System Requirements Specification Index

For

e-Loan Application (Collaborative)

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

E-LOAN APPLICATION
System Requirements Specification

1 PROJECT ABSTRACT

E-Loan Application is Spring boot RESTful application with MySQL, where it allows customers to apply for Loan Online, and same would be processed by loan clerk and sanctioned by manager from bank.

Following is the requirement specifications:

	E-LOAN
USERS	
1	Admin
2	Manager
3	Loan Clerk
4	Customer
Admin Functionalities	
1	Can register a new Manager
2	Can register a new Clerk
3	Can get the list of all registered Managers
4	Can get the list of all registered loan clerks
5	Can Enable or Disable Manager
6	Can Enable or Disable Clerk
7	Can Enable or Disable Customers
Customer Functionalities	
1	Can register itself (*Contact Details (Contact Address, mobile, email)*
2	Can apply for Mortgage loan on a property
	While applying for loan, following information is required
	a. Loan Name (Description)
	b. Loan Application number (Auto generated)
	c. Loan Amount requested
	d. Loan Application Date
	e. Business structure (Individual/Organization)
	f. Billing Indicator (Salaried person or not)
	g. Tax indicator (Tax payer or not)
3	Can track status of Loan Application
4	Can get the details of all the loans
5	Can update the details of loan application
6	Can delete the loan application

Loan Clerk Functionalities	
1	Can list all Loan Application to be processed
2	Can Process Loan Application After Receiving and physical verification with following information added to it
	a. Acres of land
	b. Land Values in Rs
	c. Appraised by (Name of third-party appraiser)
	d. Valuation Date
	e. Address of property
	f. Suggested amount of loan that can be released on property
3	Can Update the process information
4	Can list all Loan Application Already Processed by himself
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
	a. Loan Amount Sanctioned
	b. Term of loan (Duration- in months)
	c. Payment start Date
	d. Loan closure Date (Auto calculated from Payment start date and Term)
	e. Monthly payment (Calculated)
	Formulae:
	*a. Term payment amount = (Sanctioned loan amount) * (1 + interest rate/100) ^ ((term of loan/12))*
	b. Monthly payment = (Term payment amount) / (Term of loan)
4	Can list all Loan Application Finalized(Accepted or Rejected) by himself
	Interest Rate to be used: 6%

2 ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 ADMIN CONSTRAINTS:

- While disabling the manager by admin, if manager id does not exist then operation should throw custom exception.
- While enabling the manager by admin, if manager id does not exist then operation should throw custom exception.
- While disabling the clerk by admin, if clerk id does not exist then operation should throw custom exception.
- While enabling the clerk by admin, if clerk id does not exist then operation should throw custom exception.
- While disabling the customer by admin, if customer id does not exist then operation should throw custom exception.
- While enabling the customer by admin, if customer id does not exist then operation should throw custom exception.

2.2 CUSTOMER CONSTRAINTS

- While applying loan by customer, if customer id does not exist then operation should throw custom exception.
- While applying loan by customer, if customer is disabled then operation should throw custom exception.
- While fetching loan status, if loan application id does not exist then operation should throw 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 customer is disabled then operation should throw 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 customer, if loan application is already processed by clerk, then operation should throw custom exception.
- While deleting the loan application by customer, if loan application is already processed by clerk, then operation should throw custom exception.

2.3 CLERK CONSTRAINTS

- While processing loan by loan clerk, if clerk id does not exist then operation should throw custom exception.

- While processing loan by loan clerk, if clerk is disabled then operation should throw custom exception.
- While processing loan by loan clerk, if loan application id does not exist then operation should throw custom exception.
- While processing loan by loan clerk, if loan has already been processed then operation should throw custom exception.
- While updating the processing info by loan clerk, if clerk id does not exist then operation should throw custom exception.
- While updating the processing info by loan clerk, if clerk is disabled then operation should throw custom exception.
- While updating the processing info by loan clerk, if loan application id does not exist then operation should throw custom exception.
- While updating the processing info by loan clerk, if loan application has already been finalized (accepted or rejected) by manager, then operation should throw custom exception.
- While fetching all loans processed by clerk, if clerk id does not exist then operation should throw custom exception.
- While fetching all loans processed by clerk, if clerk is disabled then operation should throw custom exception.

2.4 Manager Constraints

- While sanctioning or rejecting loan by manager, if manager id does not exist then operation should throw custom exception.
- While sanctioning or rejecting loan by manager, if manager is disabled then operation should throw 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 manager, if manager is disabled then operation should throw custom exception.

