System Requirements Specification Index

For

E-Loan Application (Collaborative)

Version 4.0

IIHT Pvt. Ltd.
IIHT Ltd, No: 15, 2nd Floor, Sri Lakshmi Complex, Off MG Road, Near SBI LHO,
Bangalore, Karnataka – 560001, India
fullstack@iiht.com

Table of Contents

1. Business-Requirement:

1.1 Problem Statement:

E-Loan Application is .net core web API application integrated with MS SQL Database, where it allows customers to apply for Loan Online, and the same would be processed by the loan clerk and sanctioned by the manager from the bank.

1.2 Following is the requirement specification:

	E-LOAN
USERS	
1	Manager
2	Clerk
3	Customer
Customer Functionalities	
1	Can apply for Mortgage loan on a property
2	Can Update Mortgage
3	Get loan status of application
Loan Clerk Functionalities	
1	Can list all Loan Application to be processed
2	Can list all received application
3	Start the loan process after verify, Make sure loan status is "received" before process loan application, Make the loan application as received and check is it received.
Manager Functionalities	
1	Can list all Loan Application processed by Loan clerk
2	Can Accept or reject a loan application (with remarks)
3	If Loan is sanctioned, following information should be furnished

2. Assumptions, Dependencies, Risks / Constraints

2.1 Customer Constraints

- While applying for a loan by a customer, all fields should be filled if not fill then operation should throw custom exception.
- While fetching loan status, if loan application id does not exist then operation should throw a custom exception.
- While fetching all loan details of a customer, if customer id does not exist then operation should throw custom exception.
- While fetching all loan details of a customer, if the customer is disabled then operation should throw a custom exception.
- While updating the loan details by customer, if loan application id does not exist then operation should throw custom exception.
- While deleting the loan details by customer, if loan application id does not exist then operation should throw custom exception.
- While updating the loan details by the customer, if the loan application is already processed by the clerk, then the operation should throw a custom exception.
- While deleting the loan application by the customer, if the loan application is already processed by the clerk, then the operation should throw a custom exception.

2.2 Clerk Constraints

- While processing the loan by the loan clerk, make sure loan application status is in “Received” status and then the operation should throw a custom exception.
- While processing a loan by a loan clerk, if the clerk is disabled then the operation should throw a custom exception.
- While processing the loan by the loan clerk, if loan application id does not exist then operation should throw a custom exception.
- While processing a loan by the loan clerk, if the loan has already been processed then the operation should throw a custom exception.
- While updating the processing info by the loan clerk, if clerk id does not exist then operation should throw a custom exception.
- While updating the processing info by the loan clerk, if the clerk is disabled then operation should throw a custom exception.
- While updating the processing info by the loan clerk, if loan application id does not exist then operation should throw a custom exception.
- While updating the processing info by the loan clerk, if the loan application has already been finalized (accepted or rejected and Done) by the manager, then the operation should throw a custom exception.

- While fetching all loans processed by the clerk, if the clerk id does not exist then the operation should throw a custom exception.
- While fetching all loans processed by the clerk, if the clerk is disabled then operation should throw a custom exception.

2.3 Manager Constraints

- While sanctioning or rejecting loan by manager, loan application status must be in “Accept” status, if it does not exist then operation should throw custom exception.
- While sanctioning or rejecting a loan by the manager, if the manager is disabled then operation should throw a custom exception.
- While sanctioning or rejecting loan by manager, if loan application id does not exist then operation should throw custom exception.
- While sanctioning or rejecting loan by manager, if loan has already been finalized (rejected or sanctioned) then operation should throw custom exception.
- While fetching all loans finalized by manager, if manager id does not exist then operation should throw custom exception.
- While fetching all loans finalized by the manager, if the manager is disabled then the operation should throw a custom exception.

