NIIT

Online Banking System

Project Document 2023

UnderGuidence Of : Mr.Rakesh Srivastava

Submitted By: Saurabh Lengure, Mohd Saif, Yash Barsker

Group Number: 2

Online Banking System

Introduction:-

An online banking system is a digital platform that allows customers to access and manage their bank accounts through the Internet. Online banking systems provide a wide range of features and functionalities that enable customers to perform banking transactions, such as transferring funds, paying bills, checking account balances, and applying for loans and credit cards, from the comfort of their homes. The system is designed to provide all the necessary features to operate and manage a bank's core banking operations.

SOFTWARE REQUIREMENTS-

- MogoDB, MySQL
- Eclipse(J2EE)
- STS
- Postman
- GitLAB

Use case 1: Login Functionality

Description: The system provides complete login functionality, where users can securely log in to the system using their username and password. The login page is accessible from the online banking portal or from the core banking system Website.We will use role based access control and Configuring Single sign-On authentication based on JWT where given below roles can get token to access all the given below functionality.

Role: Admin, User, Manager.

Pre-conditions: The user should have a valid account with the bank and must have a valid username and password.

Post-conditions: The user will be able to access their account

Login Microservice

1.1 Login Controller: The REST Controller handles login and logout requests from the client and sends the request to the service layer method.

Username and password: The traditional method of authentication, where users enter a unique username and password to gain access to their accounts.

```
@PostMapping("/user/CreateUser")

RegisterUser(@RequestBody User user){

service.create()

}
```

```
@PutMapping("/user/updatePassword")

UpdatePassword(@RequestBody Int Password){

service.update()

}
```

@GetMapping("/user/validateUser")

LoginUser(@RequestBody UserId, @RequestBody Password){

service.validate()

}

1.2 Login Service: The Service layer verifies the user's credentials and generates a session for successful login requests.

The login page is the first screen that the user encounters when accessing the online banking portal. The login page should be designed to be user-friendly and intuitive, enabling users to quickly and easily access their accounts. The login page flow diagram should include the following steps:

The system confirms the account is activated and directs the user to the login page.

i. If User want to create New Account Then add all the details like UserId, User First Name, User Last Name, User Mobile Number, User Email Id, User Pin, User Password

ii. The user enters their username and password and clicks on the login button.

iii. If the username and password are correct, the user is directed to the dashboard page.

iv. If the username or password is incorrect, an error message is displayed, and the user is prompted to enter the correct credentials.

v. If User can update the password option click and they can easily update the password and then easily login to update password.

create(User user){

Repo.save(user);

}

getUserByUserNameAndPassword(userName, Password){

Validate UserId & Password.

}

Update(password){

User user = new User();

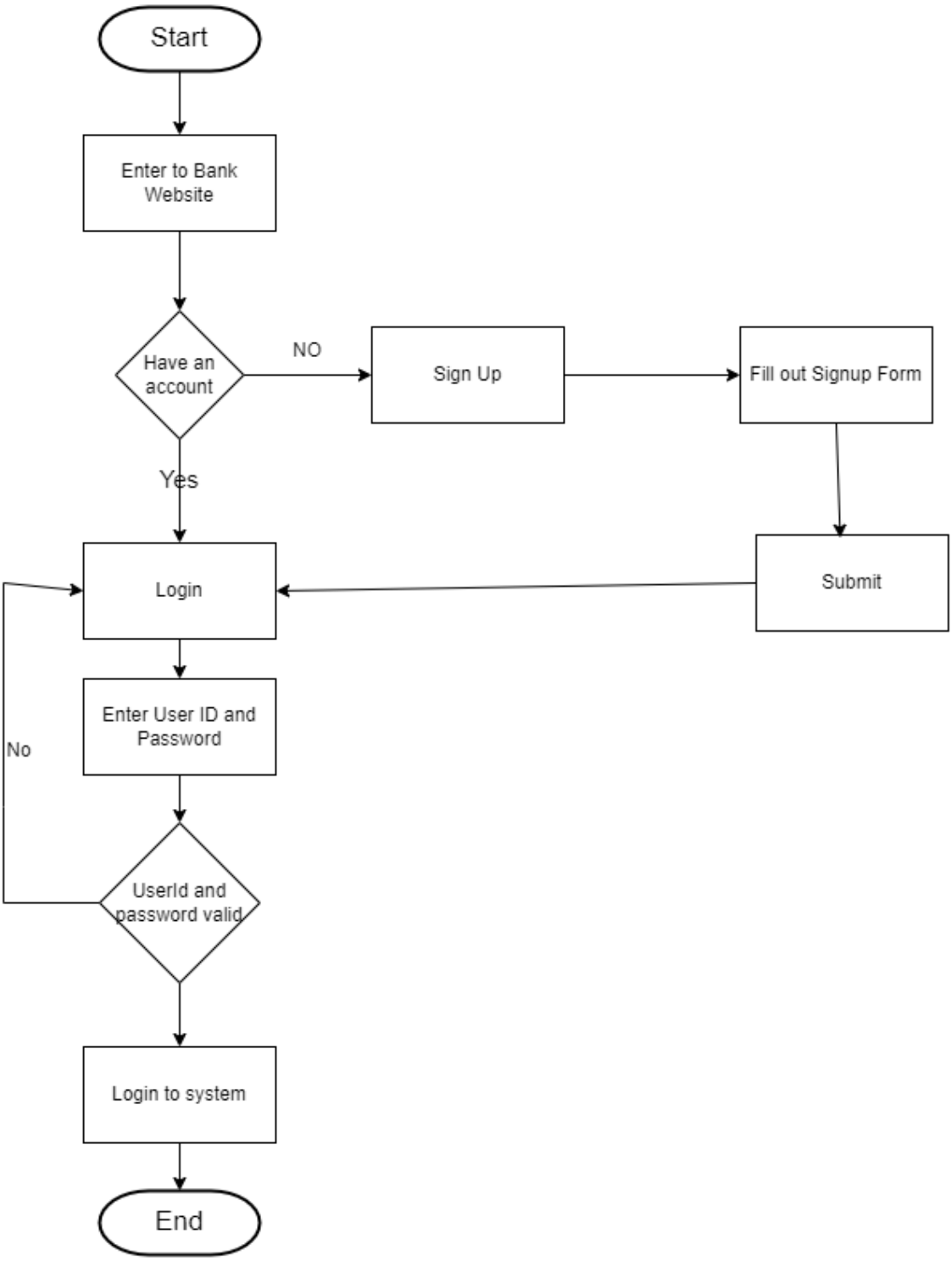
Repo.save(user.setPassword(password));

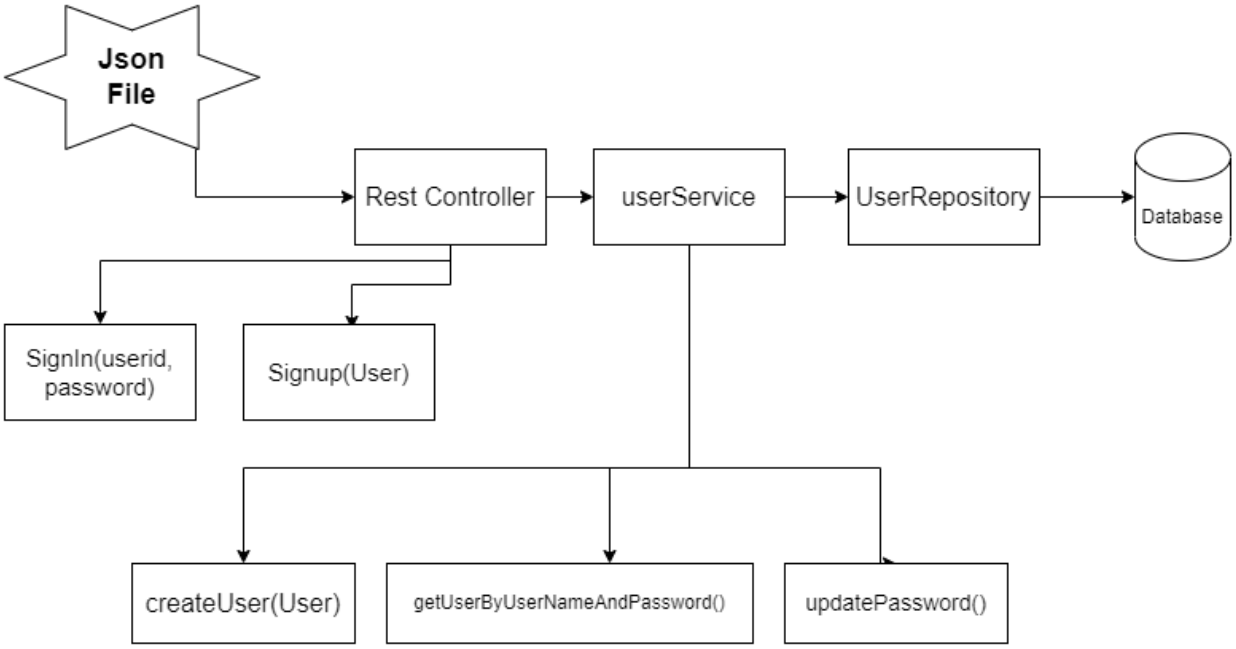
}

1.3 User Repository: The database that stores user username and password, and user roles.

1.4 User Model: UserId, UserFirstName, UserLastName, UserMobileNumber, UserEmailId, UserPin, UserPassword, CurrentBalance, AccountNumber.