Common Constraints

- 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 (FOR ADMIN TO IMPLEMENT)

- User firstName is not null, min 3 and max 100 characters.
- User lastName is not null, min 3 and max 100 characters.
- User email is not null, min 3 and max 100 characters and in proper email format.
- User mobile is not null, min 10 and max 10 characters.
- LoanName is not null, min 3 and max 100 characters.
- LoanAmount is not null, and not 0;
- acresOfLand is not null, and not 0;
- landValue is not null, and not 0;
- suggestedAmountOfLoan is not null, and not 0;
- addressOfProperty is not null, min 3 and max 150 characters;
- loanAmountSanctioned is not null, and not 0
- termOfLoan is not null, and not 0.

4 REST ENDPOINTS

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

4.1 ADMINCONTROLLER

<table>
<tr><th>URL Exposed</th><th>Purpose</th></tr>
<tr><td>1. /admin/register-clerk<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>UserDto</td></tr><tr><td>Return</td><td>UserDto</td></tr></table></td><td>Register a loan clerk</td></tr>
<tr><td>/admin/register-manager<table><tr><td>Http Method</td><td>POST</td></tr><tr><td>Parameter 1</td><td>UserDto</td></tr><tr><td>Return</td><td>UserDto</td></tr></table></td><td>Register a manager</td></tr>
<tr><td>/admin/all-clerks<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td>List<UserDto></td></tr></table></td><td>Fetches the list of all registered loan clerks</td></tr>
<tr><td>/admin/ all-managers<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>-</td></tr><tr><td>Return</td><td>List<UserDto></td></tr></table></td><td>Fetches the list of all registered managers</td></tr>
<tr><td>/admin /disable-manager/{managerId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Integer (managerId)</td></tr><tr><td>Return</td><td>UserDto</td></tr></table></td><td>Disables the Manager</td></tr>
<tr><td>/admin /enable-manager/{managerId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Integer (managerId)</td></tr><tr><td>Return</td><td>UserDto</td></tr></table></td><td>Enables the Manager</td></tr>
<tr><td>/admin /disable-clerk/{clerkId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Integer (clerkId)</td></tr><tr><td>Return</td><td>UserDto</td></tr></table></td><td>Disables the Clerk</td></tr>
<tr><td>/admin /enable-clerk/{clerkId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Integer (clerkId)</td></tr><tr><td>Return</td><td>UserDto</td></tr></table></td><td>Enables the Clerk</td></tr>
<tr><td>/admin /disable-customer/{customerId}<table><tr><td>Http Method</td><td>GET</td></tr><tr><td>Parameter 1</td><td>Integer (customerId)</td></tr><tr><td>Return</td><td>UserDto</td></tr></table></td><td>Disables the Customer</td></tr>
<tr><td>/admin /enable-customer/{customerId}</td><td>Enables the Customer</td></tr>
</table>

Http Method	GET
Parameter 1	Integer (customerId)
Return	UserDto

4.2 CUSTOMERCONTROLLER

URL Exposed		Purpose
/customer/register		Register a new user
Http Method	POST	
Parameter 1	UserDto	
Return	UserDto	
/customer/apply-loan/{customerId}		Apply for loan
Http Method	POST	
Parameter 1	Integer(customerId)	
Parameter 2	LoanDto	
Return	LoanOutputDto	
/customer/loan-status/{loanAppId}		Fetches details of applied loan
Http Method	GET	
Parameter 1	Integer (loanAppId)	
Return	LoanOutputDto	
/customer/loan-status-all/{customerId}		Fetches details of all loans applied by a customer
Http Method	GET	
Parameter 1	Integer (customerId)	
Return	List<LoanOutputDto>	
/customer/loan-update/{loanAppId}		Updated the details of applied loan
Http Method	PUT	
Parameter 1	Integer (loanAppId)	
Parameter 2	LoanDto	
Return	LoanOutputDto	
/customer/loan-delete/{loanAppId}		Deletes the loan application
Http Method	DELETE	
Parameter 1	Integer (loanAppId)	
Return	LoanOutputDto	

4.3 CLERKCONTROLLER

URL Exposed		Purpose
/clerk/all-applied		Fetches list of applied loans not yet processed
Http Method	GET	
Parameter 1	-	
Return	List<LoanOutputDto>	

