/api/interestrates
Request Type	GET
User Role	Bank managers
Trigger	Front end
Description	This endpoint will allow the bank managers to get a list of base interest rates from the database
Inputs	
Outputs	InterestRateDTOs

Table 3 : Loan Plans - Endpoint - 2

URL	/api/loanplans
Request Type	POST
User Role	Bank managers
Trigger	Front end
Description	Using this endpoint bank managers can add a new loan plan in the system
Inputs	LoanPlanDTO
Outputs	Status Code and saved LoanPlanDTO

Table 4 : Loan Plans - Endpoint - 3

URL	/api/loanplans
Request Type	GET
User Role	Customers and Bank Managers
Trigger	Front end
Description	This endpoint will allow the users to view the available loan plans in the system
Inputs	
Outputs	LoanPlanDTOs

Table 5 : Loan Plans - Endpoint - 4

URL	/api/loanplans/<planid>
Request Type	GET
User Role	Bank managers, Customers
Trigger	Front end
Description	This endpoint is used to fetch details of a particular loan plan
Inputs	PlanId
Outputs	LoanPlanDTO or status code

Table 6 : Loan Plans - Endpoint - 5

URL	/api/loanplans/<planid>
Request Type	PUT
User Role	Bank managers
Trigger	Front end
Description	Bank managers can update a loan plan using this endpoint
Inputs	PlanId, LoanPlanDTO
Outputs	Status code

Stage: Font-end design

Create the following components as per the specification provided below.

1. AddLoanPlanFormComponent
   a. Develop a component which is used by the bank managers to add new loan plan in the system
   b. Provide a navigation to the component from main menu
   c. Component should contain a form to accept the plan details
   d. For loan plan type use a drop down list
   e. Use a range slider for tenure and interest rate.
   f. Once all details are validated the form should be allowed to be submitted and an acknowledgement must be displayed along with the interest amount, total payable and EMI amount

2. LoanPlansComponent

a. Design a component for customers and bank manager and provide a navigation to it via navbar
b. The component should display the details of a loan plan in a bootstrap card
c. Provide a toggle switch using radio button at the top of component to switch between customer and bank manager views
d. In bank manager view each bootstrap card should provide an edit button which will redirect to edit loan plan form component passing planid as route parameter.

3. EditLoanPlanFormComponent
   a. Design a component for bank managers to edit a loan plan details and provide a navigation to it via navbar
   b. The component should fetch and display the existing loan plan details by using the ID passed as route parameter
   c. The user should only be allowed to change the validity and plan name.
   d. Once the details are validated allow the user to submit the form and display an acknowledgement.

Stage: Integration of Frontend and backend

1. Create a data service in the font-end application which will communicate with the micro-services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

6.1.2 Module – Loan Applications

This module will provide the following functionalities to the application users

1. Customers can submit a loan application
2. A bank manager will be able to view the loan applications
3. A bank manager can approve or reject a loan application

Stage: Database Implementation

1. Design a data base as per the following ER diagram provided.

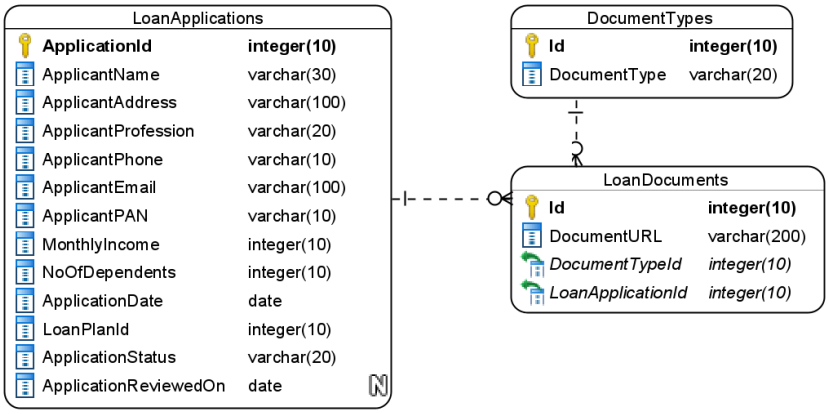


Figure 4 : ER Diagram – Loan Applications

2. Apply the following constraints apart from primary keys and foreign keys on the database
    a. ApplicantName must be minimum 10 characters long
    b. No of dependent can be 0 or more
    c. Phone number must be exactly 10 characters long
    d. Application status should be – New/Approved/Rejected

Note: Seed the data for document types as Bank statement, Payslip, ITR, SaleDeed, Medical certificate, ID proof, Address proof.