Login Service Flowchart





Conclusion:

The login functionality is a critical feature of the online banking portal, as it is the gateway to the system and plays a crucial role in ensuring the security of customer data and transactions. The login functionality should be designed to be user-friendly and intuitive, with robust security measures such as multi-factor authentication and Captcha. The use case diagram and flow chart should be designed to provide a clear understanding of the login process and the actors involved. Finally, the class design should be implemented using POJO/Entity classes to represent the user and their credentials.

Use Case 2: Transaction Management

Description:

The system provides transaction management, where users can manage all types of transactions, including deposits, withdrawals, and transfers. Customers can view their transaction history and perform transactions from their online banking portal.

Transaction management refers to the process of recording, processing, and managing financial transactions in a banking system. In online banking portals, this includes the ability to initiate, authorize, track, and reconcile transactions such as deposits, withdrawals, transfers, and payments, among others. Effective transaction management is critical for ensuring accuracy, compliance, and customer satisfaction.

Pre-conditions: The user must have a logged-in into account to perform the transaction

Post-conditions: The transaction should be successfully completed, and the user's account balance should be updated.

Transaction Microservice:

2.1 Transaction Controller: The REST Controller that handles requests for transferring funds between accounts. it sends a request to the service layer method to perform a transaction.

Basic Flow:

1. Customer selects the transaction type (deposit, withdrawal, transfer, or payment).

2. System prompts the customer to enter the transaction amount and select the account(s) involved in the transaction.

3. Customer confirms the transaction details.

4. System validates the transaction and updates the account balances.

5. System displays a confirmation message to the customer and logs the transaction in the system.

Alternate Flow 1: Insufficient Funds

1. Customer attempts to withdraw or transfer an amount that exceeds their available balance.

2. System displays an error message and prompts the customer to enter a different amount or cancel the transaction.

Alternate Flow 2: Invalid Account

1.Customer attempts to perform a transaction involving an account that does not exist or is not associated with their profile.

2.System displays an error message and prompts the customer to enter a different account or cancel the transaction.

Alternate Flow 3: Transaction History

1.This use case allows users to view transaction history for their accounts or for accounts they manage.

2.User is logged in and granted access to transaction history features based on their role.

3.User is able to view transaction history for the selected account(s).

```
@GetMapping("/user/GetBalance")

ViewBalanceController(){
service.ViewBalance()
}
```

```
@GetMapping("/user/{Long AccountNo}/{Amount}")

TransferController(@PathVariable("AccountNo") Long AccountNo,
      @PathVariable("Amount")  Int Amount){

service.Transfer()
}
```

2.2 Transaction Service: The Service layer has business logic that verifies the user's role and account balance before processing the transaction request .after a successful transaction user balance should be updated.

```
ViewBalance(){

UserRepo.GetCurrentBalance();
}
```

```
Transfer(Long ToAccountNo, Int Amount, Int userId){

Call.....>>Withdraw(Amount)

}
```

Withdraw(int BalanceToWithdraw){

ValidatePin
After Validation of Pin Successfully
check Wheather UserRepo.getCurrentBalance() is greater than BalanceToWithdraw or Not.

}

```
Deposit(Int Amount, Int userId){
  User user = Repo.findByUserId(userId);
  user.setCurrentBalance(user.getCurrentBalance + Amount);
  Repo.save(user);
}
```

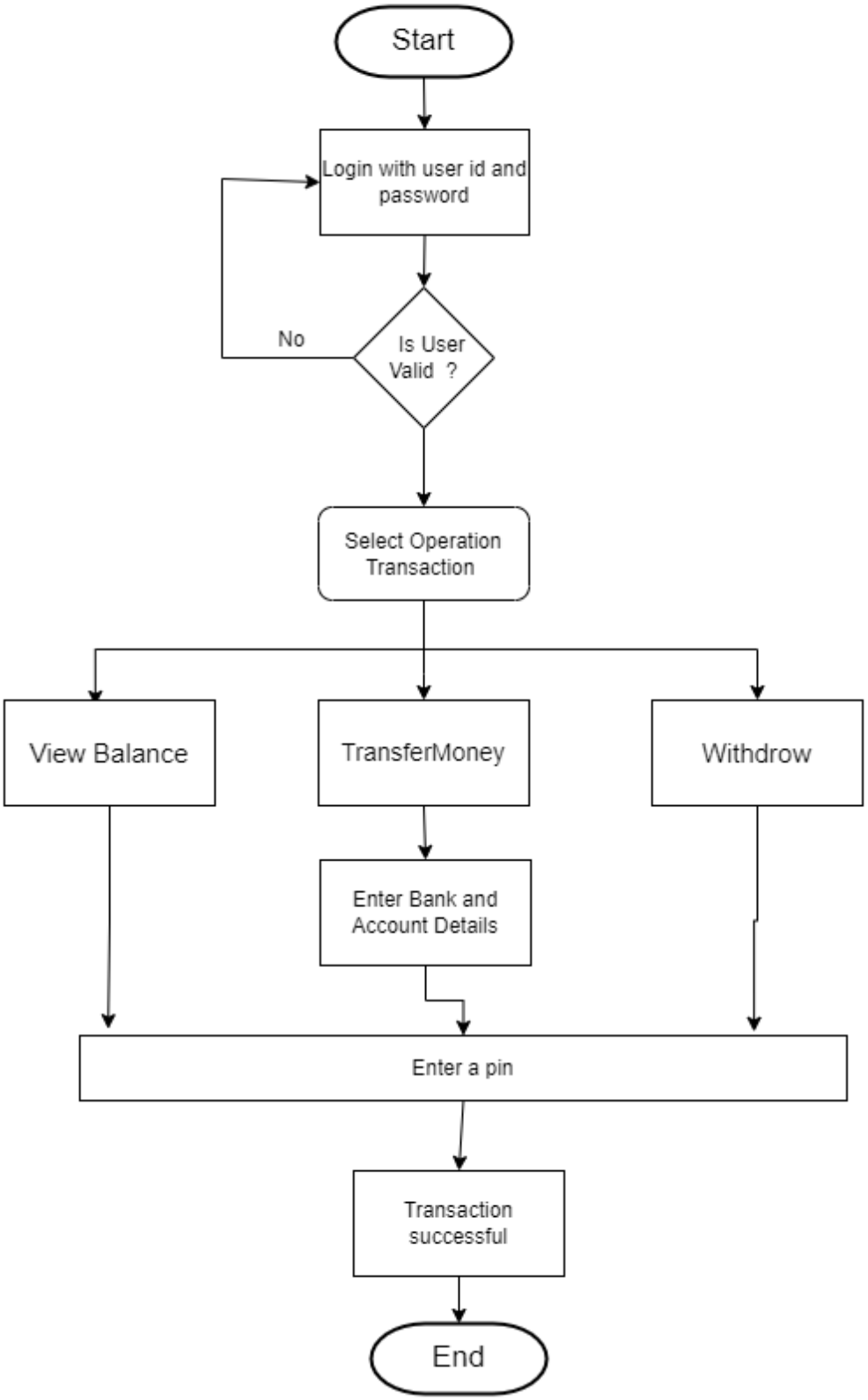
2.3 Transaction Repository: The database that stores account information and helps to perform different database operations