2.4 Common Constraints

- All users must be registered as basic users and after admin can make them clerk and manager.
- All users must be registered as UserEnabled to use this application. Later, the admin can disable or enable it.
- For all rest endpoints receiving @RequestBody, validation check must be done and must throw custom exception if data is invalid
- All the business validations must be implemented in model classes only.
- All the database operations must be implemented on entity object only
- Do not change, add, remove any existing methods in service layer
- In Repository interfaces, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data wrapped in **ResponseEntity.**

3. Business Validations

3.1 Loan_Master Class Specifications:

- LoanId string is not null, mongodb object key
- LoanName string is not null, min 3 and max 100 characters.
- LoanAmount double is not null, min 3 and max 100 characters.
- Date DateTime is not null
- BusinessStructure enum value is not null
- Billing_Indicator enum value is not null
- Tax_Indicator enum value is not null
- ContactAddress string is not null
- Phone long is not null, min 10 and max 12 characters.
- Email string is not null, valid format
- CreatedOn auto update date and time
- ManagerRemark string
- LStatus enum value is not null

3.2 User_Master Class Specifications:

- UserId string is not null, mongodb object key
- Name string is not null, min 3 and max 100 characters.
- Email string is not null, valid format
- ContactNumber long is not null, min 10 and max 12 characters.
- Address string is not null, min 10 and max 100 characters.
- IdProofType string value is not null
- IdProofNumber string is not null
- Enabled bool

3.3 Loan_Sanctioned Class Specifications:

- Id string is not null, mongodb object key
- SanctionedAmount double is not null, > 0
- Termofloan double is not null, > 0
- PaymentStartDate DateTime
- LoanCloserDate DateTime
- MonthlyPayment double is not null, > 0.
- UpdatedOn DateTime

4. CONSIDERATIONS

A. For Role of application users 3 possible values must be used: -
   1. Customer
   2. Clerk
   3. Manager

B. For Loanstatus of loan following 4 possible Enum values must be used.

1. NotReceived = 1,
2. Received = 2,
3. Accept = 3,
4. Rejected =4,
5. Done = 5

C. For BillingStatus of loan following possible Enum values must be used.

1. Salaried_Person = 1
2. Not_Salaried_Person = 2

D. For BusinessStatus of loan following possible Enum values must be used.

1. Individual = 1
2. Organisation = 2

E. For TaxStatus of loan following possible Enum values must be used.

1. Tax_Payer = 1
2. Not_tax_Payer = 2

5. Rest Endpoints

Rest End-points to be exposed in the controller along with method details for the same to be created

5.1 CustomerController

URL Exposed	Purpose
/loan-status/{loanId} Http Method: GET Parameter 1: Integer (`loanId`) Return: `loanStatus`	Fetches details/status of applied loan
/apply-mortage Http Method: POST Parameter 1: `LoanMaster` Return: `HttpResponse status code`	Apply for new mortgage/loan
/update-mortage/{loanId} Http Method: POST Parameter 1: `LoanMaster` Return: `HttpResponse status code`	Update an existing loan application if it is not received status.

5.2 ClerkController

URL Exposed	Purpose
/get-all-application Http Method: GET Parameter 1: - Return: `<IEnumerable<LoanMaster>>`	Fetches list of applied loans not yet processed
/not-received-application Http Method: Get Return: `<IEnumerable<LoanMaster>>`	Get the list of loan allocation that is only applied by customer and not processed by clerk
/received-application Http Method: GET Parameter 1: - Return: `<IEnumerable<LoanMaster>>`	Get the list of loan allocation that is only applied by customer and processed by clerk

/process-loan/{loanId} Http Method: POST Parameter 1: Integer (`loanId`) Parameter 2: `LoanProcesstrans model` Return: `HttpResponse status code`	Process the applied loan by clerk and put the assess data for loan that can manager review, before process loan make first loan as accepted by calling method “ReceivedLoan(loanId)”.