/clerk/process/{clerkId}/{loanAppId} Http Method: POST Parameter 1: Integer(clerkId) Parameter 2: Integer(loanAppId) Parameter 3: ProcessingDto Return: ProcessingDto	Process the loan to forward to Manager
/clerk/process-update/{clerkId}/{loanAppId} Http Method: PUT Parameter 1: Integer(clerkId) Parameter 2: Integer(loanAppId) Parameter 3: ProcessingDto Return: ProcessingDto	Update the processing information
/clerk/all-processed/{clerkId} Http Method: GET Parameter 1: Integer (clerkId) Return: List<LoanOutputDto>	Fetches list of processed loans by a particular clerk

4.4 MANAGERCONTROLLER

URL Exposed	Purpose
/manager/all-processed Http Method: GET Parameter 1: - Return: List<LoanOutputDto>	Fetches list of applied loans processed by loan clerk
/manager/reject-loan/{managerId}/{loanAppId} Http Method: POST Parameter 1: Integer (managerId) Parameter 2: Integer (loanAppId) Parameter 3: RejectDto Return: RejectDto	Reject loan application by providing remark
/manager/sanction-loan/{managerId}/{loanAppId} Http Method: POST Parameter 1: Integer (managerId) Parameter 2: Integer (loanAppId) Parameter 3: SanctionDto Return: SanctionDto	Sanction loan by providing sanction info
/manager/all-finalized/{managerId} Http Method: GET Parameter 1: Integer(managerId) Return: List<LoanOutputDto>	Fetches list of all finalized loans finalized by a particular manager

5 TEMPLATE CODE STRUCTURE

5.1 PACKAGE: COM.IIHT.TRAINING.ELOAN

Resources

ELoanApplication(Class)	This is the SpringBoot starter class of the application.	Already Implemented

5.2 PACKAGE: COM.IIHT.TRAINING.ELOAN.ENTITY

Resources

Class/Interface	Description	Status
Users (class)	○ Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. ○ Map this class with **user** table. ○ Generate the **Id** using the **IDENTITY** strategy	Partially implemented.
Loan(class)	○ This is class is partially implemented. ○ Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. ○ Map this class with **loan** table. ○ Generate the **Id** using the **IDENTITY** strategy	Partially implemented.
ProcessingInfo(class)	○ This is class is partially implemented. ○ Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. ○ Map this class with **process** table. ○ Generate the **Id** using the **IDENTITY** strategy	Partially implemented.

SanctionInfo(class)	○ This is class is partially implemented. ○ Annotate this class with proper annotation to declare it as an entity class with **Id** as primary key. ○ Map this class with **sanction** table. ○ Generate the **Id** using the **IDENTITY** strategy	Partially implemented.

5.3 PACKAGE: COM.IIHT.TRAINING.ELOAN.MODEL

Resources

Class/Interface	Description	Status
LoanDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attribute of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
LoanOutputDto (class)	To be used for assembling all information about loan to share with client	Already implemented.
ProcessingDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attribute of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.
RejectDto (class)	To be used for providing rejection remarks	Already implemented.
SanctionDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attribute of this class.	Partially implemented.

	(Refer **Business Validation** section for validation rules).	
SanctionOutputDto (class)	To be used for reverting the Sanction info back to cleint	Already implemented.
UserDto (class)	Use appropriate annotations from the **Java Bean Validation API** for validating attribute of this class. (Refer **Business Validation** section for validation rules).	Partially implemented.

5.4 PACKAGE: COM.IIHT.TRAINING.ELOAN.MODEL.EXCEPTION

Resources

Class/Interface	Description	Status
ExceptionResponse (class)	Object of this class is supposed to be returned in case of exception through exception handlers	Already implemented.

5.5 PACKAGE: COM.IIHT.TRAINING.ELOAN.REPOSITORY

Resources

Class/Interface	Description	Status
LoanRepository (interface)	1. Repository interface exposing CRUD functionality for **Loan** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented
ProcessingInfoRepository (interface)	1. Repository interface exposing CRUD functionality for **ProcessingInfo** Entity.	Partially implemented

	2. You can go ahead and add any custom methods as per requirements	
SanctionInfoRepository (interface)	1. Repository interface exposing CRUD functionality for **SanctionInfo** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented
UserRepository (interface)	1. Repository interface exposing CRUD functionality for **Users** Entity. 2. You can go ahead and add any custom methods as per requirements	Partially implemented

5.6 PACKAGE: COM.IIHT.TRAINING.ELOAN.SERVICE

Resources

Class/Interface	Description	Status
AdminService (interface)	Interface to expose method signatures for admin related functionality. Do not modify, add or delete any method	Already implemented.
CustomerService (interface)	Interface to expose method signatures for customer related functionality.	Already implemented.

