|  |  |
| --- | --- |
|  | **Cognizant Academy**  **Bank Lending Portal**  **FSE – Business Aligned Project**  **Case Study Specification**  **Version 1.0** |
| |  |  |  |  | | --- | --- | --- | --- | |  | **Prepared By / Last Updated By** | **Reviewed By** | **Approved By** | | **Name** | Komila Kalra |  |  | | **Role** | Trainer |  |  | | **Signature** | t-komila3 |  |  | | **Date** | 9 November 2022 |  |  | |
|  |

Table of Contents

[1.0 Important Instructions 3](#_Toc116039845)

[2.0 Introduction 5](#_Toc116039846)

[2.1 Purpose of this document 5](#_Toc116039847)

[2.2 Project Overview 5](#_Toc116039848)

[2.3 Scope 5](#_Toc116039849)

[3.0 Use Case Diagram 7](#_Toc116039850)

[4.0 System Architecture Diagram 8](#_Toc116039851)

[5.0 Development Phases 8](#_Toc116039852)

[6.0 System Requirements 9](#_Toc116039853)

[**6.1.1** **Module – Project Management** 9](#_Toc116039854)

[**6.1.2** **Module – Sprint management** 12](#_Toc116039855)

[**6.1.3** **Module – Product Backlog Management** 16](#_Toc116039856)

[**6.1.4** **Module – Defects management** 20](#_Toc116039857)

[7.0 Deployment requirements 24](#_Toc116039858)

[8.0 Design Considerations 24](#_Toc116039859)

[8.1.1 Most Important and Common rules 24](#_Toc116039860)

[9.0 Reference learning 25](#_Toc116039861)

[10.0 Project Templates 27](#_Toc116039862)

[11.0 Change Log 27](#_Toc116039863)

# 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)

1. Follow coding best practices while implementing the solution. Use appropriate design patterns wherever applicable.
2. 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.

# Introduction

## Purpose of this document

The purpose of the software requirement document is to systematically capture requirements for the project and the system “Bank Lending Portal” 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.

## Project Overview

Cognizant technology solutions undertakes a number of projects for their clients spread across the globe. In order to manage and track the progress of these projects the company has decided that a custom made agile project management tool will be developed by a team of internal developer. The project is aimed at helping the bank to minimize lending risk and track the emi’s to save the bank from major loss.

## Scope

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

|  |  |  |
| --- | --- | --- |
| **Req. No.** | **Req. Name** | **Req. Description** |
| REQ\_01 | **Customer Management module** | * This module will be used by the customer for registering him/her and submitting loan app with the loan portal. * Customer can reject or accept approved loan * Customer can also update his/her information in portal |
| REQ\_02 | **Installment Calculator Management module** | * The Installment Calculator Management module will be used by banker and customer to calculate monthly installment * This module will give detail report how much prinical and interest is deducted each month |
| REQ\_03 | **Debt Collection Management module** | * The Debt Collection Management module will be used by bank officer. * The module keep track of customer installment’s * The module also minimize loss by identifying the defaulters so that timley recovery can be done by the team |
| REQ\_04 | **Loan Program management module** | * Views the loan application. * Check the credit risk. * If credit risk approves the loan then after customer’s acceptance the loan is sanctioned * Detailed installment plan is shared with customer |