Stage: Data Access Layer Design

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
    a. Return document types list
    b. Insert a loan application
    c. Return list of loan applications
    d. Return a loan application by id
    e. Update a loan application status

Stage: Business Logic Layer Development

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes using the single responsibility principle which perform the given operations as follows
    a. Get all document types list
    b. Create loan application
    c. Get all new loan applications
    d. Get loan application by id
    e. Repond to loan application
5. Following business rules must be implemented as part of the business service class

a. Each customer can only have 2 new loan application and each of them must be for a different loan plan. If customer tries to create more than 2 new request then a user-defined exception "MaximumRequestLimitReachedException" should be thrown.
b. If a loan application is reject a customer can raise 2 more new applications for the same loan plan
c. User phone number should be 10 digits only

Stage: Unit Testing

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

Stage: Micro-service implementation

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All exceptions in the micro-service must be handled and logged using a logging library
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 7 : Loan Applications - Endpoint - 1

URL	/api/documenttypes
Request Type	GET
User Role	Customers
Trigger	Front end
Description	This enpoint will allow the customers to choose the type of document they are uploading
Inputs	
Outputs	DocumentTypeDTOs

Table 8 : Loan Applications - Endpoint - 2

URL	/api/loanapplications
Request Type	POST
User Role	Customers
Trigger	Front end
Description	Using this endpoint the customer can submit their application for the loan
Inputs	LoanApplicationDTO
Outputs	Status code

Table 9 : Loan Applications - Endpoint - 3

URL	/api/loanapplications
Request Type	GET
User Role	Bank managers
Trigger	Front end
Description	Bank manager can view the list of new loan applications submitted by the customers
Inputs	
Outputs	LoanApplicationDTO

Table 10 : Loan Applications - Endpoint - 4

URL	/api/loanapplications/<applicationid>
Request Type	GET
User Role	Bank managers
Trigger	Front end
Description	Using this endpoint the bank managers can view single loan application to approve/reject the application
Inputs	LoanApplicationID
Outputs	LoanApplicationDetailsDTO

Table 11 : Loan Applications - Endpoint - 5

URL	/api/loanapplications/<applicationid>
Request Type	PUT
User Role	Bank managers
Trigger	Front end
Description	With this endpoint the bank managers can submit a response either approving or rejecting a loan application
Inputs	LoanApplicationID, LoanApplicationResponseDTO
Outputs	Status code

Stage: Font-end design

Create the following components as per the specification provided below.

1. LoanApplicationFormComponent
   a. Create a form component which can be navigated to from the navigation bar by the customers.
   b. The component should contain a form to accept the loan application details.
   c. Use a dropdown to select the profession.
   d. Use the fileupload control to upload the supporting documents.
   e. Once all details are validated and successfully submitted an acknowledgement must be displayed.

2. LoanApplicationsListComponent
   a. Create a loan application list component which is accessible to bank managers by navigation from application menu.
   b. The component should list all the existing loan application submitted by the customers in the bootstrap table.
   c. Each row should have a process icon upon clicking it should navigate to process loan application component by passing the application via route parameters

3. ProcessLoanApplicationComponent
   a. Develop a component for bank managers which can be used to display the information of a single loan application so that it can be approved/rejected.
   b. The component should display all details of the loan application
   c. It should also provide a text box for rejection reason
   d. The component should also have a approve and reject button upon clicking it the loan application must be processed accordingly.