	Do not modify, add or delete any method	
ClerkService (interface)	Interface to expose method signatures for clerk related functionality. Do not modify, add or delete any method	Already implemented.
ManagerService (interface)	Interface to expose method signatures for manager related functionality. Do not modify, add or delete any method	Already implemented.

5.7 PACKAGE: COM.IIHT.TRAINING.ELOAN.SERVICE.IMPL

Resources

Class/Interface	Description	Status
AdminServiceImpl (class)	• Implements **AdminService**. Contains template method implementation. • Need to provide implementation for admin related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.
CustomerServiceImpl (class)	• Implements **CustomerService**. Contains	To be implemented.

	template method implementation. • Need to provide implementation for admin related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	
ClerkServiceImpl (class)	• Implements **ClerkService**. Contains template method implementation. • Need to provide implementation for admin related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.
ManagerServiceImpl (class)	• Implements **ManagerService**. Contains template method implementation. • Need to provide implementation for admin related functionalities • Add required repository dependency • Do not modify, add or delete any method signature	To be implemented.

5.8 PACKAGE: COM.IIHT.TRAINING.ELOAN.EXCEPTION

Resources

Class/Interface	Description	Status
GlobalHandler (class)	• RestControllerAdvice Class for defining global exception handlers. • Contains Exception Handler for **InvalidDataException** class. • Use this as a reference for creating exception handler for other custom exception classes	Partially implemented.

Class/Interface	Description	Status
AlreadyFinalizedException (Class)	• Custom Exception to be thrown when trying reject, sanction or update the process info of a loan which is already finalized. • Need to create Exception Handler for same wherever needed (local or global)	Already created.
AlreadyProcessedException (Class)	• Custom Exception to be thrown when trying update loan application or trying to add process info for already processed loans. • Need to create Exception Handler for same wherever needed (local or global)	Already created.

ClerkDisabledException (Class)	• Custom Exception to be thrown when trying to perform any activity based on clerk id, which has been disabled by admin • Need to create Exception Handler for same wherever needed (local or global)	Already created.
CustomerDisabledException (Class)	• Custom Exception to be thrown when trying to perform any activity based on customer id, which has been disabled by admin • Need to create Exception Handler for same wherever needed (local or global)	Already created.
ManagerDisabledException (Class)	• Custom Exception to be thrown when trying to perform any activity based on manager id, which has been disabled by admin • Need to create Exception Handler for same wherever needed (local or global)	Already created.
ManagerNotFoundException (Class)	• Custom Exception to be thrown when trying to perform any activity based on manager id, which is not registered • Need to create Exception Handler for same wherever needed (local or global)	Already created.

ClerkNotFoundException (Class)	• Custom Exception to be thrown when trying to perform any activity based on clerk id, which is not registered • Need to create Exception Handler for same wherever needed (local or global)	Already created.
CustomerNotFoundException (Class)	• Custom Exception to be thrown when trying to perform any activity based on customer id, which is not registered • Need to create Exception Handler for same wherever needed (local or global)	Already created.
LoanNotFoundException (Class)	• Custom Exception to be thrown when trying to perform any activity based on loan id, which is not applied • Need to create Exception Handler for same wherever needed (local or global)	Already created.

5.9 PACKAGE: COM.IIHT.TRAINING.ELOAN.CONTROLLER

Resources

Class/Interface	Description	Status
AdminController (Class)	• Controller class to expose all rest-endpoints for admin related activities.	To be implemented

	• May also contain local exception handler methods	
CustomerController (Class)	• Controller class to expose all rest-endpoints for customer related activities. • May also contain local exception handler methods	To be implemented
ClerkController (Class)	• Controller class to expose all rest-endpoints for clerk related activities. • May also contain local exception handler methods	To be implemented
ManagerController (Class)	• Controller class to expose all rest-endpoints for manager related activities. • May also contain local exception handler methods	To be implemented

6 CONSIDERATIONS

A. For Role of Users three possible values must be used
   1. Customer
   2. Clerk
   3. Manager
B. For status of loan following 4 possible values must be used

Applied
Processed
Sanctioned
Rejected

7 EXECUTION STEPS TO FOLLOW

1. **Open the tools terminal from Terminal option**
2. **On command prompt, cd into your project folder (cd <Your-name-hashkey>)**
3. **To build your project and run test cases use command:**

   mvn clean package
4. **To launch your application, move into target folder (cd target). Run the following command to run the application:**

   java -jar eloan-0.0.1-SNAPSHOT.jar
5. **For accessing MySQL open mysql terminal from Terminal option**