Table 1 : Application Modules

# Use Case Diagram

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

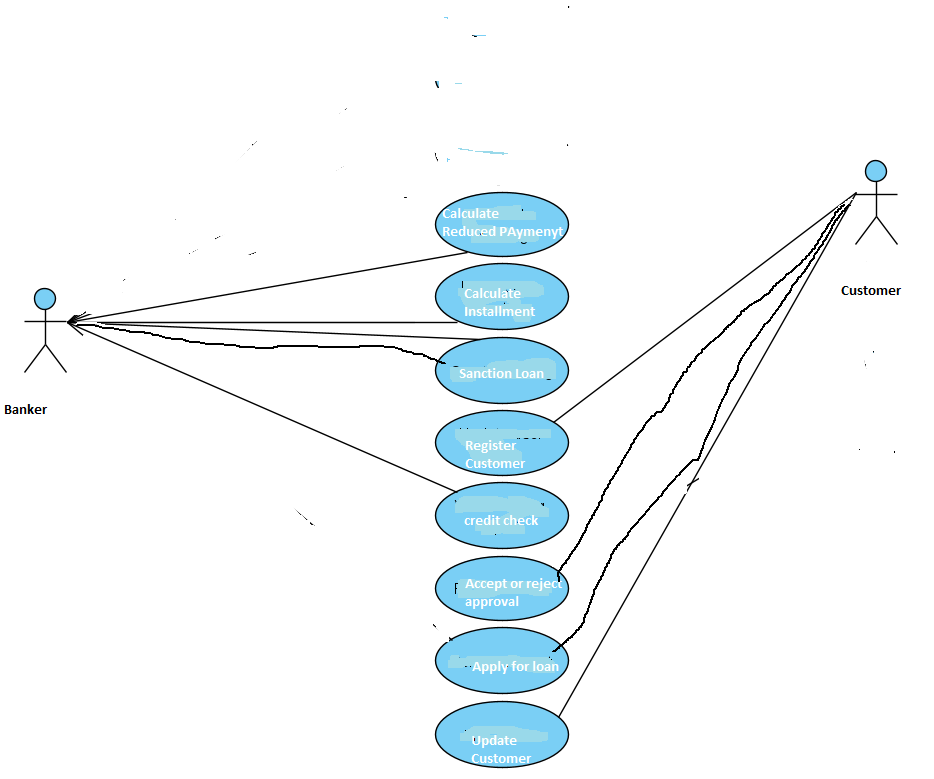


Figure 1 : Use case diagram

s

# System Architecture Diagram

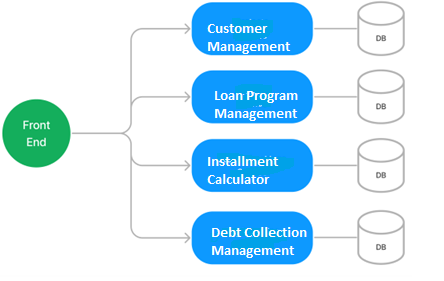


Figure 2 : Application Architecture Diagram

# 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 font-end.
* Each stage of the development phase must be completed alongside the learning milestone

# System Requirements

### **Module – Customer Management**

The Customer Management module handles ‘customers’. The module will provide the following features.

1. Register customer in Loan Portal
2. Customer can update his/her details
3. Apply for a loan
4. Update loan details
5. Accept or reject approved loan amount

**Stage: Database Implementation**

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

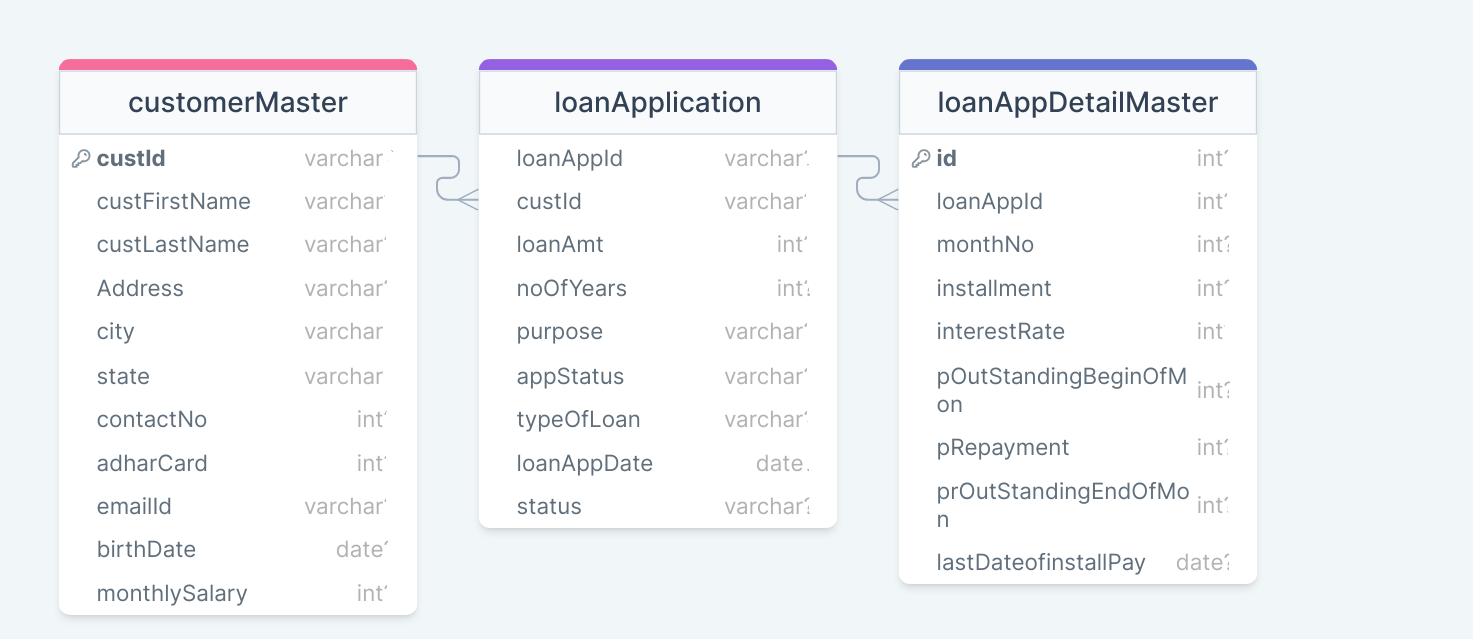


Figure 3 : ER Diagram – Customer Management

1. Enforce the following constraints on the database apart from primary key, foreign key and unique keys
   1. Title, first name and last name must be minimum 3 characters long
   2. Allowed values for App\_status are – NewLoan/Canceled/Approved/Sanctioned
   3. custID should always be 6 characters long
   4. Allowed values for the status is accepted/rejected/no status.
   5. loanAppDate should be current date

Note: Add few clients on the application startup.