Stage: Integration of Frontend and backend

1. Create a data service in the font-end application which will communicate with the micro services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

6.1.3 Module – EMI Management

This module will allow the features to the users

1. The bank manager can create an EMI plan
2. Customers can view their EMI plans
3. The customers can also pay their EMIs

Stage: Database Implementation

1. Design a data base as per the following ER diagram provided.

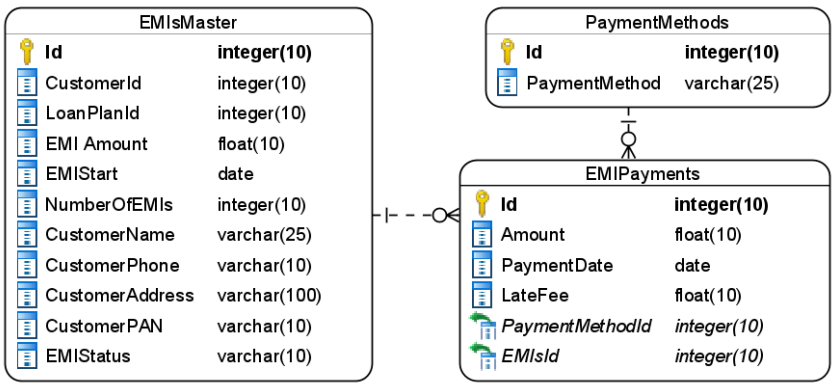


Figure 5 : ER Diagram - EMI Management

2. Apart from primary and foreing keys implement the following additional constraints
    a. Default EMI status should be OnGoing
    b. Allowed values for EMI status are – OnGoing/Defaulted/FullyPaid
    c. PaymentDate should be today by default.
    d. EMI amount should be positive number.

Note: Seed data into the payment methods table as Card/NetBanking/UPI respectively

Stage: Data Access Layer Design

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
    a. Return list of payment methods
    b. Insert a new EMI plan
    c. Return an EMI plan for customer loan
    d. Insert an EMI payment
    e. View EMI payments by customer loan

Stage: Business Logic Layer Development

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
    a. Get payment methods list
    b. Create new EMI plan
    c. Pay EMI
    d. View EMI plan for a customer loan
    e. View EMI payments for customer loan
5. Following business rules must be implemented as part of the business service class
    a. Total number of EMI payments by the customer for a loan should not exceed the number of EMI mentiond in the EMI plan. In-case of user paying more than the total number then raise a user-defined exception as "LoanPaymentCompletedException".
    b. In-case of late EMI payments that is after 5th of the month, then a fine of 0.025% of EMI amount should be levied per day.
    c. Ensure that a single customer doesn't have more than 2 active loans in the system

Stage: Unit Testing

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

Stage: Micro-service implementation

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. All the exceptions must be handled and logged using a logging library.
4. Create the following end-points and test them using postman and export the requests into a json file.

Table 12 : EMI Management - End point - 1

URL	/api/paymentmethods
Request Type	GET
User Role	Customers
Trigger	Front end
Description	Customers can choose one of the payments listed by this endpoint for paying EMI
Inputs	
Outputs	PaymentMethodDTOs

Table 13 : EMI Management - End point - 2

URL	/api/emiplans
Request Type	POST

User Role	Bank managers
Trigger	Front ends
Description	This endpoint will allow the bank managers to create a new EMI plan for the customers
Inputs	EMIPlanDTO
Outputs	Status code

Table 14 : EMI Management - End point - 3

URL	/api/emiplans/<customerid>/<loanplanid>
Request Type	GET
User Role	Customers
Trigger	Front end
Description	Customers can view their EMI plan using this endpoint
Inputs	CustomerId, LoanPlanId
Outputs	EMIPlanDTO

Table 15 : EMI Management - End point - 4

URL	/api/emiplans/<customerid>/<loanplanid>
Request Type	POST
User Role	Customers
Trigger	Front end
Description	This endpoint will be used by customers to pay an EMI
Inputs	CustomerId, LoanPlanId, EMIPaymentDTO
Outputs	Status code

Table 16 : EMI Management - End point - 5

URL	/api/emiplans/<customerid>/<loanplanid>/paymenthistory
Request Type	GET
User Role	Customers
Trigger	Front end
Description	Using this endpoint a customer can view their previous EMI payments
Inputs	CustomerId, LoandPlanId
Outputs	EMIPaymentDTOs

Stage: Font-end design

Create the following components as per the specification provided below.

1. NewEMIPlanFormComponent
   a. Develop a component to be used by bank managers which contains a form to create a new EMI plan.
   b. EMI status should be automatically taken as OnGoing
   c. Once all the details are validated, user should be able to get an acknowledgement on submission of form.