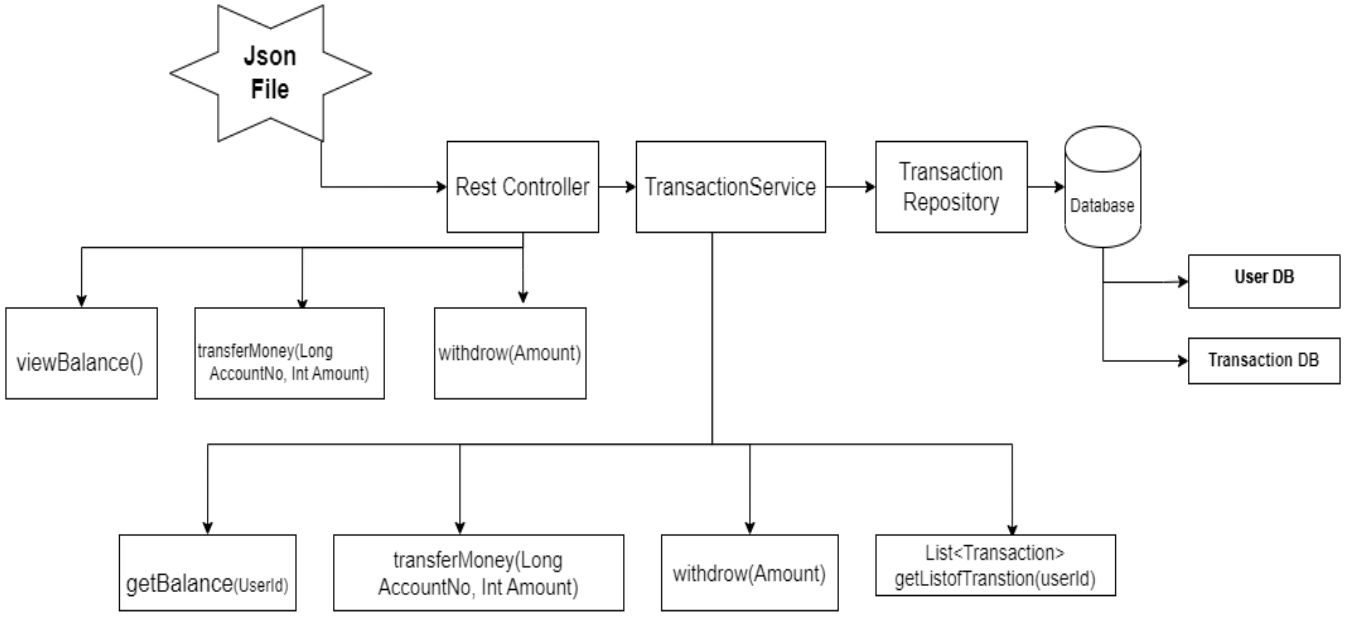
2.4 User Repository: Use Rest Template to fetch the details of the User..

2.5 Transaction Model: TransactionId, UserId (@Mapping), ToAccountNumber, AmountTransfer, TransferDate, Description(TransactionType).

2.6 User Model: UserId(@Mapping), UserFirstName, UserLastName, UserMobileNumber, UserEmailId, UserPin, UserPassword, CurrentBalance, AccountNumber.

Transaction Manager Flowchart





Conclusion: The transaction manager is a critical component of an internet banking system that enables customers to perform transactions online, improving convenience and reducing costs. However, it's crucial to implement strong security measures to ensure that transactions are secure and that customers' personal and financial information is protected from cyber threats.

Use case 3: Loans service

Description: In a core banking or online banking portal, loans refer to the financial products that allow customers to borrow money from the bank for various purposes, such as buying a home, starting a business, or funding education. The use of such loans may require a customer to go through a credit check and provide collateral. The portal enables customers to apply for loans, view their loan status, make loan payments, and manage their loan accounts. The bank uses the portal to process loan applications, disburse funds, and monitor loan repayments

Pre-conditions: The user should meet the eligibility criteria set by the bank for the specific loan

Post-conditions: The loan should be approved and disbursed to the user's account, and the user should be able to make repayments

Basic Flow:

1.Customer logs in to the online banking portal.

2.Customer selects the loan product they are interested in and completes the loan application form.

3.The bank's loan officer receives the loan application and reviews the customer's credit history, collateral, and other eligibility criteria.

4.If the loan application is approved, the bank sends a loan agreement to the customer through the portal.

5.The customer reviews loan agreement online.

6.The bank disburses the loan amount to the customer's account through the portal.

7.The customer can view the loan account details, payment schedule, and payment history through the portal.

Alternate Flow:

1.If the loan application is rejected, the bank's loan officer sends a notification to the customer through the portal explaining the reason for rejection..

Loan Microservice:

3.1 Loan Controller: The REST Controller that handles requests for processing loan applications and calls particular methods from service layer to perform operation

```
@PostMapping
ApplyLoanController(@RequestBody Loan loan){
service.ApplyLoan()
}
```

```
@GetMapping("/user/{UserId}")
TrackLoanStatusController(@PathVariable("userId") Int userId){
    service.TrackLoanDetails()
}
```

3.2 Loan Service: The Service layer that verifies the user's role and credit score before approving or denying the loan application and generates loan reports based on user-specified criteria.

```
Apply Loan(Loan loan){

}
```

```
TrackLoanDetails(Int userId){
```

CallListOfLoan(userId)

}

ListOfLoanDetails(Int UserId){

}

3.3 User Repository: Use Rest Template to fetch the details of the User.

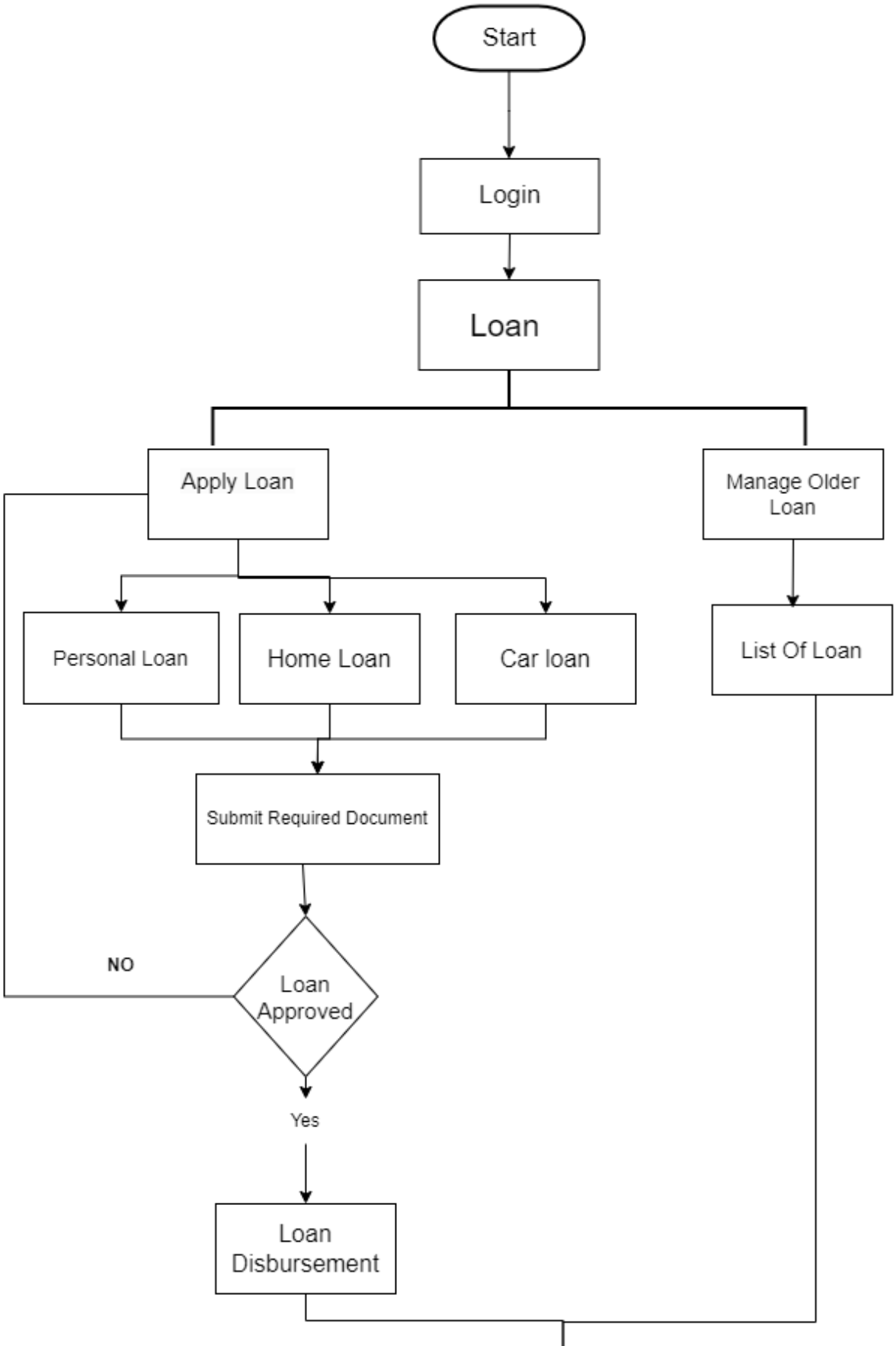
3.4 Loan Repository: The database that stores loan account information helps to perform different database operations related to loan