**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
   1. Return list of customers
   2. Add new customer
   3. Update existing customer
   4. Return new Loan application list
   5. Add new Loan application
   6. Return a specific loan application
   7. Update Loan application
   8. Accept or Reject Loan approved

**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
   1. Return list of customers
   2. Add new customer
   3. Update existing customer
   4. Return new Loan application list
   5. Add new Loan application
   6. Return a specific loan application
   7. Return a specific loan application
   8. Update Loan application
   9. Accept or Reject Loan approved
5. Following business rules must be implemented as part of the business service class
   1. CustomerId must be auto-generated. It should be in the format XY0000. First 2 letters of lastname followed by a 4 digit number
   2. Phone number should be exactly 10 digits long
   3. Email address should always have @cognizant.com
   4. Firstname and lastname should only have alphabets and last name must be minimum 3 characters long.
   5. Adhar Card should be unique if a customer tries to register again with same AdharID system should throw exception as “DuplicateAccountException”.
   6. If a customer’s age is less than 18 or greater than 58 years system should throw exception as “AgeBarException” and account should not get created.
   7. Salary should be greater than zero
   8. LoanId must be auto-generated. It should be in the format XY0000. First 2 letters of loan type followed by a 4 digit number
   9. Amount should be positive
   10. Customer can update his loan record only if appstatus is new else it should throw exception “LoanUnderProcessingException”
   11. NoOfYear should be positive
   12. loanAppDate should be current date
   13. Customer can make changes in loan application only if loan status is new. Once the banker will start working on his/her loan application customer cannot change any information .He/she can only cnage status
   14. Customer can change status only if app\_status is approved

**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 : Custmer Management - Endpoint - 1

|  |  |
| --- | --- |
| **URL** | /api/customers/new |
| **Request Type** | POST |
| **User Role** | Customer |
| **Trigger** | Front end |
| **Description** | Using this endpoint the customer will be able to register him/her |
| **Inputs** | CustomerDTO |
| **Outputs** | Status code |

Table 3 : Custmer Management - Endpoint - 2

|  |  |
| --- | --- |
| **URL** | /api/customers |
| **Request Type** | GET |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the banker to view all the customer present in the system |
| **Inputs** |  |
| **Outputs** | CustomerDTOs |

Table 4: Custmer Management - Endpoint - 3

|  |  |
| --- | --- |
| **URL** | /api/customers/<custId> |
| **Request Type** | Get |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the banker will be able to see the information of customer |
| **Inputs** | custId |
| **Outputs** | CustomerDTO |

Table 5 : Custmer Management - Endpoint – 4

|  |  |
| --- | --- |
| **URL** | /api/customers/<custId>/update |
| **Request Type** | PUT |
| **User Role** | Customer |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the customer will be able to update his record |
| **Inputs** | custId, CustomerDTO |
| **Outputs** | Status code |

Table 6 : Custmer Management - Endpoint – 5

|  |  |
| --- | --- |
| **URL** | /api/customers/loan/<status> |
| **Request Type** | GET |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the banker to view all the loan records based on the status like NewLoan/Pending/approved/ |
| **Inputs** | Status |
| **Outputs** | CustomerDTOs |

Table 7 : Custmer Management - Endpoint – 6

|  |  |
| --- | --- |
| **URL** | /api/customers/loan/<date> |
| **Request Type** | GET |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the banker to view /pull all the loan records based the date/ |
| **Inputs** | Status |
| **Outputs** | LoanAppDTOs |

Table 8 : Custmer Management - Endpoint – 7

|  |  |
| --- | --- |
| **URL** | /api/customers/loan/<laonAppId>/update |
| **Request Type** | PUT |
| **User Role** | Customer |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the customer will be able to update his loan request |
| **Inputs** | loanAppId loanAppDTO |
| **Outputs** | Status code |

Table 9 : Custmer Management - Endpoint – 8

|  |  |
| --- | --- |
| **URL** | /api/customers/loan/<laonAppId> |
| **Request Type** | Get |
| **User Role** | Customer |
| **Trigger** | Front end |
| **Description** | With the help of this endpoint the customer will view his account |
| **Inputs** | loanAppID |
| **Outputs** | loanDTO |

Table 10 : Custmer Management - Endpoint – 9

|  |  |
| --- | --- |
| **URL** | /api/customers/loan/checkCustomerAcceptanceStatus/<loanAppID> |
| **Request Type** | Get |
| **User Role** | Banker |
| **Trigger** | BackEnd |
| **Description** | Using this endpoint the banker will check customer has accepted or canceled the loan |
| **Inputs** |  |
| **Outputs** | LoanAppDTO |

|  |  |
| --- | --- |
| **URL** | /api/loanapps/recievesanctionAmount/<loanAppID> |
| **Request Type** | Get |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Using this endpoint the customer will receive the reduced payment list |
| **Inputs** | loanAppID |
| **Outputs** | ReducedPaymentDTO’s |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. CreateCustomerComponent
2. Create a component which will be used by customer to register him/her
3. The component should provide a form for the user
4. The type of loan must be selected from a dropdown list
5. Once all the customer details are validated, customer should be able to submit the form and get an acknowledgement.