2. MyEMIPlanComponent
   a. Design a component which can be used by customers to view the details of their EMI plan
   b. The component should provide a textbox for accepting customer id and loan plan id and search and display the EMI plan data accordingly
   c. The component should also provide a button to view payment history upon clicking it the payment history for the loan emi should be displayed in a table
3. PayEMIComponent

a. Design a component which can be used by the customer to pay the EMIs
b. Add a navigation to the component via application menu bar.
c. Form should accept the EMI details and payment details
d. Payment method must be selected from the dropdown list
e. Once all details are validated, the form should be submitted and an acknowledgement is to be displayed to the customers.

Stage: Integration of Frontend and backend

1. Create a data service in the font-end application which will communicate with the micro-services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

6.1.4 Module – EMI Extensions Management

This module supports the following operations in the system

1. The customers can request for extension of EMI payments
2. The branch manager can view all the pending requests for the extensions
3. The branch manager can either approve or reject the request by providing the sufficient remarks.

Stage: Database Implementation

1. Design a data base as per the following ER diagram provided.

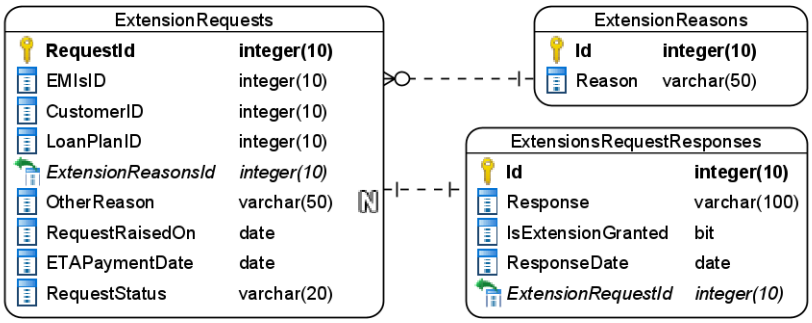


Figure 6 : ER Diagram – EMI Extensions

2. Enfore the following constraints along with primary and foreign keys
    a. ResponseDate must be same as RequestRaisedOn or a future date
    b. RequestRaisedOn must be taken as today by default
    c. ETAPaymentDate must be a future date
    d. Reponse should be atleast 50 characters long
    e. Allowed values for request status are – New/Approved/Pending

Note: Pre-populate the Extension Reasons table with 5 reasons. Make sure the last reason is added as others

Stage: Data Access Layer Design

1. Create a library project and add ORM support into it.
2. Use the ORM to map the entities to database as per the ER diagram provided.
3. Use repository per entity pattern and generate the repositories to perform the following operations
    a. Return extension reasons list
    b. Insert a new extensions request
    c. Return extension requests list
    d. Return an extension request by id
    e. Update an extension request status

Stage: Business Logic Layer Development

1. Develop a library which reference the Data Access Library project created earlier
2. This class library will contain various service classes which will encapsulate the business logic for the application.
3. Use dependency injection to in service classes to inject the required repositories.
4. Create the service classes following the single responsibility principle which perform the given operations as follows
    a. Fetch all extension reasons list
    b. Add a new extension request
    c. Fetch new extension requests
    d. Fetch an extension request by ID
    e. Approve/Reject extension request
5. Following business rules must be implemented as part of the business service class
    a. Each customer is allowed to have a maximum of 6 extension requests in a span of 3 years.
    b. A customer cannot raise more than 3 extension request for a single loan. If a customer tries to raise more than 3 requests generate a user-defined exception as “MaximumExtensionsLimitReachedException”
    c. In a single financial year more than 2 requests are not allowed for a customer

Stage: Unit Testing

1. Create a new Unit test project to test the service classes created in business logic layers
2. Mock all the repositories using a mocking framework.

Stage: Micro-service implementation

1. Create a API project which references the business logic layer created earlier
2. Implement service documentation using swagger
3. Create the following end-points and test them using postman and export the requests into a json file.