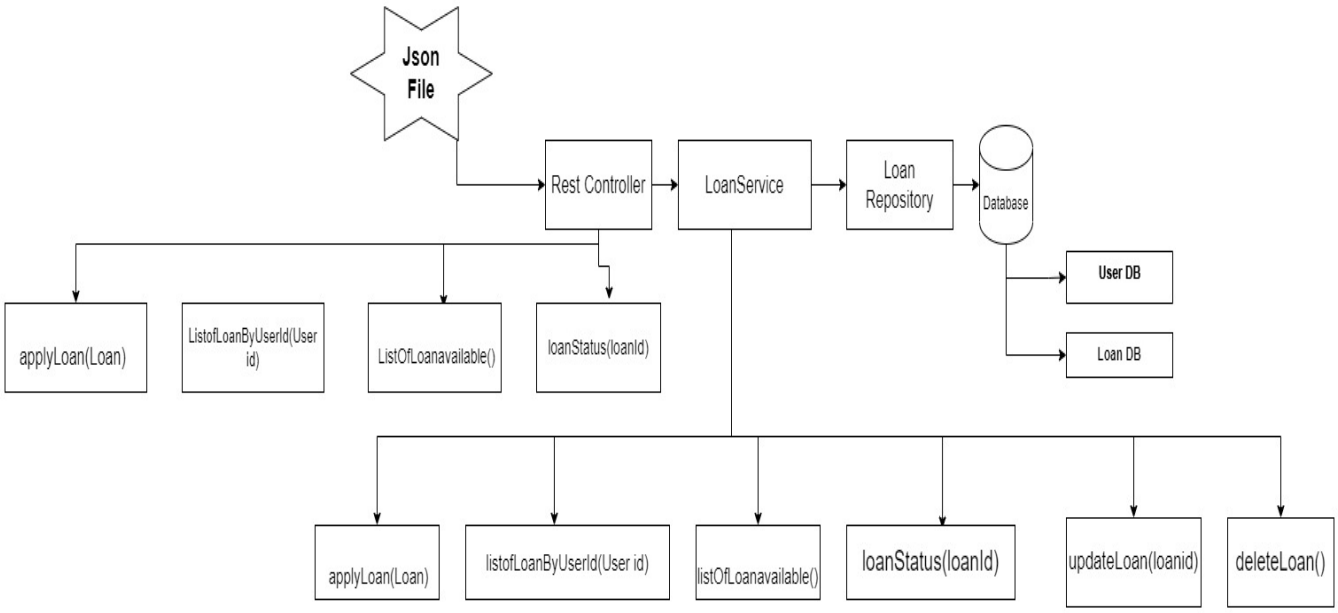
3.5 Loan Model: userId(@mapping), LoanId, List<LoanType> loanType, principalLoanAmount, interestOnLoan, loanStatus, DisbusmentDate Document document.

3.6 Document Model: AdharCard Number, PanCard Number, Salary.

3.7 User Model: UserId(@Mapping), UserFirstName, UserLastName, UserMobileNumber, UserEmailId, UserPin, UserPassword, CurrentBalance, AccountNumber.

Loan FlowChart





Conclusion: The loan process is a crucial component of an internet banking project. It enables customers to apply for loans online, track the status of their loan applications, and make loan payments conveniently from their accounts. At the same time, it allows banks to manage loan applications efficiently and provide faster loan approvals, reducing the turnaround time for loan disbursement.

However, the loan process involves several steps, including loan eligibility checks, loan approval, loan disbursement, loan repayment, and loan closure, that need to be carried out with due diligence and attention to detail. Banks must ensure that all regulatory requirements are met, that customers are informed about the terms and conditions of the loan, and that their personal and financial information is protected throughout the loan process.

Use case 4: Lockers Availability

Description: Lockers are secure storage units offered by banks to their customers for storing valuable items like documents, jewellery, and cash. They are typically available at bank branches and can be accessed during business hours. Lockers provides a added layer of security for customers and are a popular service offered by the banks through their core banking or online banking portals.

Pre-conditions: The user should have a valid account with the bank, and there should be an available locker to provide to the user.

Post-conditions: The user should be able to book the locker and access it

.

Locker microservice

4.1 Locker Controller: The REST Controller that handles requests for checking locker availability and assigning lockers to customers.

```
@PostMapping
ApplyForLockerController(@RequestBody Int userId, @RequestBody Int LockerSize){
service.ApplyForLocker();
}
```

4.2 Locker Service: The Service layer that verifies the user's role and locker availability before assigning a locker to a customer. And update data in the database

ApplyForLocker(UserId, LockerSize){

If Repo.NoOfAvalability > Zero And LockerSize equals to

Repo.LockerSize. Then Locker Assigned to User.

}

4.3 User Repository: Use Rest Template to fetch the details of the User.

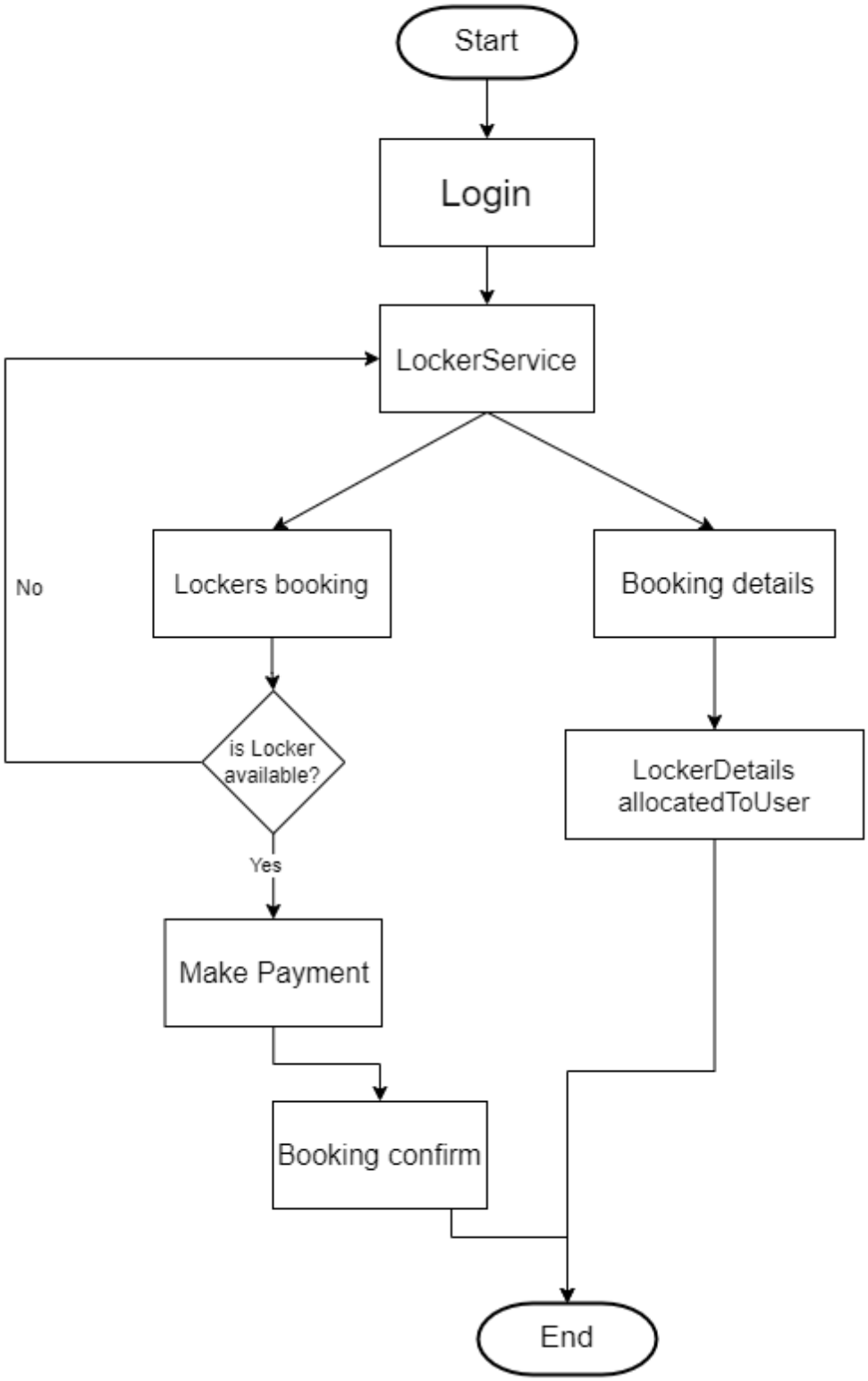
4.4 Locker Repository: The database that stores locker information perform crud operation related to locker modal