1. UpdateCustomerComponent
2. Design a update customer component and provide a navigation to it via navbar
3. The component can be used by customer to view and update his details
4. NewLoan ApplictionComponent
5. Provide a navigation to the component from the menu bar of the application
6. The component should provide a form for the user
7. The type of loan must be selected from a dropdown list
8. No of years must be selected from drop down list
9. Add a button next to dropdown to save the status.
10. Upon saving the status successfully display an acknowledgement.
11. UodateLoanApplicationComponent
12. Provide a navigation to the component from the menu bar of the application
13. The customer should be able tochange loan amount, number of years and type of loan only if loan status is “New”
14. If application status is “approved” only then customer should be able to change his status accepted/rejected i. drop down with accepted/rejected should be activated
15. Add a button next to dropdown to save the data
16. Upon saving the status successfully 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

### **Module – Loan program Management**

This module will provide various features to the banker for loan processing activity. Below are the features provided by this module

1. Pull new loan application on bases of date
2. Contact installment calculation dept to calculate monthly installment
3. Review loan application and calculate credit score
4. Approve loan
5. Sanction loan or close file

**Stage: Database Implementation**

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

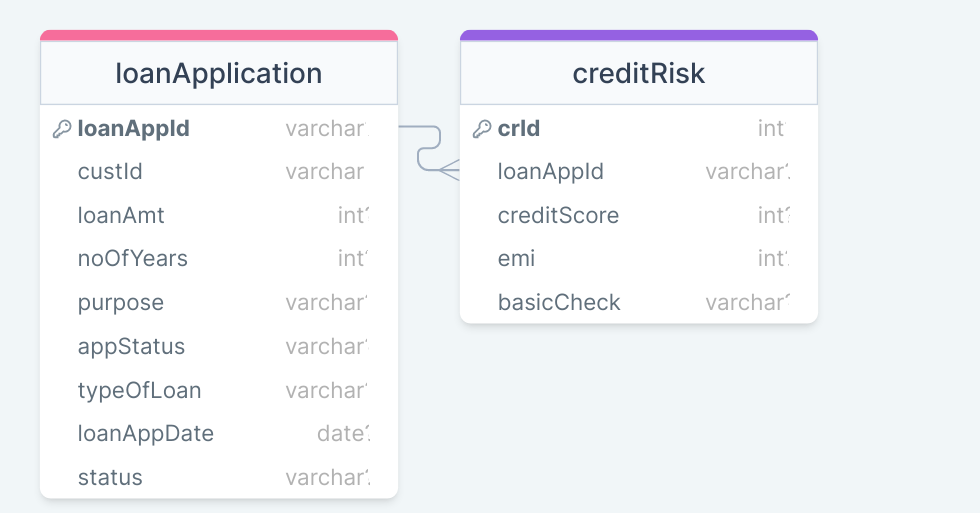


Figure 4 : ER Diagram – Loan Program Management

1. Apply the following constraints apart from primary keys and foreign keys on the database
   1. CrID must be auto-generated. It should be in the format CR<laonApID>. First 2 letters “CR” followed by a loanAppID.
   2. Values allowed for the basicCheck are – pass/fail/pending
   3. Value for emi should be positive
   4. Credit score should be between 200 to 700

**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
   1. Return list of new loan applications on the bases of date
   2. Return a specific loan application on the bases of loan application Id
   3. Insert monthly installment records in the installmentDetail table for every loan approved.
   4. Update the loan application on the bases on loan application 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 using the single responsibility principle which perform the given operations as follows
   1. Return list of new loan applications on the bases of date
   2. Return a specific loan application on the bases of loan application Id
   3. Insert monthly installment records in the installmentDetail table for every loan approved and accepted by customer.
   4. Check whether customer is meeting the loan criteria if yes then check the credit score
   5. Approve or cancel the loan of customer
   6. Update the loan application on the bases on loan application id
5. Following business rules must be implemented as part of the business service class
   1. Number of years loan should be less than customer’s i.e loan seekers retirement age. For example if customer age is 48 years he cannot get loan for 20 years. He/she can get loan only for 10 years.
   2. Monthly installment cannot be greater than 50% of inhand salary
   3. Customers age should be >= 18 years and <= 58
   4. Credit score should be greater than 500 . If credit score is less than 500 then loan is canceled