Table 17 : EMI Extensions - End point - 1

URL	/api/emiextensions/reasons
Request Type	GET

User Role	Customers
Trigger	Front end
Description	The customers can choose the extension reasons from the reasons list returned by the endpoint
Inputs	
Outputs	ReasonDTO

Table 18 : EMI Extensions - End point - 2

URL	/api/emiextensions/newrequest
Request Type	POST
User Role	Customers
Trigger	Front end
Description	Using this endpoint a customer can raise a new extension request for loan EMI payment
Inputs	ExtensionRequestDTO
Outputs	Status code

Table 19 : EMI Extensions - End point - 3

URL	/api/emiextensions
Request Type	GET
User Role	Bank managers
Trigger	Front end
Description	Using this endpoint a bank manager can view new emi extension requests raised in the system
Inputs	
Outputs	ExtensionRequestDTOs

Table 20 : EMI Extensions - End point - 4

URL	/api/emiextensions/<requestid>
Request Type	GET
User Role	Bank managers
Trigger	Front end
Description	The bank managers can use this endpoint to view the extension request so that they can approve/reject
Inputs	RequestID
Outputs	ExtensionRequestDTO

Table 21 : EMI Extensions - End point - 5

URL	/api/emiextensions/<requestid>
Request Type	PUT
User Role	Bank managers
Trigger	Front end
Description	With the help of this endpoint the bank manager can approve/reject the extensions request
Inputs	RequestID, ResponseDTO
Outputs	Status code

Stage: Font-end design

Create the following components as per the specification provided below.

1. NewExtensionRequestFormComponent
   a. Create a component with a form and allow the navigation to it for customers
   b. The component should allow the user to select the reason from a dropdown list.

c. If customer selects the reason as others, then he must manully provide a reason in textbox which should be hidden otherwise.
d. Validate all the data before it's submitted
e. On successful submission of request display an acknowledgement.

2. ExtensionRequestListComponent
   a. Create a component which is accessible to bank managers
   b. The component should display the extensions in a tabular formats using bootstrap tables
   c. Each ticket should have respond button which should navigate to Respond extension component by passing the request id through route parameters

3. RespondExtensionComponent
   a. Design a new component for bank managers process the extension requests.
   b. The component must display the request details along with accept/reject buttons.
   c. If bank manager should also provide a proper response when processing the request.
   d. Once the details are updated an acknowledgement should be displayed.

Stage: Integration of Frontend and backend

1. Create a data service in the font-end application which will communicate with the micro-services.
2. Use the data service in the components to make them interact with the API
3. Valid error messages should be shown based on various response status codes received form the API

7.0 Deployment requirements

1. All the Microservices must be deployed on a local web server like IIS or Apache Tomcat
2. All the Microservices must be independently deployable.
3. These services must be consumed from an Front end app running in a local environment.

8.0 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

Refer this link for the coding standards.

https://cognizantonline.sharepoint.com/:w:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/Coding%20standards/Effective%20coding%20standards.docx?d=w6430574d9db5478bbbe37c25b16e68e2&csf=1&web=1&e=84lTVf

Category	**Rule**
Database	Table names in database must be pascal cased and plural. All primary keys must be named as Pk_<table>. All foreign keys must be named as FK_<PrimaryKeyTable>_<ForeignKeyTable>
Database	Column names must be pascal cased. Multi-word column must be split using _ (underscore)
Coding	Follow pascal casing for naming classes, interfaces, methods, properties and other public members
Coding	Use camel casing for method parameter name, backing fields for properties and private variables. Consts must be capitalized
Coding	All exceptions must be handled and logged using a logging library
Coding	For communication between micro-services use the HttpClient class available in .Net and Java
Unit testing	Each method in services classes in business logic must be unit tested using nUnit/jUnit
Unit testing	Use a mocking library to mock the repositories while performing tests for business logic layer
Code Coverage	Should be minimum 90%
Front end(Angular/React ONLY)	Use pascal casing for the component names
Front end(Angular/React ONLY)	Create all components and data services in Angular/React project in dedicated folders
GitHub	Create ONLY Private Repositories. No password should be stored. DO NOT Mention in the Profile that You work for Cognizant

9.0 Reference learning

Please go through all of these k-point videos for

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 .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

10.0 Project Templates

backend-dotnet.zip

Java-Backend.zip

loanmanagement-angular-fe.zip

loanmanagement-react-fe.zip

11.0 Change Log

	Changes Made			
V1.0.0	Initial baseline created on 05-November-2022 by Khaleelullah Hussaini Syed			
	Section No.	**Changed By**	**Effective Date**	**Changes Effected**