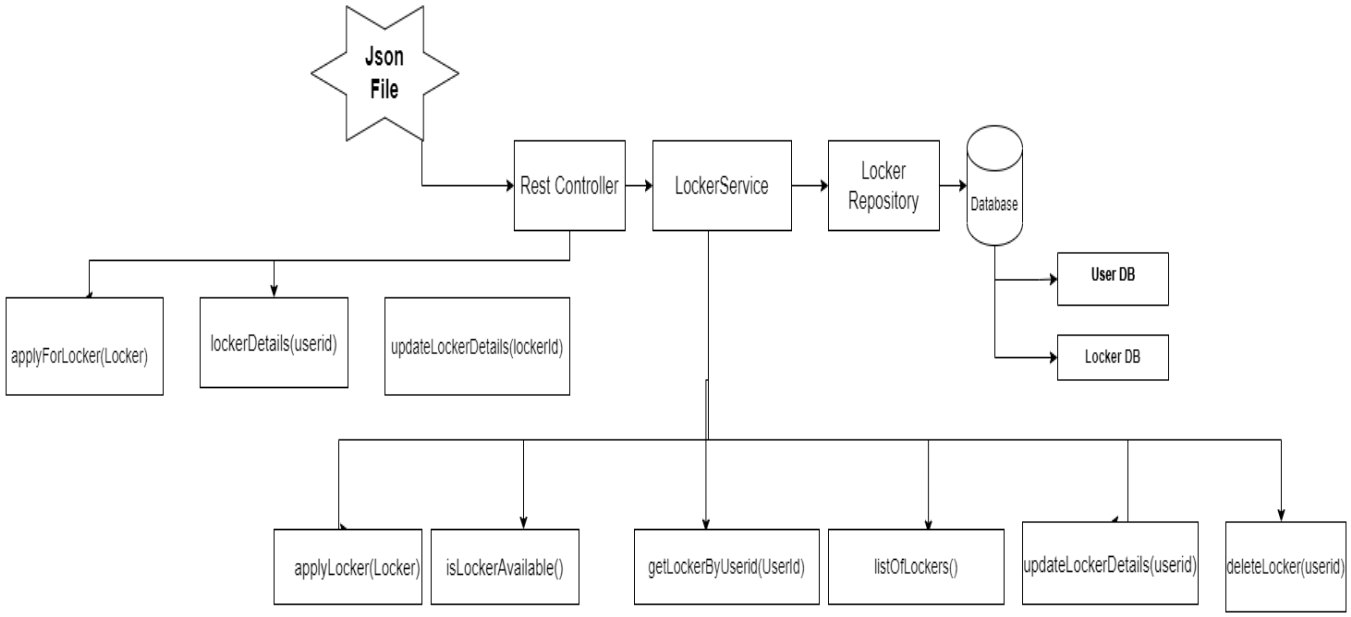
4.5 NewLockerModel: UseId(@Mapping), Locker locker.

4.6 LockerModel: noOfAvailabilty, LockerId, LockerSize, IssuedDate.

4.7 User Model: UserId(@Mapping), UserFirstName, UserLastName, UserMobileNumber, UserEmailId, UserPin, UserPassword, CurrentBalance, AccountNumber.

Locker Flowchart





Conclusion: locker availability is a valuable feature of an online banking system that can provide customers with a secure and convenient way to store their valuables. By offering locker availability and related services, banks can improve their overall service offerings, generate additional revenue, and build customer loyalty.

Use case 5:Credit card

Description: Credit cards are financial products that allow users to borrow money up to a certain limit, which they must pay back with interest. They provide convenient access to credit and are widely accepted for making purchases in stores, online, and over the phone. Credit cards can also offer various rewards, benefits, and insurance protections, depending on the issuer and the type of card.

Use Case Name: Credit Card Management

Actors: Customer, Credit Card Issuer

Pre-conditions: The user should have a valid account with the bank, meet the eligibility criteria for the specific credit card product and submit all required documents and information.

Post-conditions: The credit card should be approved and issued to the user, and the user should be able to use it for purchases or cash withdrawals up to the approved credit limit

Basic Flow:

1.The customer logs in to the online banking portal.

2.The customer can view their credit card account balance, available credit, and recent transactions through the portal.

3.The customer can apply for additional credit cards or upgrade their existing card through the portal.

4.The credit card issuer can monitor the customer's credit card account status and send reminders for missed payments.

5.The credit card issuer can adjust the credit limit or interest rate based on the customer's creditworthiness and usage.

Credit card microservice:

5.1 Credit Card Controller: The REST Controller handles requests for processing credit card applications and generating credit card reports.

```
@PostMapping
ApplyForNewCreditCardController(@RequestBody CreditCard creditCard){
service.ApplyForNewCreditCard()
}
```

```
@GetMapping("/User/{userId}")
TrackCreditCardStatusController(@PathVariable("userId") Int userId){
    service.TrackCreditCardDetails()
}
```

5.2 Credit Card Service: The Service layer that verifies the user's role and credit score before approving or denying the credit card application

```
ApplyForNewCreditCard(CreditCard creditCard){

}
```

```
TrackCreditCardDetails(Int userId){

CallListOfCreditCardDetails(userId)
}
```

ListOfCreditCardDetails(Int UserId){

}

5.3 User Repository: Use Rest Template to fetch the details of the User.

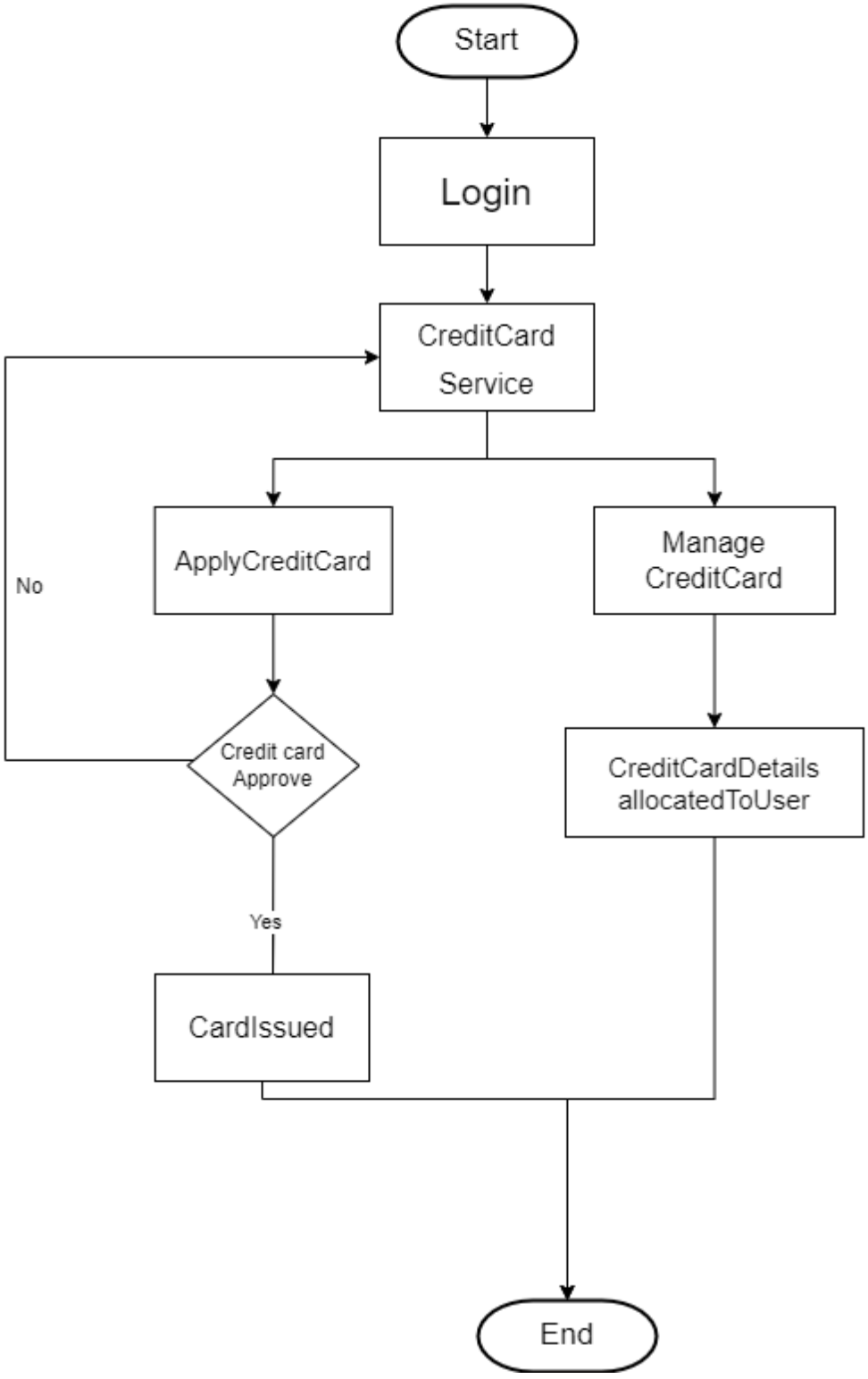
5.4 Credit Card Repository: The database that stores credit card account information.

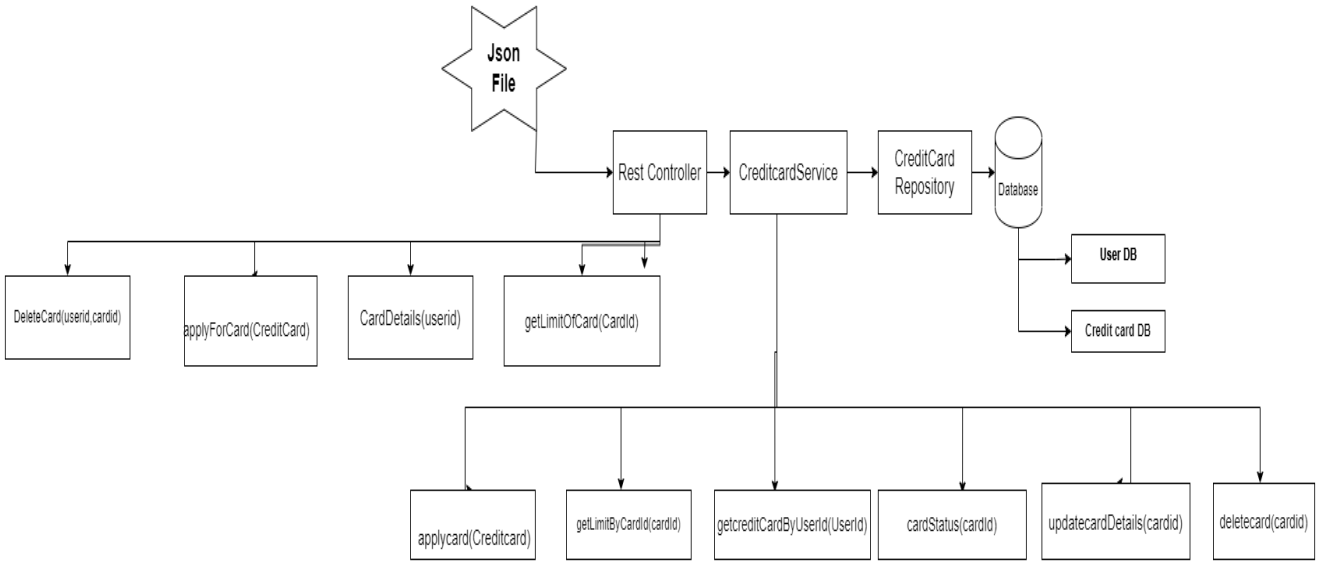
5.5 Credit Card Model: userId(@mapping), Credit CardId, List<CreditCardType> CreditCardType, Document document.

5.6 Document Model: AdharCardNumber, PanCardNumber, Salary.

5.7 User Model: UserId(@Mapping), UserFirstName, UserLastName, UserMobileNumber, UserEmailId, UserPin, UserPassword, CurrentBalance, AccountNumber..

CreditCard Flowchart





Conclusion: Credit card functionality is a vital feature of an online banking system that enables customers to access credit and make payments securely. The credit card flow in an online banking project involves several steps, including application, approval, card issuance, activation, usage, and payment. By integrating credit card functionality into the online banking system, banks can provide customers with a seamless experience, increase customer satisfaction, and generate revenue from interest and fees.

Use case 6: Gift Card

Description: The system provides gift card management, where managers can manage gift card accounts Customers can purchase gift cards from their online banking portal, view their gift card accounts and redeem a gift.

Purchase Gift Card: The customer purchases a gift card for themselves or someone else

Select Gift Card: The customer selects the type of gift card they want to purchase, such as a birthday card or a holiday card

Gift Cards Microservice:

Gift cards are prepaid cards that can be used to purchase goods or services from a particular retailer or group of retailers. They are a popular gift option and offer convenience and flexibility for both the giver and the recipient. Gift cards can be purchased in-store or online, and may have expiration dates or restrictions on usage.

Use Case Name: Gift Card Management

Actors: Customer, Retailer, Gift Card Issuer

Preconditions:

•The customer must have a valid gift card issued by a participating retailer or gift card issuer.

•The bank must offer gift card management services through the online banking portal.

Basic Flow:

1.The customer logs in to the online banking portal.

2.The customer can view their gift card balance and transaction history through the portal.

3.The customer can reload their gift card balance through the portal using different payment methods.

4.The customer can purchase gift cards for themselves or others through the portal.

5.The retailer can process gift card payments and verify gift card balances through the portal.

7.The gift card issuer can monitor gift card balances, expiration dates, and fraud prevention processes through the portal.

Post-conditions:

•The customer can manage their gift card balance and transactions through the online banking portal.

•The participating retailer can process gift card payments through the portal.

•The gift card issuer can monitor gift card balances, transactions processes through the portal.

6.1 Gift Card Controller: The REST Controller that handles requests for purchasing gift cards, checking gift card balances, and redeeming gift cards.

```
@PostMapping
ApplyForGiftCardController(){
service.ApplyForGiftCard();
}
```

@GetMapping

ListOfGiftCardsController(){

service.ListOfGiftCards();

}

6.2 Gift Card Service: The Service layer that verifies the user's role and gift card balance before processing the gift card request and generates gift card reports based on user-specified criteria.

ApplyForGiftCard(Int UserId,Int GiftCardAmount){

Call.....>>Withdraw(GiftCardAmount)

}

Withdraw(int BalanceToWithdraw){

check Wheather Repo.getBalance() is greater than BalanceToWithdraw or Not.