**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 11 : Loan Program Management - Endpoint – 1

|  |  |
| --- | --- |
| **URL** | /api/loanapps/pull/<date> |
| **Request Type** | GET |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | This endpoint will be used to fetch new loan applications from customer microservices on bases on date.Only after checking in local database. If records for date are not there in local database we will fetch from remote server |
| **Inputs** | Date |
| **Outputs** | LoanAppDTOs |

Table 12 : Loan Program Management - Endpoint – 2

|  |  |
| --- | --- |
| **URL** | /api/loanapps/update/<loan\_ap\_id> |
| **Request Type** | Update |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Banker will use the endpoint to update a loan application |
| **Inputs** | LoanAppDTO |
| **Outputs** |  |

Table 13 : Loan Program Management - Endpoint – 3

|  |  |
| --- | --- |
| **URL** | /api/ loanapps/view/<loan\_ap\_id> |
| **Request Type** | POST |
| **User Role** | Product owner |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the banker owners to view the loan information on bases of loan app id |
| **Inputs** | loan\_ap\_id |
| **Outputs** | LoanAppDTO |

Table 14 : Loan Program Management - Endpoint – 4

|  |  |
| --- | --- |
| **URL** | /api/loanapps/basicCheck |
| **Request Type** | PUT |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Using this endpoint the banker will do basic check like age and salary is more than 50 %of monthly emi |
| **Inputs** | LoanAppDTO |
| **Outputs** | LoanStatusDTO |

Table 15 : Loan Program Management - Endpoint – 5

|  |  |
| --- | --- |
| **URL** | /api/loaapps/callCheckEmi |
| **Request Type** | GET |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | The product owners will use this endpoint to update the details of a meeting |
| **Inputs** | LoanDTO |
| **Outputs** | EmiCalcDTO |

Table 16 : Loan Program Management - Endpoint – 6

|  |  |
| --- | --- |
| **URL** | /api/loanapps/creditScore |
| **Request Type** | PUT |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Using this endpoint the banker calculate credit score which is a final check for loan approval or rejection |
| **Inputs** | LoanAppDTO |
| **Outputs** | LoanStatusDTO |

Table 17 : Loan Program Management - Endpoint – 6

|  |  |
| --- | --- |
| **URL** | /api/loanapps/checkCustomerAcceptanceStatus/<loanAppID> |
| **Request Type** | Get |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Using this endpoint the banker will check customer has accepted or canceled the loan .If accepted then reduced payment is calculated and shared with customer |
| **Inputs** |  |
| **Outputs** | LoanAppDTO |

|  |  |
| --- | --- |
| **URL** | /api/loanapps/sanctionAmount/<loanAppID> |
| **Request Type** | Get |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Using this endpoint the banker request the installment calculation division to calculation reduced payment |
| **Inputs** | loanAppID |
| **Outputs** | ReducedPaymentDTO’s |

**Stage: Font-end design**

Create the following components as per the s

pecification provided below.

1. SanctionLoanAmountComponent
   1. Create a Sanction loan amount component which can be navigated to from the menu of the application.
   2. The component will provide a form to accept a Loan application id and sanction button to sanction loan of the applicant.

1. CreditScoreComponent
   1. Create a Sanction loan amount component which can be navigated to from the menu of the application.
   2. The component should contain a form to accept the credit score for a given loan application.
   3. The loan application id must be selected form a dropdown list.
   4. Once all details are validated and saved then an acknowledgement must be displayed
2. UodateLoanApplicationComponent
3. Provide a navigation to the component from the menu bar of the application
4. The banker should be not able tochange loan amount, number of years and type of loan
5. The banker can change application status is “approved” only then customer status is pending. drop down with aproved/rejected should be activated
6. Add a button next to dropdown to save the data
7. PullNewMonthlyLoanApplicationsComponent
   1. Create a component which will display the monthly payment collection.
   2. Component should also provide a date component with month and year and “Pull” button to pull the new monthly loan applications.
8. ViewSpecificLoanApplicationsComponent
   1. Create a component which will display the specific loan applicaiton.
   2. Component should also provide a date component with month and year and “view” button to display the specific loan application.

**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

### **Module – Debt Collection Management**

The banker should be able to minimize the risk during loan collection . This module will provide various features to banker

1. Pull the monthly tracking installment records from installement microservice
2. Banker should be able to update the list of customer’s who have paid installments.
3. Banker should be able to generate the report of defaulters.

**Stage: Database Implementation**

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

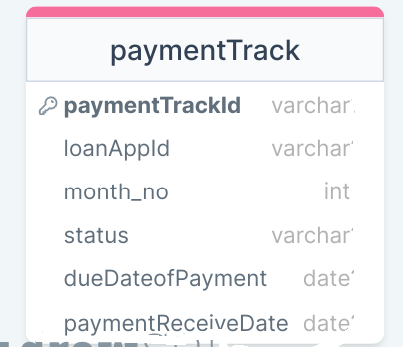


Figure 5 : ER Diagram – Debt Collection Management

1. Apart from primary and foreing keys implement the following additional constraints
   1. The values allowed for payment track status are – Received/Not Recieved The default value must be Not Received.