5.3 ManagerController

URL Exposed	Purpose
/get-all-application Http Method: GET Parameter 1: - Return: `<IEnumerable<LoanMaster>>`	Fetches list of all loan application
/accept/{loanId}/{remark} Http Method: POST Parameter 1: Integer (loanId) Parameter 2: string (remark) Return: `LoanMaster`	Accept loan application by providing remarks before processing loan sanction.
/reject/{loanId}/{remark} Http Method: POST Parameter 1: Integer (loanId) Parameter 2: string (remark) Return: `LoanMaster`	Reject loan by providing remark info if not possible to pass.
/sanctioned-loan/{loanId} Http Method: GET Parameter 1: Integer(loanId) Parameter 2: `LoanApprovalViewModel model` Return: `HttpResponse status code`	Sanction loan with adding relevant info and amount.

6. Template Code Structure

6.1 Package: E-Loan

Resources

Names	Resource	Remarks	Status
Package Structure			
controller	Admin, Clerk, Customer, Manager Controller	These controllers handle all application Function, update/Edit show information and login existing users.	Partially Implemented
Startup.cs	Startup CS file	Contain all Services settings and SQL Db Configuration.	Already Implemented
Properties	launchSettings.json file	All URL Setting for API	Already Implemented

6.2 Package: E-loan.Bussinesslayer

Resources

Names	Resource	Remarks	Status
Package Structure			
Interface	ILoanClerkServices, ILoanCustomerServices, ILoanManagerServices interface	Inside all these interface files contains all business logic functions.	Already Implemented
Service	LoanClerkServices, LoanCustomerServices, LoanManagerServices CS file	Using this all class we are calling the Repository method and use it in the program and on the controller.	Partially Implemented
Repository	ILoanClerk, ILoanCustomer, ILoanManager, Repository CS file and interface.	All these interfaces and class files contain all CRUD operation code for MongoDb.	Partially Implemented
ViewModels, Enum	LoanApprovalViewModel , UserMasterRegisterViewModel.	Contain all view Domain entities for show and bind data.	Already Implemented

6.3 Package: e-loan.DataLayer

Resources

Names	Resource	Remarks	Status
Package Structure			
DataLayer	ELoanDbContext cs file	All Connection and database setting class	Already Implemented

6.4 Package: E-loan.Entities

Resources

Names	Resource	Remarks	Status
Package Structure			
Entities and Enum	UserMaster, LoanMaster, LoanProcesstrans, LoanApprovaltrans, Response CS file and BillingStatus, BusinessStatus, IdProofType, LoanStatus, TaxStatus are Enum file	All Entities/Domain attribute are used for pass the data in controller and check the various constants using enum	Already Implemented

6.5 Package: E-loan.Tests

Resources

The E-Loan.Tests project contains all test case classes and functions for code evaluation. Don't edit or change anything inside this project.

7. Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**

2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) Terminal → New Terminal.**

3. **On command prompt, cd into your project folder (cd <Your-Project-folder>).**

4. **To connect SQL server from terminal:**
   (E-Loan/sqlcmd -S localhost -U sa -P pass@word1)
   - **To create database from terminal -**
     1. **Create Database E_Loan_Db**
     2. **Go**

5. **Steps to Apply Migration(Code first approach):**
   - **Press Ctrl+C to get back to command prompt**
   - **Run following command to apply migration-**
     (E-Loan /dotnet-ef database update)

6. **To check whether migrations are applied from terminal:**
   (E-Loan /sqlcmd -S localhost -U sa -P pass@word1)

   ```
   1> Use E_Loan_Db
   2> Go
   1> Select * From __EFMigrationsHistory
   2> Go
   ```

7. **To build your project use command:**
   (E-Loan /dotnet build)

8. **To launch your application, Run the following command to run the application:**
   (E-Loan /dotnet run)

9. **This editor Auto Saves the code.**

10. **To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.**

11. **To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

12. **To run the test cases in CMD, Run the following command to test the application:**

13. **(You can run this command multiple times to identify the test case status,and refactor code to make maximum test cases passed before final submission)**

    (E-Loan.Tests/dotnet test --logger "console;verbosity=detailed").

14. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B - command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**

15. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**

16. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**