}

```
ListOfGiftCards(int UserId){
    Repo.findByUserId();
}
```

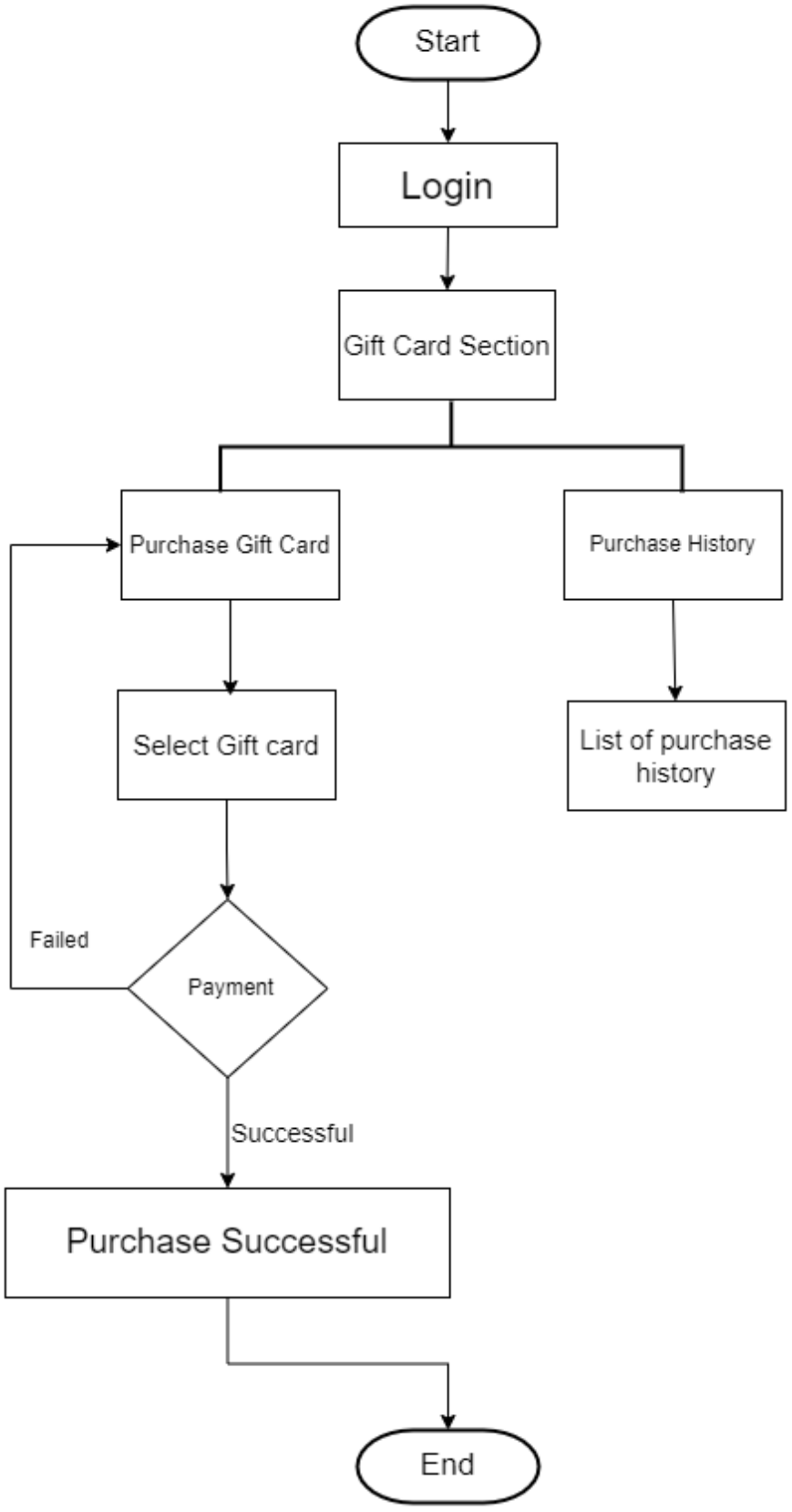
6.3 User Repository: Use RestTemplate to fetch the details of the User.

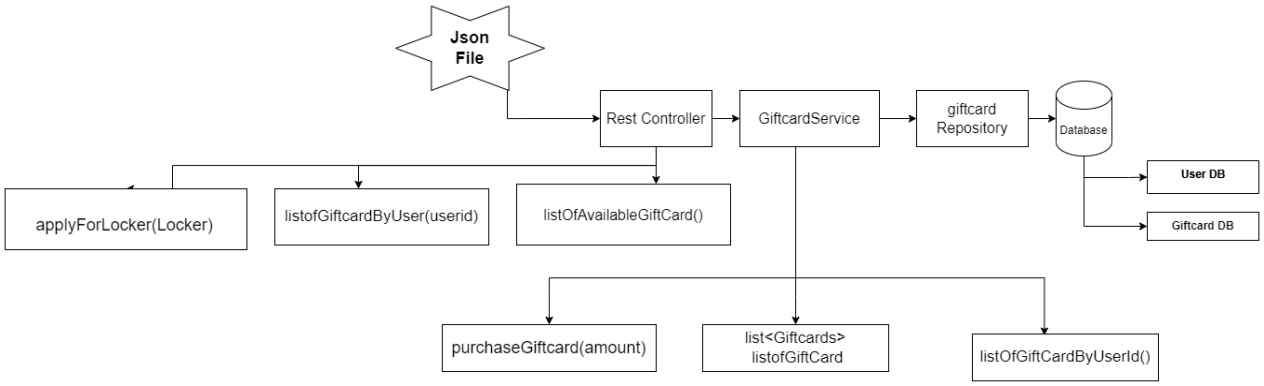
6.4 Gift Card Repository: The database that stores gift card account information.

6.5 Gift Card Model: UserId(@mapping), GiftCardId, GiftcardNumber, Redeem Code, PurchaseDate, Gift Card Balance.

6.6 UserModel: UserId(@Mapping), UserFirstName, UserLastName, UserMobileNumber, UserEmailId, UserPin, UserPassword, CurrentBalance, AccountNumber.

GiftCard Flowchart




Json File
Rest Controller
GiftcardService
giftcard Repository
Database
User DB
Giftcard DB
applyForLocker(Locker)
listofGiftcardByUser(userid)
listOfAvailableGiftCard()
purchaseGiftcard(amount)
list<Giftcards> listofGiftCard
listOfGiftCardByUserId()

Use case 7: Eureka Services

Eureka Server is an application that holds the information about all client-service applications. Every Micro service will register into the Eureka server and Eureka server knows all the client applications running on each port and IP address. Eureka Server is also known as Discovery Server.

Building a Eureka Server

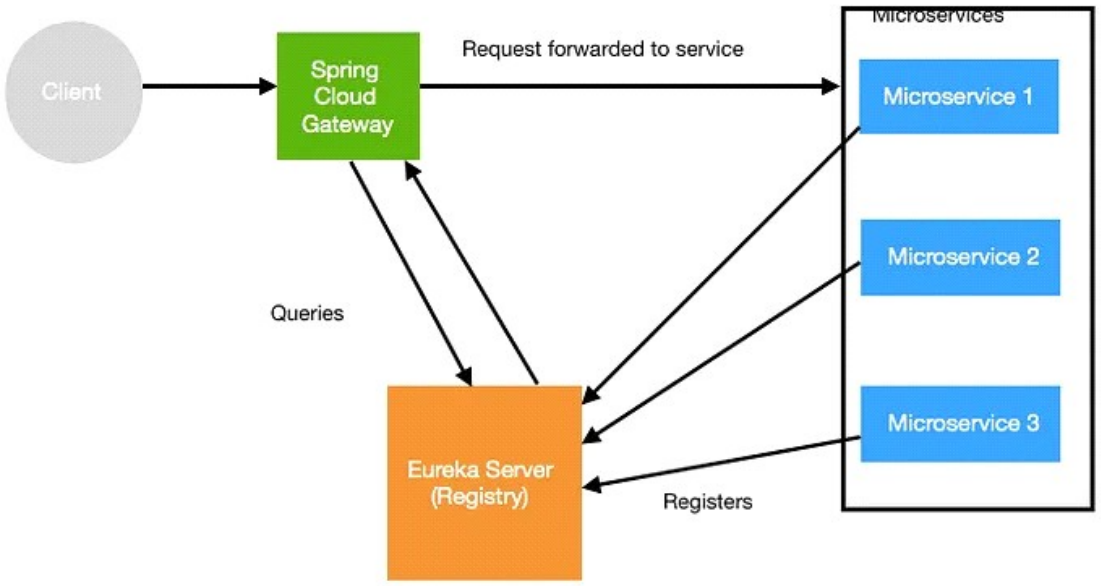
Eureka Server comes with the bundle of Spring Cloud. For this, we need to develop the Eureka server and run it on the default port 8761.

We will connect Eureka Server to all the MicorServices Like Login, Transaction Management, Credit Card, Loan, Locker, Gift Card.

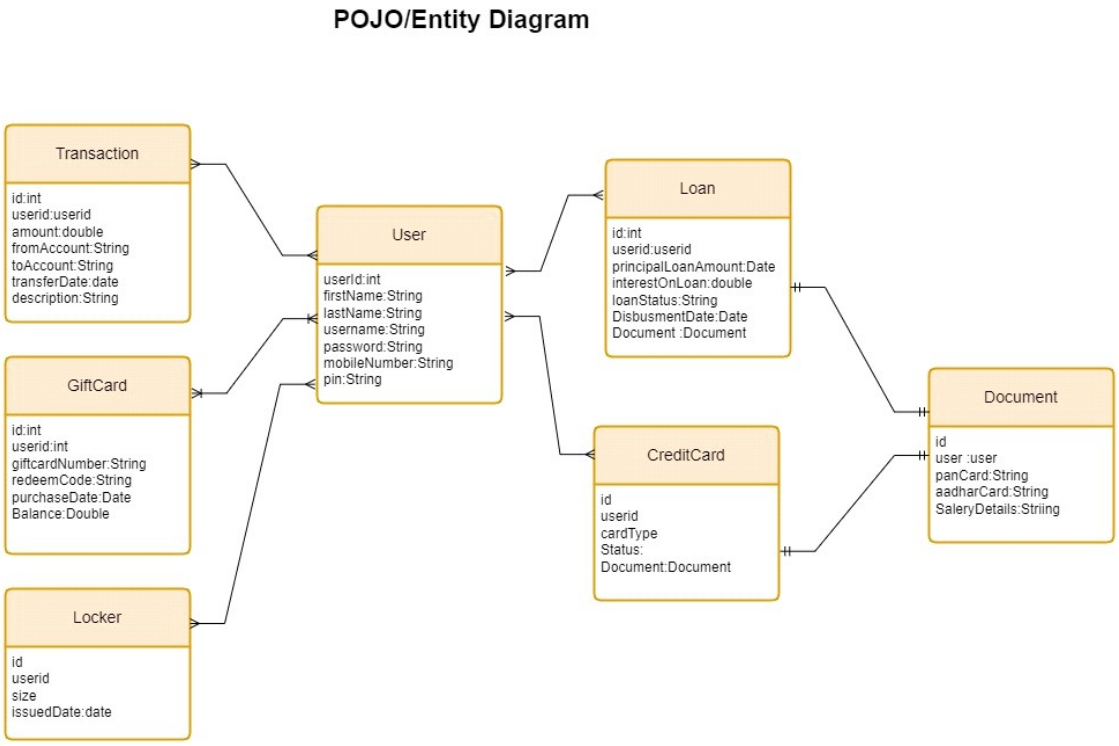
Use case 8: Cloud Services

Microservices is an architectural style that structures an application as a collection of small, loosely coupled, and independently deployable services. Each microservice focuses on performing a specific business functionality and communicates with other microservices through APIs. Combining Eureka and cloud-based microservices means utilizing Eureka as the service discovery mechanism within a cloud infrastructure, where microservices are deployed and communicate with Each Other.

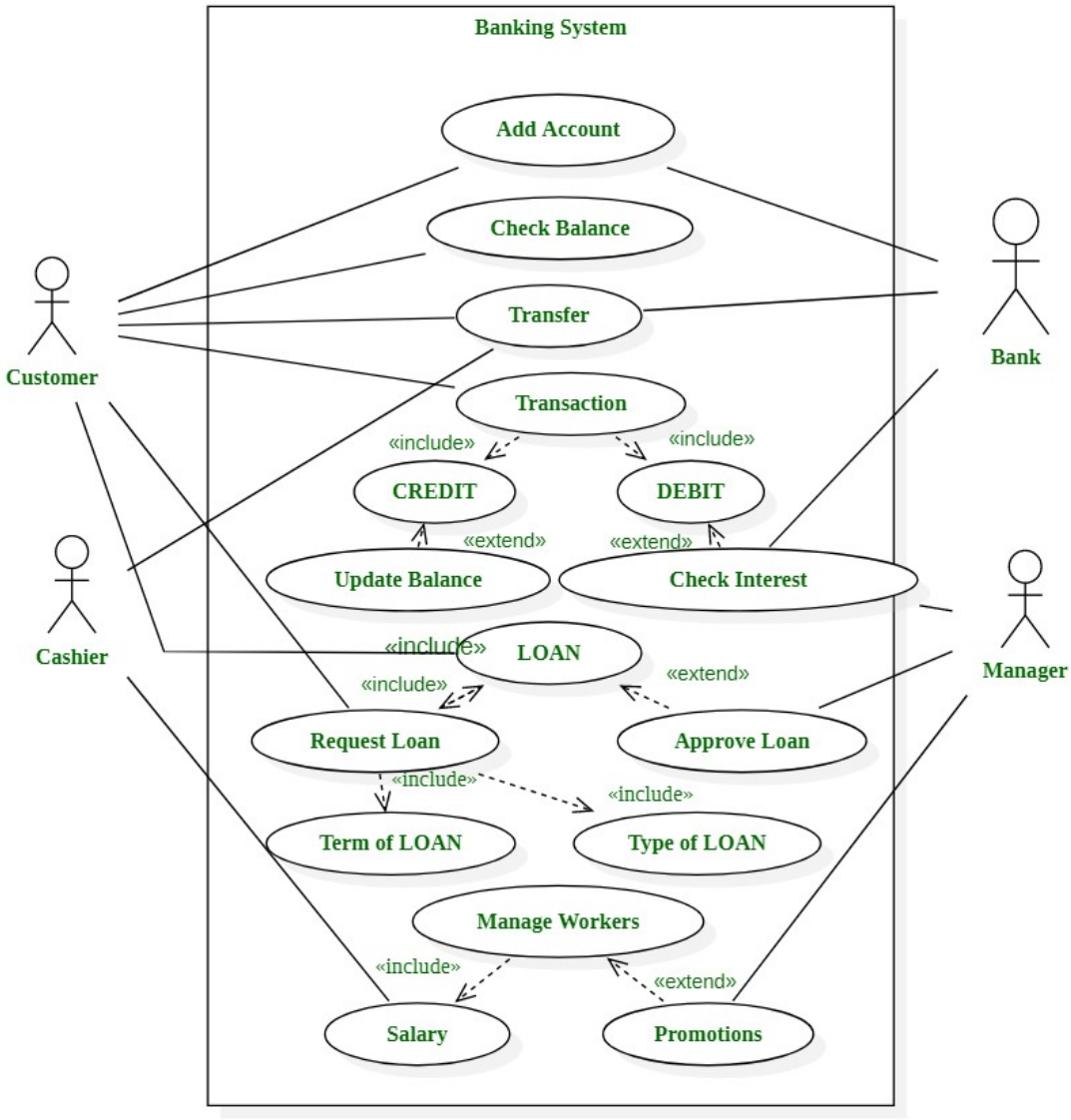
Here we will connect Spring Cloud Gateway to all the MicorServices (Login, Transaction Management, Credit Card, Loan, Locker, Gift Card) with the Eureka Server as Shown Below Diagram.



Entity Diagram:



USE CASE DIAGRAM:


Banking System
Add Account
Check Balance
Transfer
Transaction
«include»
«include»
CREDIT
DEBIT
«extend»
«extend»
Update Balance
Check Interest
«include»
LOAN
«include»
«extend»
Request Loan
Approve Loan
«include»
«include»
Term of LOAN
Type of LOAN
Manage Workers
«include»
«extend»
Salary
Promotions
Customer
Bank
Cashier
Manager