**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
   1. Add pulled monthly tracking record in payment track
   2. Update monthly installment payment status
   3. Return list of defaulters

**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
   1. Add pulled monthly tracking record in payment track
   2. Update monthly installment payment status
   3. Return list of defaulters
5. Following business rules must be implemented as part of the business service class
   1. Amount cannot be negative
   2. installmentStatus can be recieceved/not received
   3. Name of customer come’s in defaulter list if installment payment is not received before 15th of the said month

**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 18 : Debt Collection Management - End point - 1

|  |  |
| --- | --- |
| **URL** | /api/debtCollection/pull |
| **Request Type** | Get |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the banker to pull list of processed loan applications which need to be tracked for payment |
| **Inputs** | Month & year |
| **Outputs** | TrackDTOs |

Table 19 : Debt Collection Management - End point - 2

|  |  |
| --- | --- |
| **URL** | /api/debtCollection/create |
| **Request Type** | POST |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the banker to store the pulled records to track mothnly status of installments |
| **Inputs** | TrackDTOs |
| **Outputs** | Status code |

Table 20 : Debt Collection Management - End point - 3

|  |  |
| --- | --- |
| **URL** | /api/debtCollection/update/<loan-app-id> |
| **Request Type** | GET |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the banker to update monthly status of installments |
| **Inputs** | StatusDTO |
| **Outputs** | Status code |

Table 21 : Debt Collection Management - End point - 4

|  |  |
| --- | --- |
| **URL** | /api/debtCollection/listOfdefaulters |
| **Request Type** | GET |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | This endpoint will allow the banker to view list of defaulters who have not paid before 15th of said month |
| **Inputs** |  |
| **Outputs** | TrackDTO’s |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. UpdatePaymentStatusComponent
2. Design a component which can be used by developers to view the list of customer’s due for monthly installment in the form of table.
3. Each row in table must contain two button received , not received
4. PullMonthlyPaymentCollectionListComponent
   1. Create a component which will display the monthly payment collection.
   2. Component should also provide a date component with month and year and “Pull” button to pull the list of customers whose payment is due in said month and store it in database.
5. DefaulterReportComponent
   1. Develop a defaulter report component which will be used by the banker to see a defaulter’s report
   2. Provide a navigation to the component in the application menu
   3. Component should contain a date component with only month and year to and a “Show Report” button. Once the users provides a pmonth and year and clicks the show report button then display the defaulter report to the user in a bootstrap table

**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

### **Module – Installment Calculator management**

Allmost all the loan application goes through the Installement calculation process.

1. Monthly installment is calculated for the loan application’s if age validation has passed
2. Detailed installment report is generated for every sanctioned loan.
3. Add a new loan type and interest rate
4. Update the loan details

**Stage: Database Implementation**

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

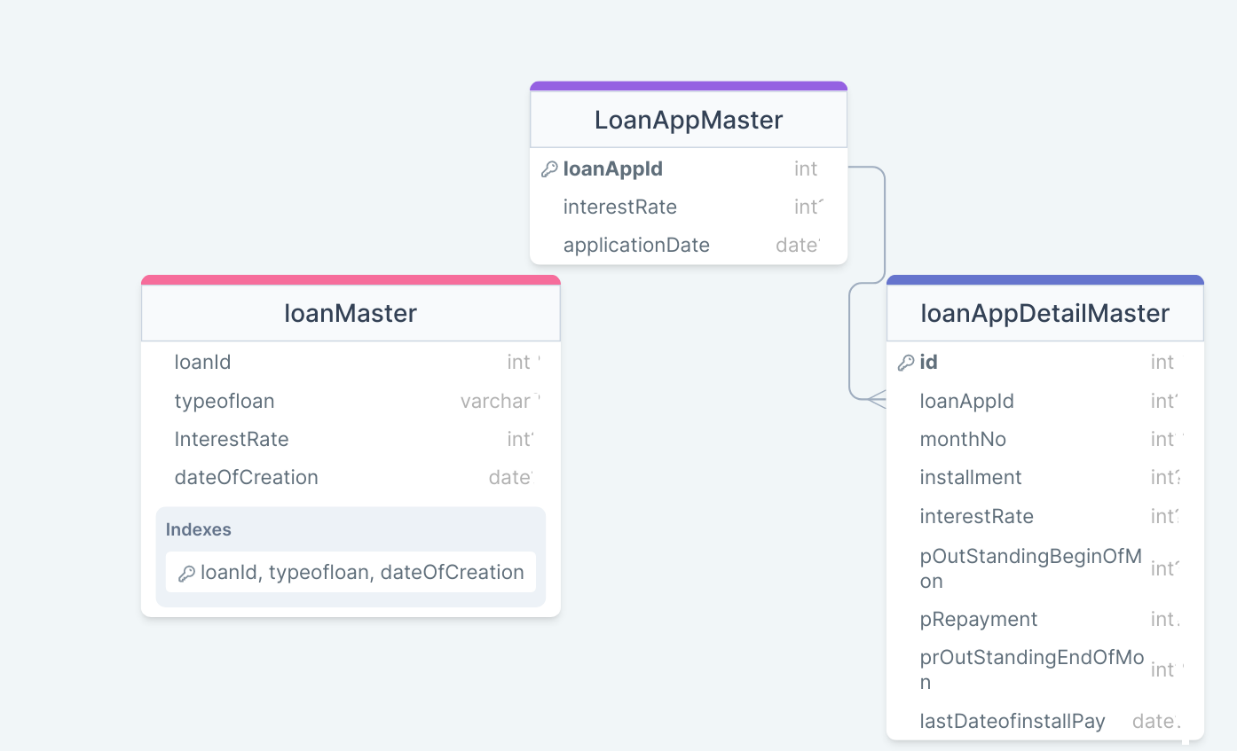


Figure 6 : ER Diagram – Installment Calculator Management

1. Enfore the following constraints along with primary and foreign keys
   1. DetectedOn must be taken as today by default
   2. Allowed values for priority is P1, P2 and P3
   3. Allowed values for severity are – Blocker, Critical, Major, Minor and Low
   4. ExpectedResolutionDate must be a future or today’s date.

**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
   1. Insert new loan
   2. Return information about particular type of loan
   3. Return Reduced payment method on bases of particular loan\_app\_id
   4. Update loan information on bases of 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
   1. Insert new loan
   2. Return information about particular type of loan
   3. Return Reduced payment method on bases of particular loan\_app\_id
   4. Update loan information on bases of id
   5. Return a defects report
   6. Calculate installement
5. Following business rules must be implemented as part of the business service class
   * 1. Note we can take loan for 10 moths or 3 months also not necessary to take for 10 years
     2. Loan interest rate cannot be updated if already some body has taken loan on that interest rate. In such cases we will add a new record and new customer’s will be given be given loan at latest interest rate. To find the current interest rate we will pick latest record on the bases of loan type and latest date of creation
     3. Month,interest rate and Principal Outstanding at the beginning of the month cannot be zero or negative
     4. Date of creation cannot be less than current date
   1. **Formula for installemt Calculation is -**
      1. **P x R x (1+R)^N / [(1+R)^N-1] where-**
      2. **P**= Principal loan amount
      3. **N**= Loan tenure in months
      4. **R**= Monthly interest rate
      5. The rate of interest **(R)**on your loan is calculated per month.
      6. **R** = Annual Rate of interest/12/100
      7. If rate of interest is 7.2% p.a. then r = 7.2/12/100 = 0.006 Note rate of interest will be picked up from Loan Master table . 7.2% is an assumption to explain the concept
      8. For example, If a person avails a loan of ₹10,00,000 at an annual interest rate of 7.2% for a tenure of 120 months (10 years), then his EMI will be calculated as under:
      9. **EMI= ₹10,00,000 \* 0.006 \* (1 + 0.006)120 / ((1 + 0.006)120 - 1) = ₹11,714.**
      10. The total amount payable will be ₹11,714 \* 120 = ₹14,05,703.
   2. **Reduced Payment method. Detailed installment report -**

Lets calculate interest and principal calculat

1. For the First Month

Principal Outstanding at the beginning of the month= Rs 50 lacs

#### Monthly Interest Rate = 10% ÷ 12 (10% is the annual interest rate. Therefore, for the month, you will pay 10% ÷ 12) = 0.8333%

#### Interest Component of EMI =  0.8333% × 50 lacs = Rs 41,667

#### Principal Component of EMI = Rs 48,251 – Rs 41,667 = Rs 6,584

#### Principal Outstanding at the end of the month (beginning of the next month) = Rs 50 lacs – Rs 6,584 = Rs 49.93 lacs

. 2. For the Second Month

Principal Outstanding at the beginning of the month= Rs 49.93 lacs

#### Monthly Interest Rate = 10% ÷ 12  = 0.8333%

#### Interest Component of EMI =  0.8333% × 49.93 lacs = Rs 41,612

#### Principal Component of EMI = Rs 48,251 – Rs 41,612 = Rs 6,639

#### d.Principal Outstanding at the end of the month (beginning of the next month) = Rs 49.93 lacs – Rs 6,639 = Rs 49.87 lacs

Here is how the payment schedule will go (assuming interest rate remains constant and there are no prepayments)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Loan\_App\_ID** | **Month** | **Principal Outstanding at the Beginning of the Month** | **EMI** | **Interest** | **Principal Repayment** | **Principal Outstanding at the End of the Month** | **Last date of emi payment** |
| HO002 | 1 | 5,000,000 | 48,251 | 41,667 | 6,584 | 4,993,416 | 10-9-2022 |
| HO002 | 2 | 4,993,416 | 48,251 | 41,612 | 6,639 | 4,986,776 | 10-10-2022 |
| HO002 | 3 | 4,986,776 | 48,251 | 41,556 | 6,695 | 4,980,082 | 10-11-2022 |
| HO002 | 4 | 4,980,082 | 48,251 | 41,501 | 6,750 | 4,973,331 | 10-12-2022 |

**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 22 : Installment Calculator Management- End point - 1

|  |  |
| --- | --- |
| **URL** | /api/loan/new |
| **Request Type** | POST |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Using this endpoint the banker will introduce a new loan type in bank.It means loan master table |
| **Inputs** | LoanDTO |
| **Outputs** | Status code |

Table 23 : Installment Calculator Management - End point - 2

|  |  |
| --- | --- |
| **URL** | /api/ loan /<loan\_id> |
| **Request Type** | Get |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Using this endpoint the banker will be able to view a particular type of loan |
| **Inputs** | Loan\_id |
| **Outputs** | LoanDTO |

Table 24 : Installment Calculator Management - End point - 3

|  |  |
| --- | --- |
| **URL** | /api/ loan /<loan\_id>/update |
| **Request Type** | Get |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Using this endpoint the banker will be able edit the loan information |
| **Inputs** |  |
| **Outputs** | LoanDTO |

Table 25 : Installment Calculator Management - End point - 4

|  |  |
| --- | --- |
| **URL** | /api/loan |
| **Request Type** | Get |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Using this endpoint the banker will be able to view all types of loan |
| **Inputs** |  |
| **Outputs** | LoanDTO’s |

Table 26 : Installment Calculator Management - End point - 5

|  |  |
| --- | --- |
| **URL** | /api/loan/emicalc |
| **Request Type** | Post |
| **User Role** | Banker |
| **Trigger** | Front end |
| **Description** | Banker will use this endpoint to calculate monthly emi |
| **Inputs** | LoanCalcDTO |
| **Outputs** | LoanCalcDTO |

Table 27 : Installment Calculator Management - End point - 6

|  |  |
| --- | --- |
| **URL** | /api/loan/reducedPaymentCalc |
| **Request Type** | Post |
| **User Role** | Banker |
| **Trigger** | Frontend |
| **Description** | A banker will be able to view all reduced payment details. Banker will check if database does not have the details in database, It will calculate and store it in database |
| **Inputs** | LoanCalcDTO |
| **Outputs** | ReducedPaymentDTO’s |

**Stage: Font-end design**

Create the following components as per the specification provided below.

1. NewLoanComponent
2. Create a component which contains a form to accept details of a loan.
3. Priority must be accepted using radio buttons
4. The severity should be selected from a dropdown list
5. Once all the details are validated then users must be allowed to submit the form and get an acknowledgement.

1. UpdateLoanComponent
2. Create a component which is accessible to developers from the nav bar
3. Search Loan component is displayed in update Loan component
4. The component should allow to make in changes in loan record.
5. Each row should have view details button which should navigate to Defect details component by passing in the defect id.
6. Once all the details are validated then users must be allowed to submit the form and get an acknowledgement.
7. Search LoanComponent
8. Develop a component which is accessible to the banker from the application nav bar
9. Provide a textbox to enter loan id code and a button to fetch the loan information.
10. The loan information should be displayed in a tabular format.

**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

# 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.

# Design Considerations

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



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>

### Most Important and Common rules

|  |  |
| --- | --- |
| **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 |

Any deviation from the high level design must be approved by trainer, mentor and solutions team

# Reference learning

Please go through all of these k-point videos for

Microservices deployment into Azure Kubernetes Service.

|  |
| --- |
| [AzureWithCICD-1](https://cognizant.kpoint.com/app/video/gcc-19532393-d4e0-4fd9-8a0c-80ecbdb349d3) |
| [AzureWithCICD-2](https://cognizant.kpoint.com/app/video/gcc-6633a958-ab72-4c69-b926-fe832e4b56a1) |
| [AzureWithCICD-3](https://cognizant.kpoint.com/app/video/gcc-553eb186-c1cf-448e-96fc-a96fe37b2e6a) |
| [AzureWithCICD-4](https://cognizant.kpoint.com/app/video/gcc-fad7d4af-d651-4501-99c6-2785190670c2) |

**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](https://dzone.com/articles/microservices-communication-feign-as-rest-client) |
| Swagger (Optional) | [https://dzone.com/articles/centralized-documentation-in-Microservice-spring-b](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> |

# Project Templates









# Change Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Changes Made | | | |
| V1.0.0 | Initial baseline created on 9-November-2022 by Komila Kalra | | | |
| V1.0.1 |  | | | |
| **Section No.** | **Changed By** | **Effective Date** | **Changes Effected** |
| 2.3 | Komila Kalra | 9 November 2022 |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |