Rapid Banking Assistant Application

Instructor:	Emad Nasrallah
Class:	**MAD 3463**
Due Date:	Tuesday, Oct. 8
Percentage of Final Grade:	35%
Version	1.0

Project Members

Student Name	**Student Id**
Vibin Erapakkal joby	C0765779
Lijosh Abraham	C0764921

CONTENTS

INTRODUCTION

A bank is a financial institution which deals with money and its substitutes, it also provides other financial services. This project is basically providing a full service user friendly interface to the bank employees to assist their clients who approach them in terms of creating a new account or having trouble with an existing account, primarily specializing in primitive bank system.

Our **Rapid Banking Assistant** is an application for maintaining customer's accounts in a bank. The administrator, in terms of bank, the employee controls the whole system. This system provides access to the banking employees to create an account for their clients. The application can be used for normal banking purposes and is capable of performing the following functions

1. Create an account for the clients
2. Deposit money
3. Withdraw money
4. Transfer money between accounts
5. Interac money to others account
6. Pay utility bills
7. Activating credit card

PROJECT DESCRIPTION

This Project is aimed primarily at the bank employees and henceforth the client’s information is safeguarded with utmost security and even the bank employees would need the client’s assistant in terms of card number or their mother’s maiden name to take a look at their clients personal or banking information. Keeping this in mind we have designed the application in a systematic manner where people who have limited access to the internet or people who are very old, who would need assistance in terms of banking transactions are guided by the banking staff using our application.

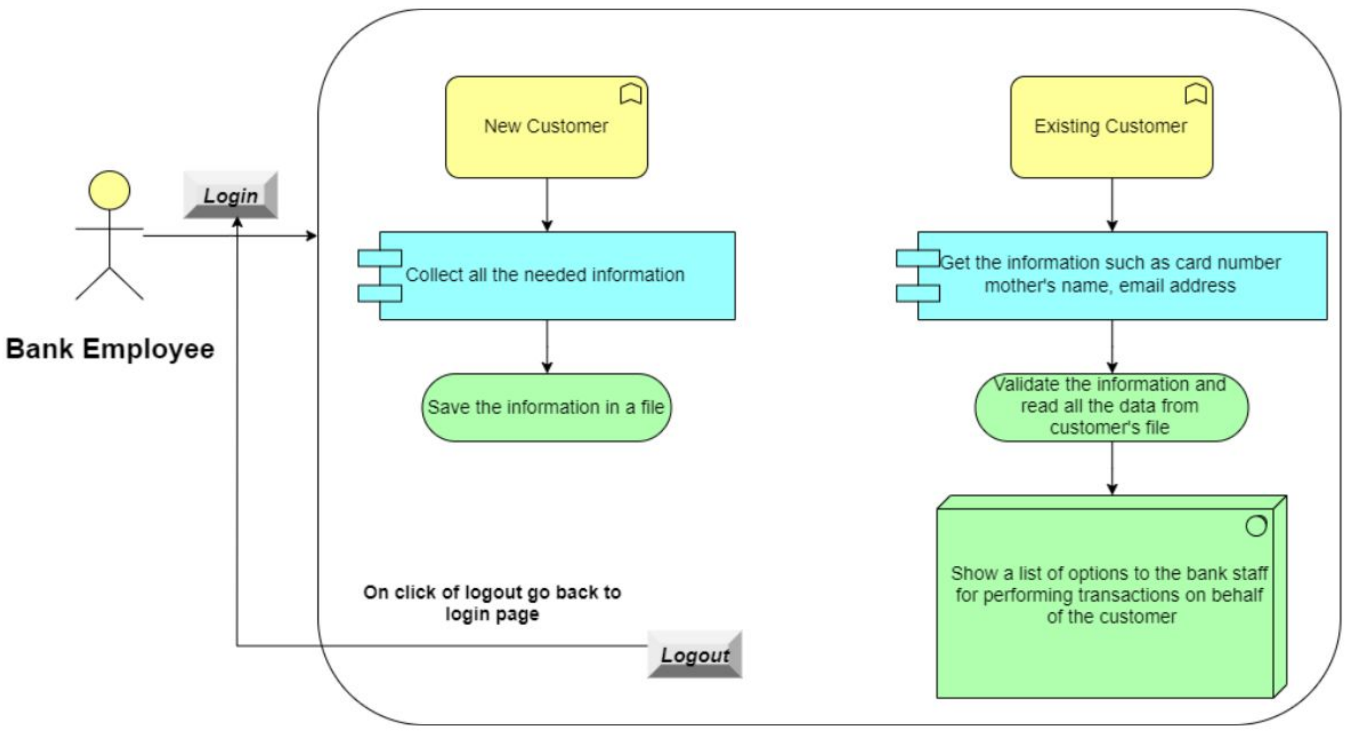
Our application has two primary features to assist people in need and we have categorized based on the most commonly used features

1. **New Customers** → which helps in creating an account with the bank with an exciting offer of 100$ cash in their account specially aimed at the new customers
2. **Existing Customers** → which helps in doing transaction between accounts or to a new account or in seeing their mini statement or in paying their utility bills

There are multiple sub options under existing customers page

- Depositing Money
- Transfer between two accounts
- Interac Money Transfer
- Withdrawing Money
- Paying Utility Bills like Hydro/Mobile bill
- Activating Credit card

HIGH LEVEL FUNCTIONAL DIAGRAM



Login
Bank Employee
New Customer
Collect all the needed information
Save the information in a file
Existing Customer
Get the information such as card number mother's name, email address
Validate the information and read all the data from customer's file
Show a list of options to the bank staff for performing transactions on behalf of the customer
On click of logout go back to login page
Logout

Number of pages(UI) created

1. Login page
2. Main menu
3. New customer registration
4. Existing customer sign in
5. Existing customer transaction page

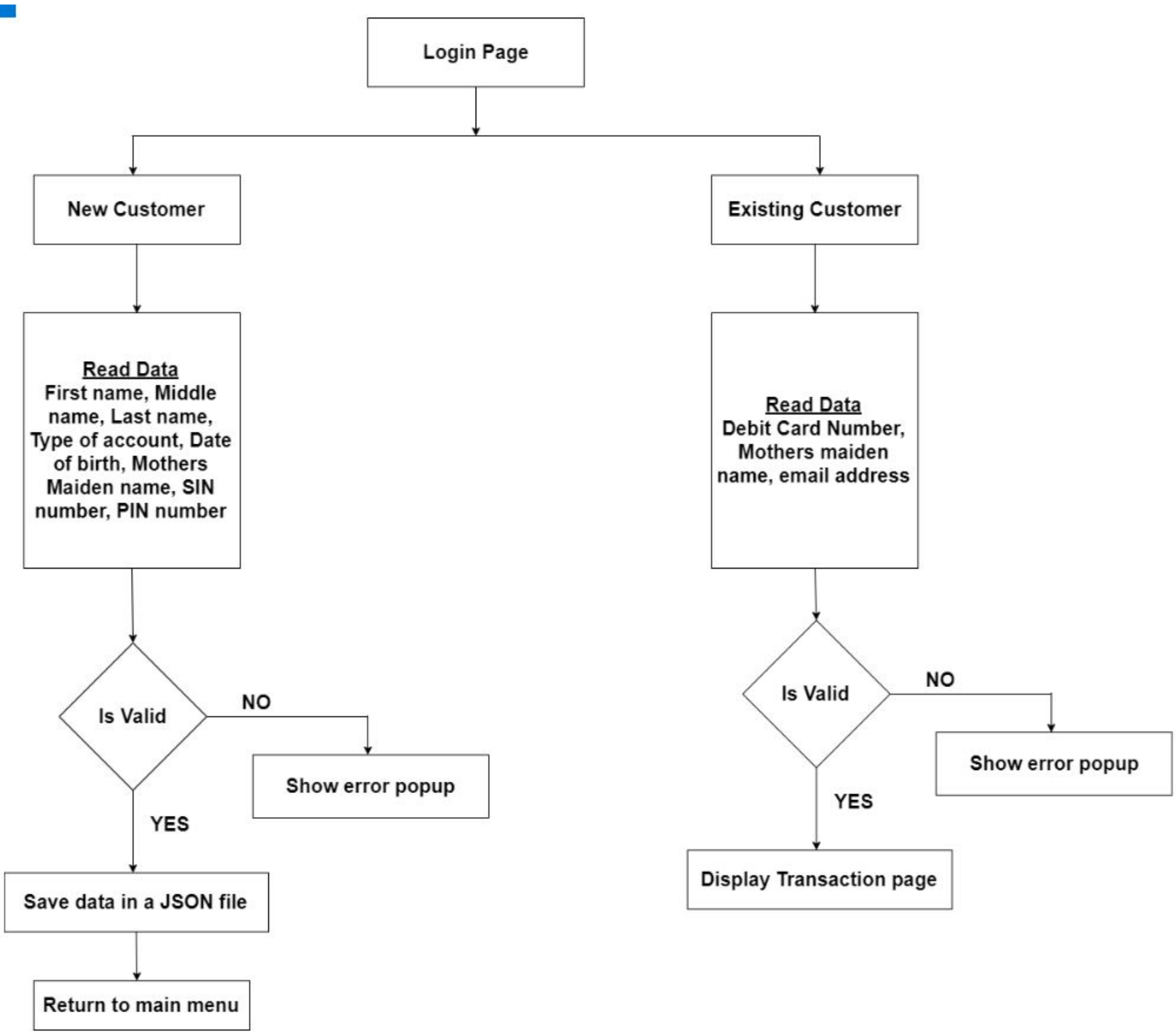
List of technologies/ Framework used technically

1. Java FX
2. Core Java
3. Files Framework (java.io package)
4. JSON / GSON

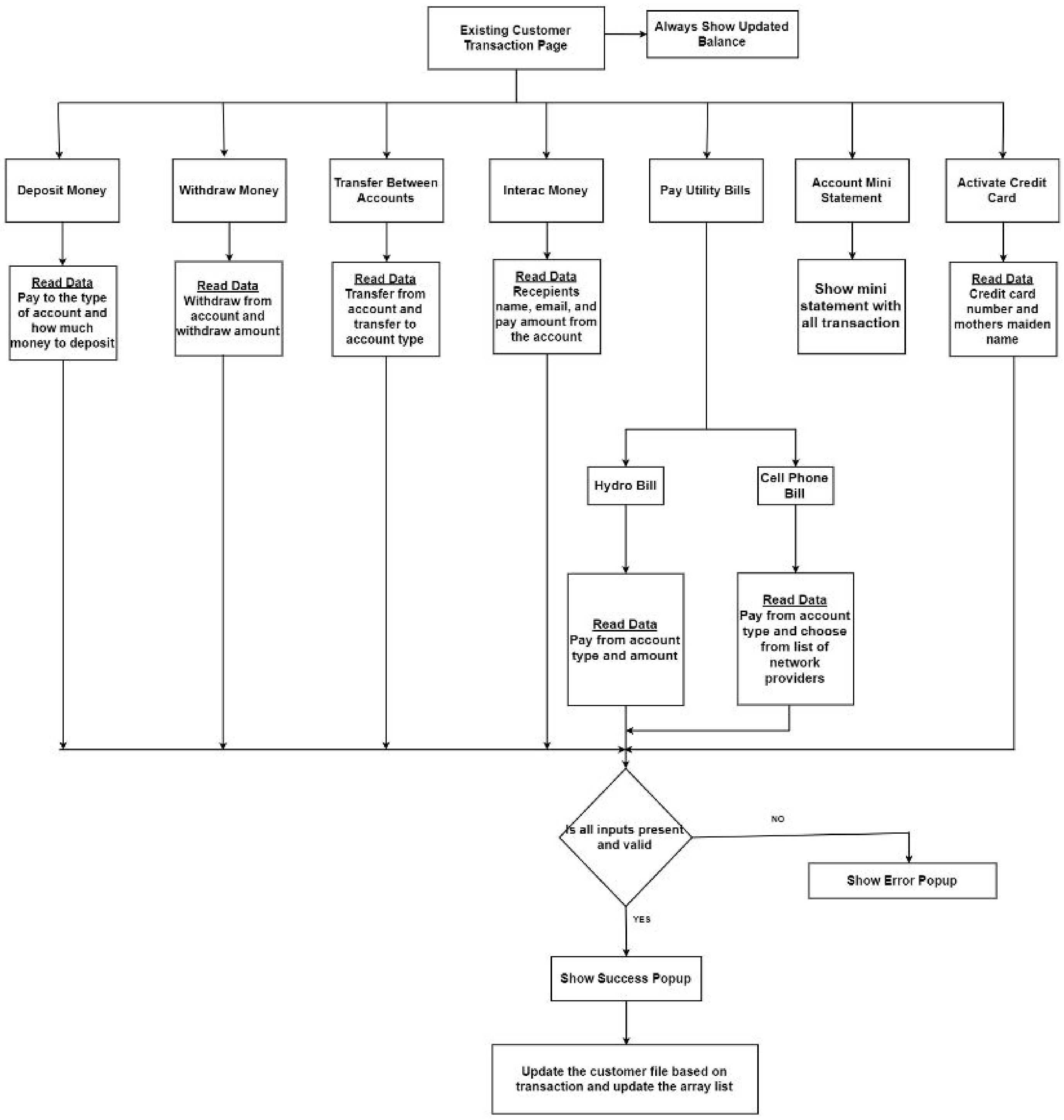
LOW LEVEL FUNCTIONAL DIAGRAM

Once the user logs in, application provides two options

- Create a new customer using the new Customer button
- Perform transactions using the Existing customer button



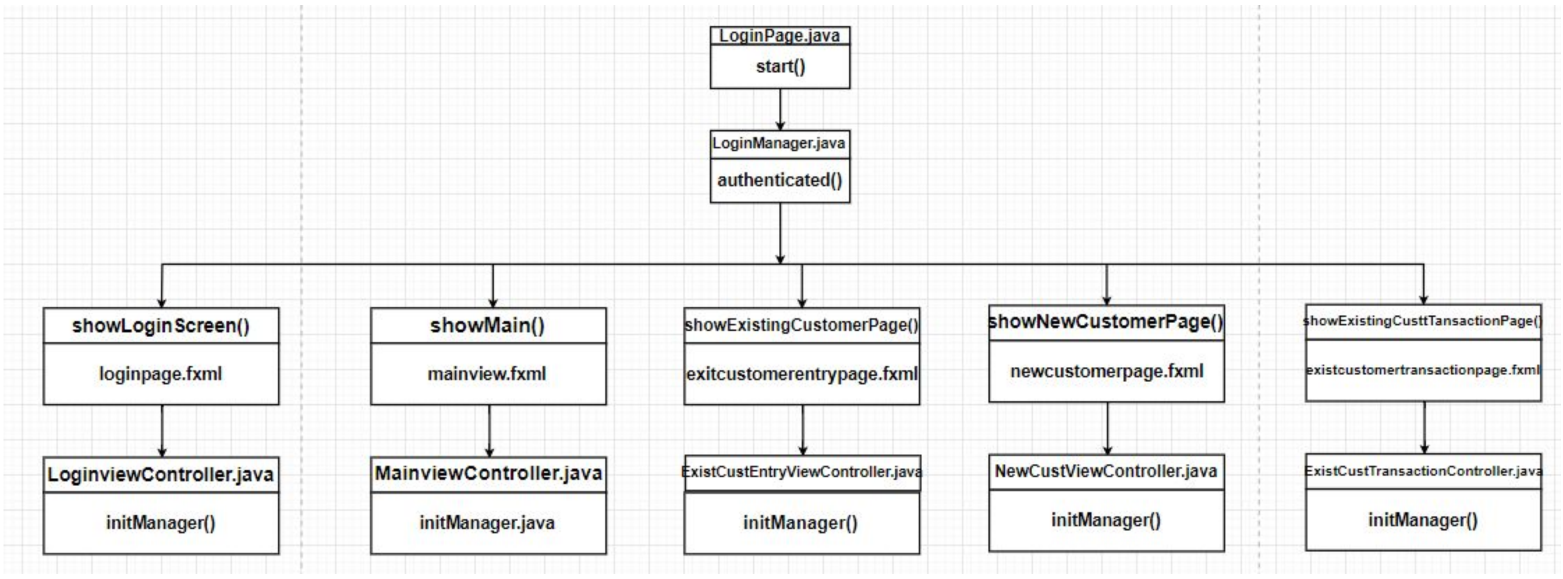
Below is a low level diagram of how the existing customer transaction page works and how the transactions are performed and after each transaction, the balance of the customer is updated synchronously onto the transaction page which the user can view and adequate validations are also performed to make sure there is no incorrect data entered into the field.



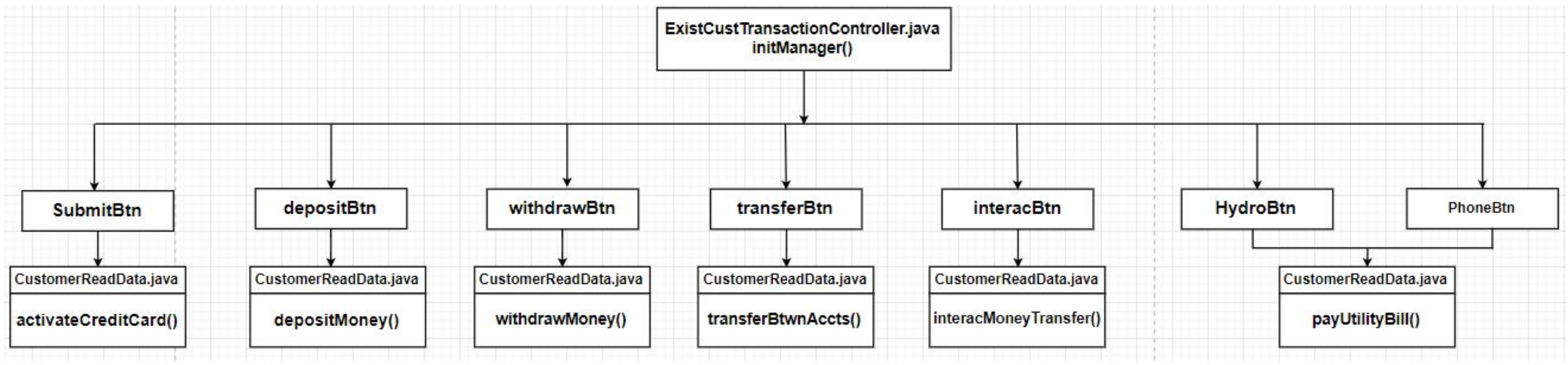
Existing Customer Transaction Page
Always Show Updated Balance
Deposit Money
Withdraw Money
Transfer Between Accounts
Interac Money
Pay Utility Bills
Account Mini Statement
Activate Credit Card
Read Data Pay to the type of account and how much money to deposit
Read Data Withdraw from account and withdraw amount
Read Data Transfer from account and transfer to account type
Read Data Recepients name, email, and pay amount from the account
Show mini statement with all transaction
Read Data Credit card number and mothers maiden name
Hydro Bill
Cell Phone Bill
Read Data Pay from account type and amount
Read Data Pay from account type and choose from list of network providers
Is all inputs present and valid
NO
Show Error Popup
YES
Show Success Popup
Update the customer file based on transaction and update the array list

HIGH LEVEL TECHNICAL DIAGRAM

Basic UI technical flow chart is detailed below

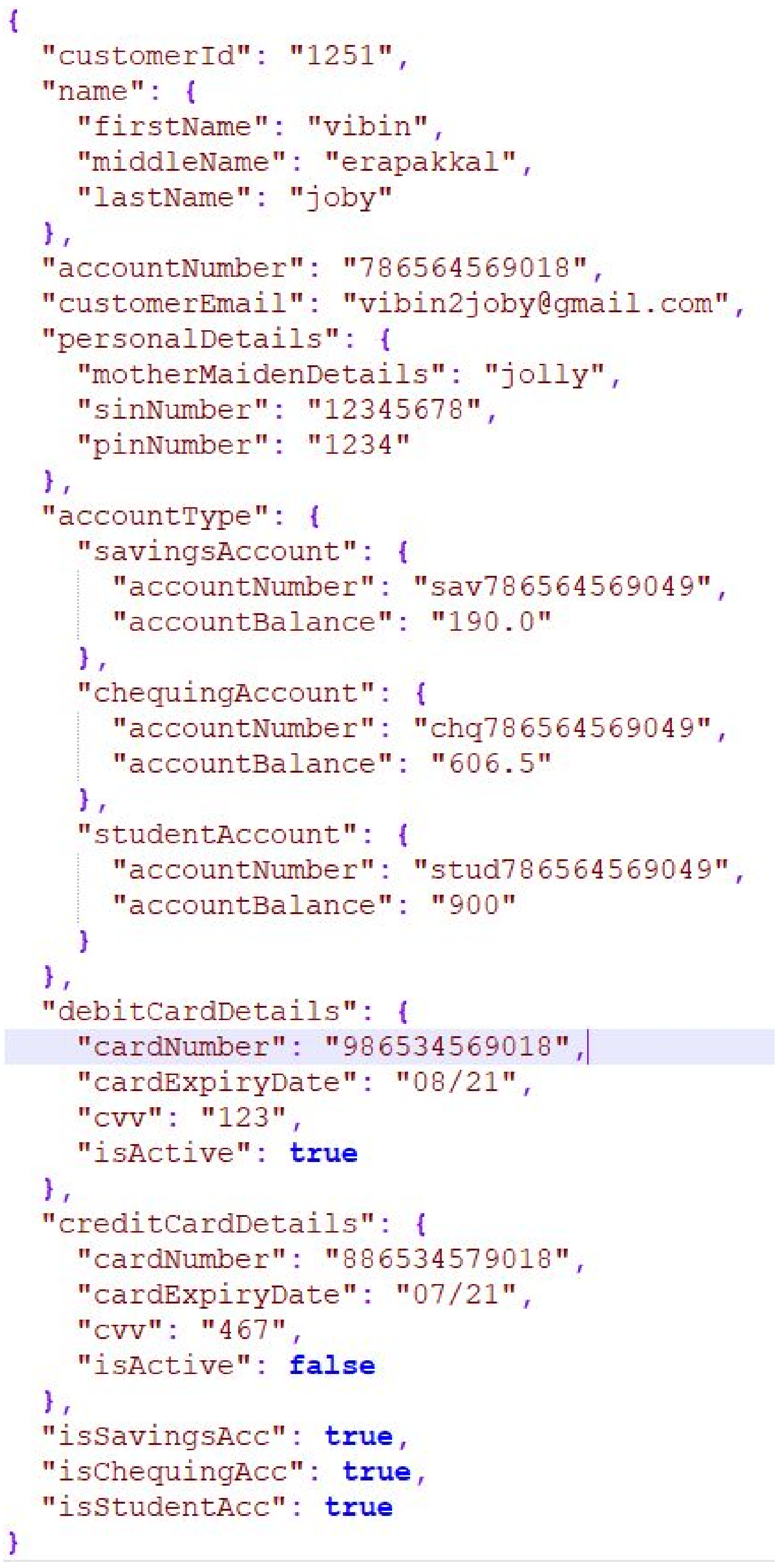


Below is the technical flow chart of the Existing customer transaction page which shows the method which gets invoked on click of each button in the corresponding page



FILES FRAMEWORK

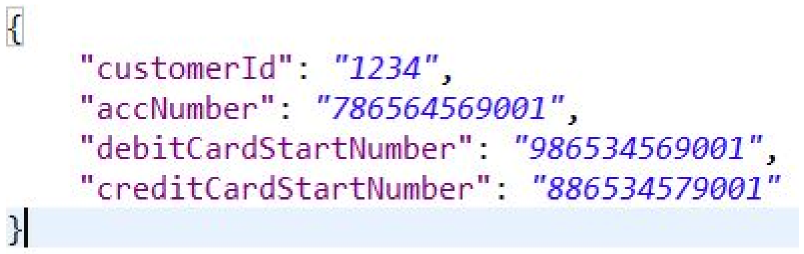
Below is a sample template of the JSON file created for a new customer



```
{
  "customerId": "1251",
  "name": {
    "firstName": "vibin",
    "middleName": "erapakkal",
    "lastName": "joby"
  },
  "accountNumber": "786564569018",
  "customerEmail": "vibin2joby@gmail.com",
  "personalDetails": {
    "motherMaidenDetails": "jolly",
    "sinNumber": "12345678",
    "pinNumber": "1234"
  },
  "accountType": {
    "savingsAccount": {
      "accountNumber": "sav786564569049",
      "accountBalance": "190.0"
    },
    "chequingAccount": {
      "accountNumber": "chq786564569049",
      "accountBalance": "606.5"
    },
    "studentAccount": {
      "accountNumber": "stud786564569049",
      "accountBalance": "900"
    }
  },
  "debitCardDetails": {
    "cardNumber": "986534569018",
    "cardExpiryDate": "08/21",
    "cvv": "123",
    "isActive": true
  },
  "creditCardDetails": {
    "cardNumber": "886534579018",
    "cardExpiryDate": "07/21",
    "cvv": "467",
    "isActive": false
  },
  "isSavingsAcc": true,
  "isChequingAcc": true,
  "isStudentAcc": true
}
```

The default path set in the code base is **C:\Users\vibin\Customer_Data\** and the file names are created using customer's first name appended with customer's id generated at the time of account creation in a text file format, for example **vibin1251.txt** and henceforth maintaining uniqueness in files generated.

We have created a JSON file named **application.json** in resources folder and below is the file created initially with default values



```
{
    "customerId": "1234",
    "accNumber": "786564569001",
    "debitCardStartNumber": "986534569001",
    "creditCardStartNumber": "886534579001"
}
```

Once the customer inputs data in the UI, we assign the customer id, account number and debit and credit card numbers from the above mentioned file to the customer's saving file and then we increment the numbers by 1 in the **application.json** file. In this way we maintain uniqueness in terms of customer id and account number for each customer and also they are sequential.

All the inputs received from the customer are stored in a **POJO**(Plain Old Java Object) file and at the time of saving the data to a file, this POJO is converted to a JSON object using GSON jar.

The system reads all the files from a specified path and its stored as a JSON object into a POJO file which is stored into an Array List at the start of the application when the CustomerReadWriteData.java file is initialized.

VALIDATIONS

Basically there are two types of validations in the system

- ❖ UI Validations
- ❖ Back end validations

UI Validations:

- ➢ ***Check for mandatory Fields***

  All the UI screens has mandatory field validations to reduce the burden of validating in the back end code so that if there is some input is missing we would identify it upfront and show an error popup asking the user to enter the required fields.
- ➢ ***Show error popup***

  When the backend method returns an ErrorDetails object file we will parse the information from the object and display the message and description of the error in a popup to the user

Back End Validations:

- ➢ ***Error code 001***

  This error code is for validating during the new account registration if an email is already associated with an existing bank account

- ➢ ***Error code 002***

  This error code is for storing all the common exceptions which would occur in the system and the same would be returned to the front end with the message and description

- ➢ ***Error code 003***

  This error code is for validating during the credit card activation to check if the customer has already activated the card

- ➢ ***Error code 004***

  This error code is for validating the balance a customer has to perform a transaction. Say suppose the customer has only 100 dollars in his account and he is trying to perform a transaction i.e (pay utility bills or transfer money) then we would be throwing a message stating **"Insufficient Balance"**

- ***Error code 005***

  This error code is for validating during the credit card activation to check if the customer has entered an incorrect credit card number for activation and the system would return an error stating **"Invalid Card..Please check the card number"**

- ***Error code 006***

  This error code is for validating during Interac money transfer option where the user would enter an email to transfer and if the email is not present in the banking database, we would return an error stating **"Email does not exist in the Bank"**

- ***Error code 007***

  This error code is for validating during Transfer between accounts option where the user has a possibility of entering the pay from account type and pay to account type as same and that would result in an error message as **"The From and To account cannot be the same!!"**

- ***Error code 008***

  This error code is for validating during Interac money transfer option where the user has to enter the recipient's email address for the money transfer and if the customer enters his email address itself instead of the recipient's email address then the system identifies it and throws a message stating **"From and To Email cannot be the same!!"**

- ***Error code 009***

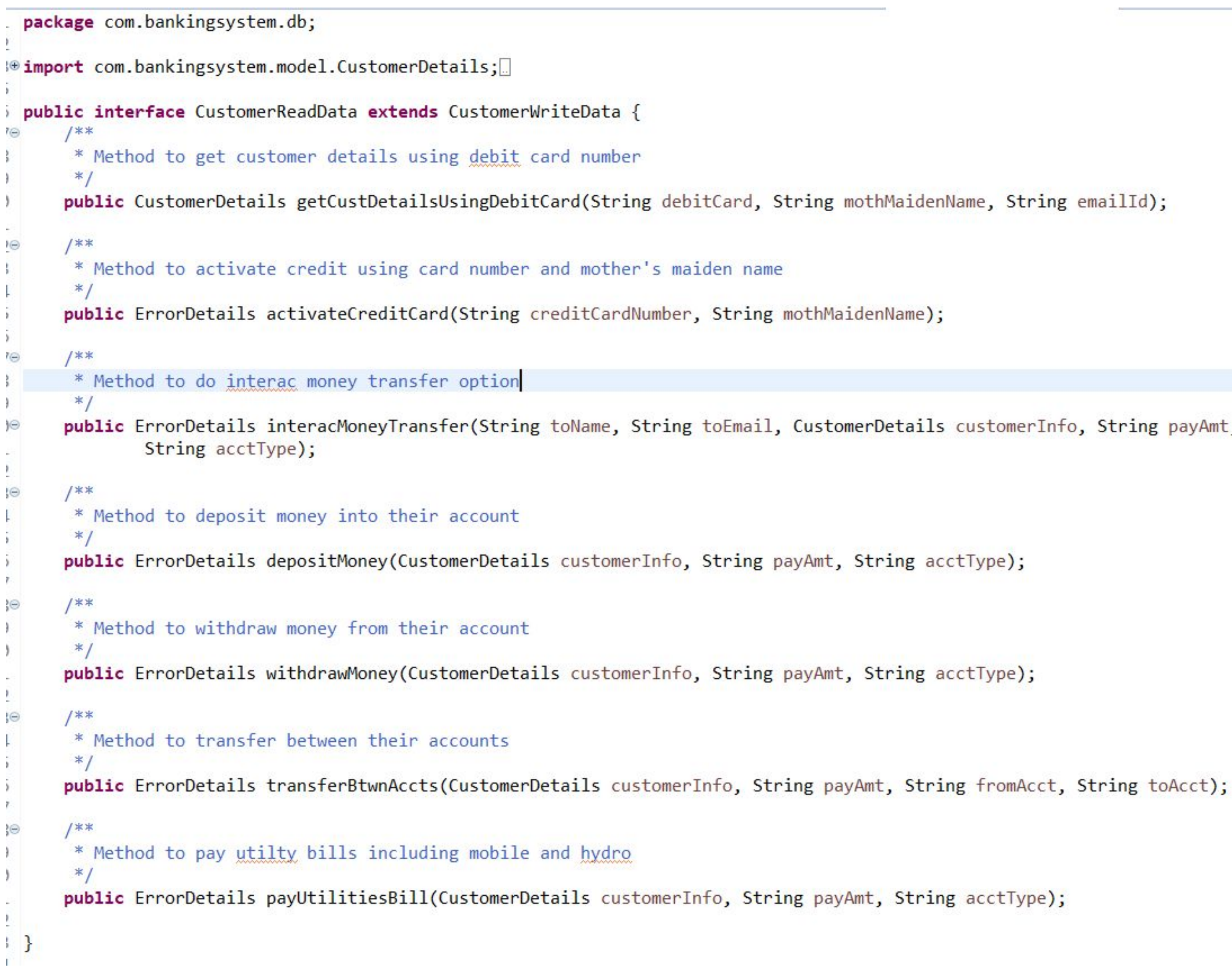
  This error code is for validating all the payment related transaction where in which the customer enters an incorrect number in the input and the system processes the data and finds the data is not a number it would throw a message stating **"Please enter a valid number for payment!!"**

  Other basic constraints kept in code base include but not limited to are "not null check" to prevent Null pointer exceptions.

INHERITANCE HIERARCHY

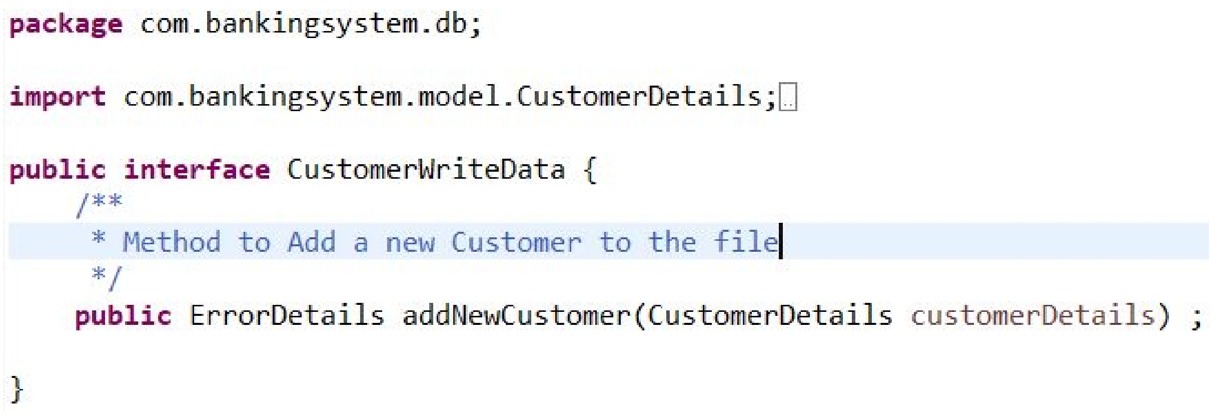
Two Interfaces are primarily created

- CustomerReadData.java
  This interface has all the common methods used to read data from the customer's file



```java
package com.bankingsystem.db;

import com.bankingsystem.model.CustomerDetails;

public interface CustomerReadData extends CustomerWriteData {
    /**
     * Method to get customer details using debit card number
     */
    public CustomerDetails getCustDetailsUsingDebitCard(String debitCard, String mothMaidenName, String emailId);

    /**
     * Method to activate credit using card number and mother's maiden name
     */
    public ErrorDetails activateCreditCard(String creditCardNumber, String mothMaidenName);

    /**
     * Method to do interac money transfer option
     */
    public ErrorDetails interacMoneyTransfer(String toName, String toEmail, CustomerDetails customerInfo, String payAmt
            String acctType);

    /**
     * Method to deposit money into their account
     */
    public ErrorDetails depositMoney(CustomerDetails customerInfo, String payAmt, String acctType);

    /**
     * Method to withdraw money from their account
     */
    public ErrorDetails withdrawMoney(CustomerDetails customerInfo, String payAmt, String acctType);

    /**
     * Method to transfer between their accounts
     */
    public ErrorDetails transferBtwnAccts(CustomerDetails customerInfo, String payAmt, String fromAcct, String toAcct);

    /**
     * Method to pay utilty bills including mobile and hydro
     */
    public ErrorDetails payUtilitiesBill(CustomerDetails customerInfo, String payAmt, String acctType);

}
```

- CustomerWriteData.java
  This Interface holds all the method to perform a write operation on the customer’s file



```java
package com.bankingsystem.db;

import com.bankingsystem.model.CustomerDetails;

public interface CustomerWriteData {
    /**
     * Method to Add a new Customer to the file
     */
    public ErrorDetails addNewCustomer(CustomerDetails customerDetails) ;

}
```

CustomerReadData.java inherits the properties of CustomerWriteData.java making the former as a parent interface and the later as a child interface.

Two Constants files created

- BankingConstants.java
  This has all the hard coded values in the code to a single file
- ErrorConstants.java
  This has all the hard coded values of the error and it’s message and description

BankingConstants.java inherits the properties of ErrorConstants.java making the former as a parent interface and the later as a child interface.

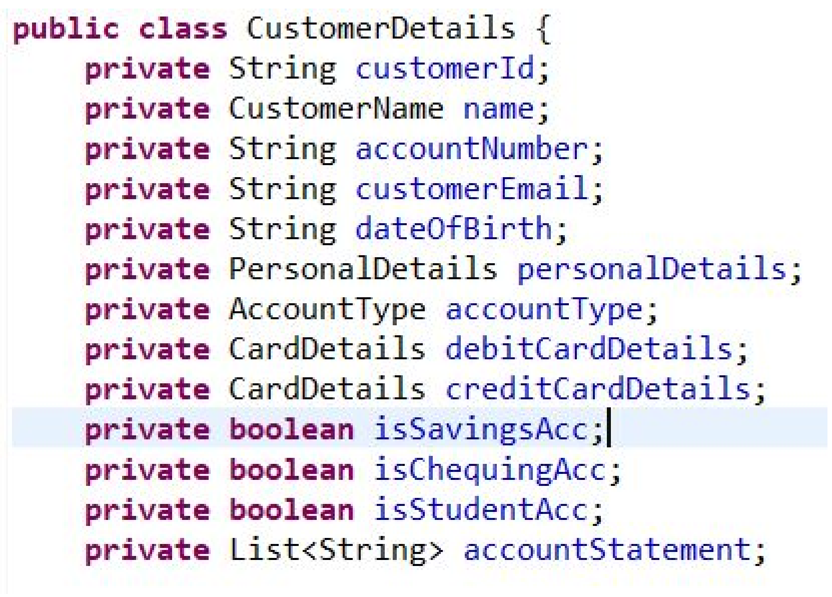
Class which is used for the interface implementation is

- CustomerReadWriteDataImpl.java

This class provides all the implementation for the methods in both the interfaces by implementing CustomerReadData.java
It also implements BankingConstants.java interface which is a constant file

CustomerDetails.java

This is the main POJO file with getters and setters used to save the inputs as a JSON in a text file when a new customer is created in the system and the same file is used to convert the file contents back to JSON object using GSON framework for reading the data of the clients.



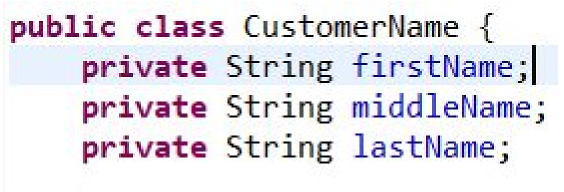
```
public class CustomerDetails {
    private String customerId;
    private CustomerName name;
    private String accountNumber;
    private String customerEmail;
    private String dateOfBirth;
    private PersonalDetails personalDetails;
    private AccountType accountType;
    private CardDetails debitCardDetails;
    private CardDetails creditCardDetails;
    private boolean isSavingsAcc;
    private boolean isChequingAcc;
    private boolean isStudentAcc;
    private List<String> accountStatement;
```

Attributes and its reason for existence:

- ***customerId***

  This field is used as a unique indicator to name the file along with the first name while saving it.

- ***name***



```
public class CustomerName {
    private String firstName;
    private String middleName;
    private String lastName;
```

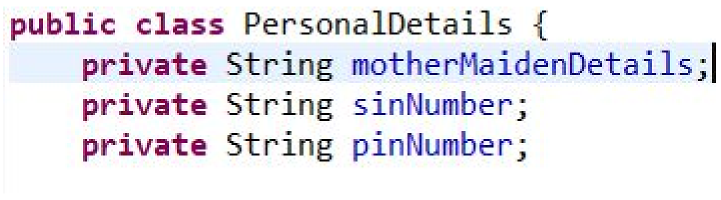
  Name is saved as first , middle and last name in the class named CustomerName.java

- ***accountNumber***

  This is sequentially generated from the front end making it unique and stored in the file

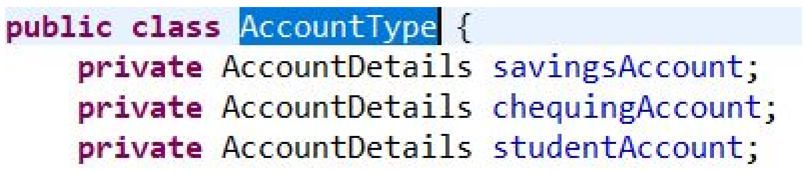
- ***customerEmail***
  This is obtained from the user’s inputs and is used for a variety of transactions most importantly during Interac

- ***dateOfBirth***
  Date of birth of a customer is stored from the user’s inputs
- ***personalDetails***

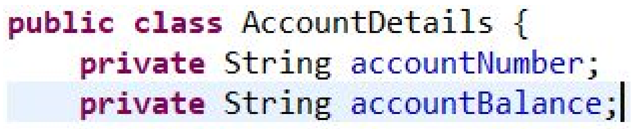


```java
public class PersonalDetails {
    private String motherMaidenDetails;
    private String sinNumber;
    private String pinNumber;
```

  We store the client’s mother’s maiden name, sin number of the client and the Pin number of the client where he resides in the class called PersonalDetails.java
- ***accountType***



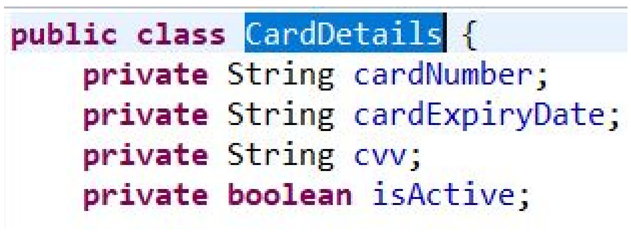
```java
public class AccountType {
    private AccountDetails savingsAccount;
    private AccountDetails chequingAccount;
    private AccountDetails studentAccount;
```



```java
public class AccountDetails {
    private String accountNumber;
    private String accountBalance;
```

  - ❖ The type of accountType variable is a class named AccountType.java to differentiate the account type as Savings/Chequing/Student account
  - ❖ The variables in AccountType are of type AccountDetails.java which has attributes named accountNumber and accountBalance where the account number for each account type is appended with “chq”/”sav”

- ***debitCardDetails / creditCardDetails***



```java
public class CardDetails {
    private String cardNumber;
    private String cardExpiryDate;
    private String cvv;
    private boolean isActive;
```

The type of debitCardDetails/ creditCardDetails is a class called CardDetails which stores the client’s card number,expiry date,cvv and if the card is active where the card number is sequentially generated in the front end making it unique

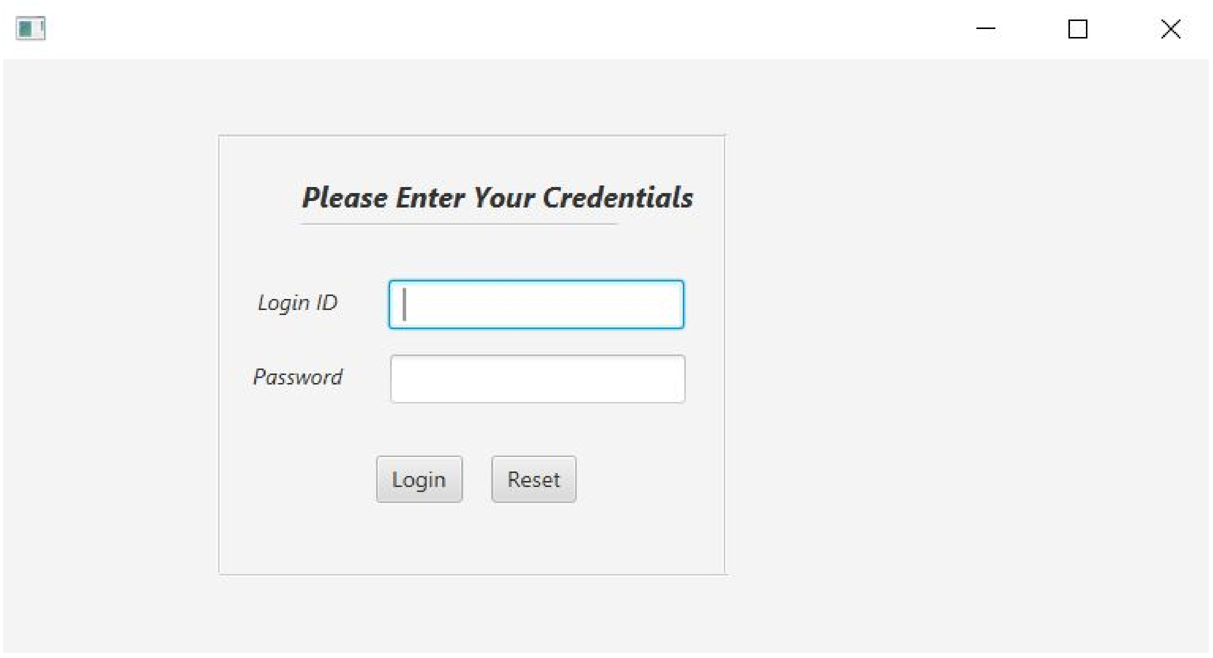
- ***isSavingsAcc/ isChequingAcc/ isStudentAcc***
  These boolean attributes are used to identify the type of account, the client has and based on that the system will dig in deep to identify the account number and the balance details
- ***accountStatement***
  This attribute is used to store all the transaction history in an Array list

Source Code GITHub Link:

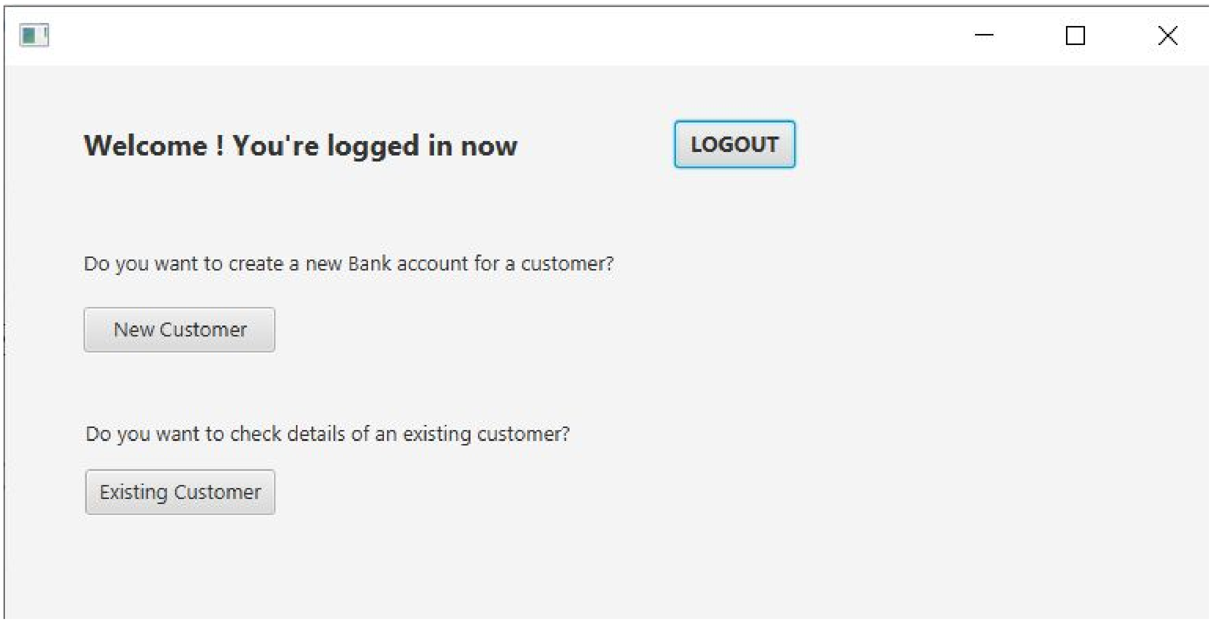
https://github.com/vibinjoby/BankingApplication

USER CATALOG

Login Page



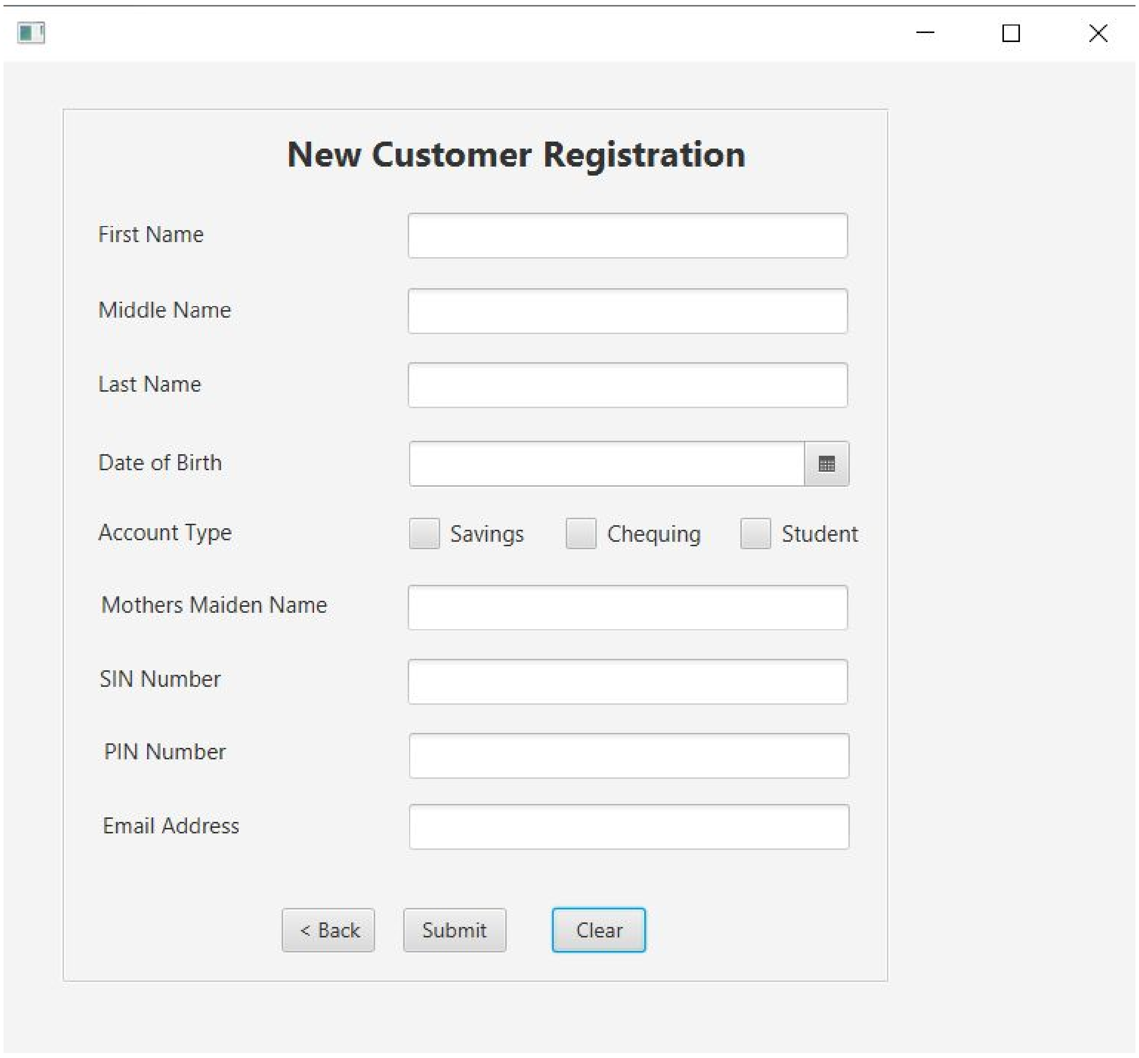
Main Menu


Welcome ! You're logged in now

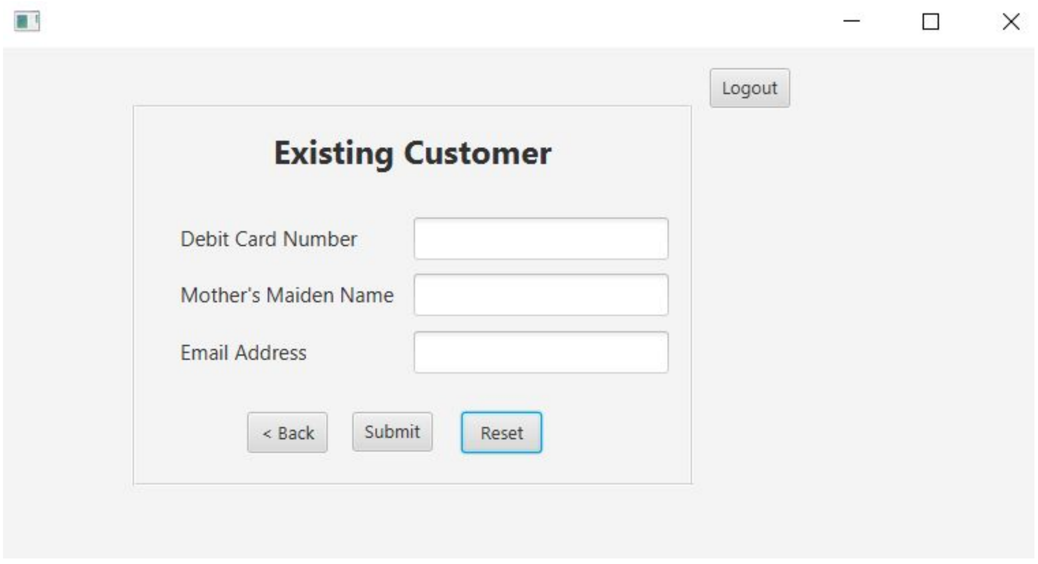
Do you want to create a new Bank account for a customer?

Do you want to check details of an existing customer?

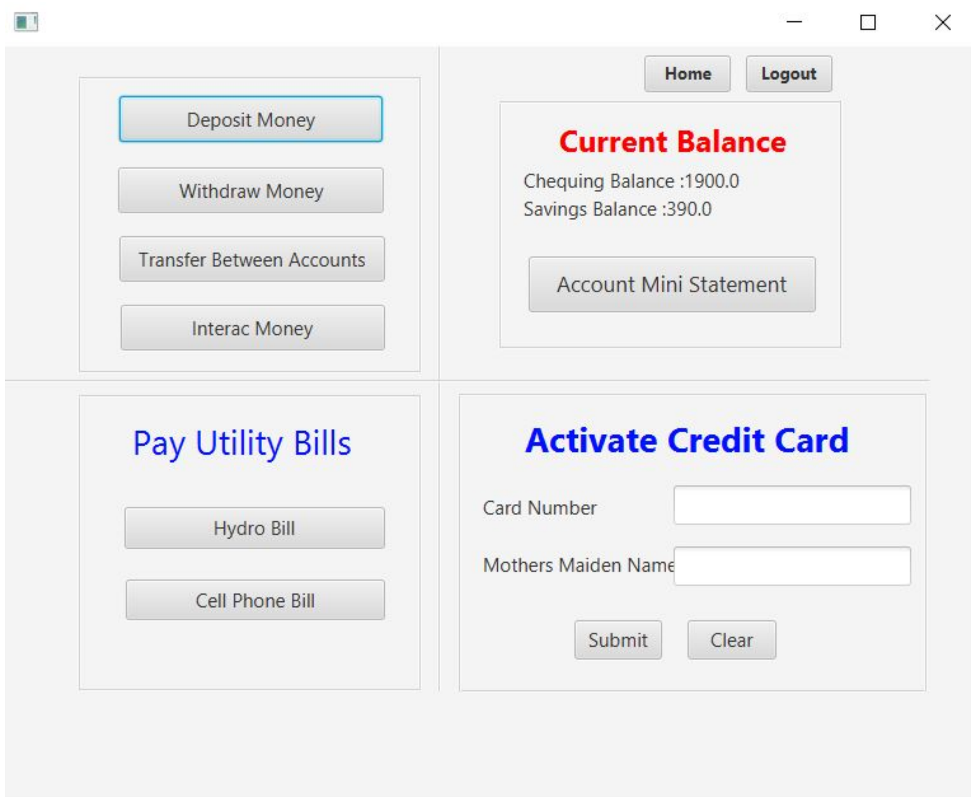
New Customer Registration Page



Existing Customer Page

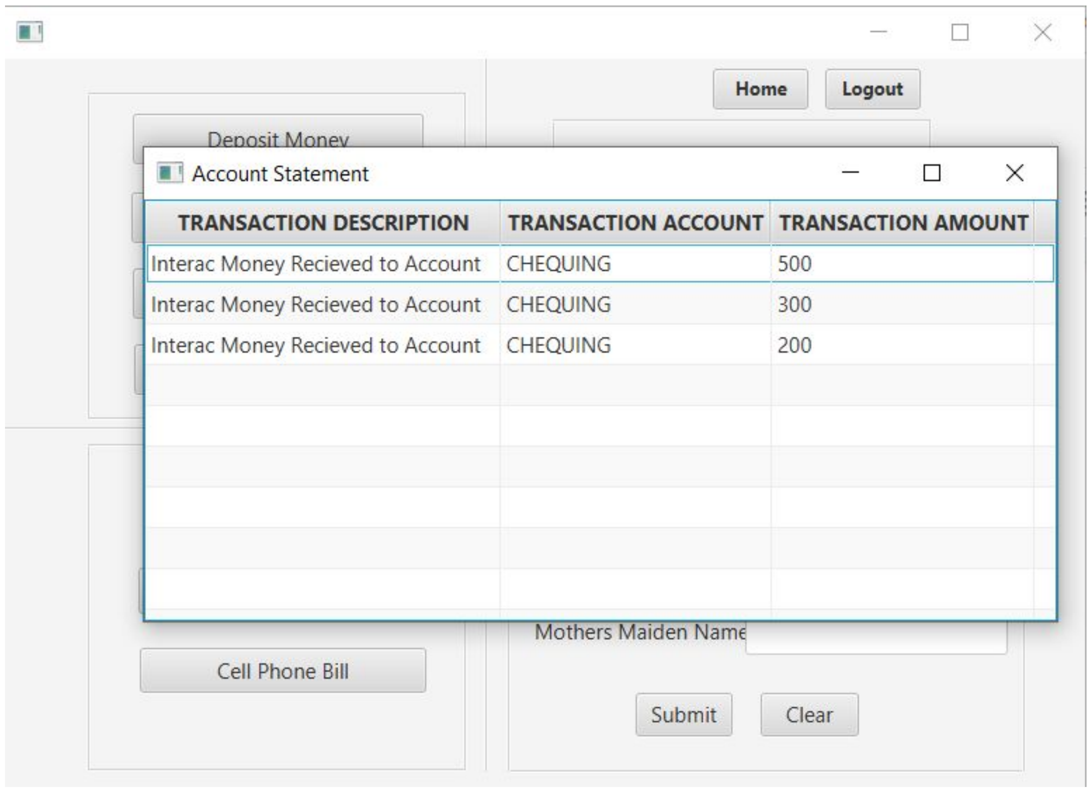


Customer Transaction Page



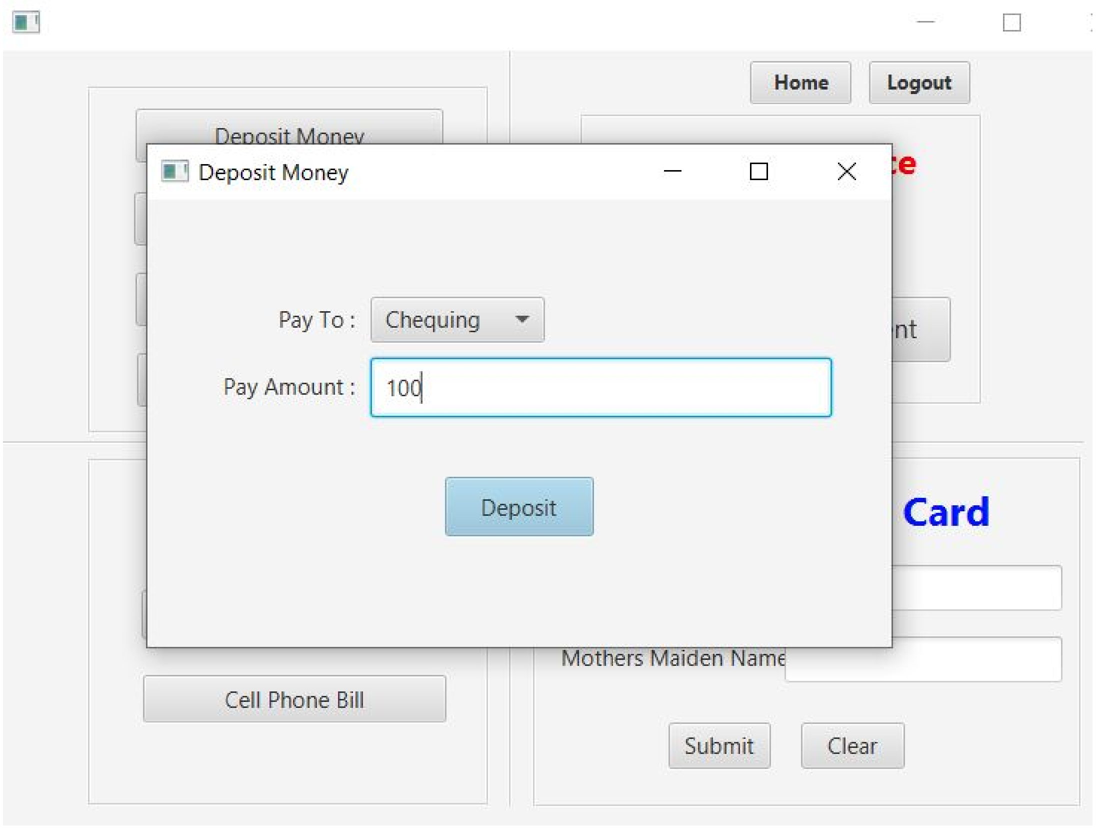
Chequing Balance :1900.0
Savings Balance :390.0

Account Statement



TRANSACTION DESCRIPTION	TRANSACTION ACCOUNT	TRANSACTION AMOUNT
Interac Money Recieved to Account	CHEQUING	500
Interac Money Recieved to Account	CHEQUING	300
Interac Money Recieved to Account	CHEQUING	200

Deposit Money



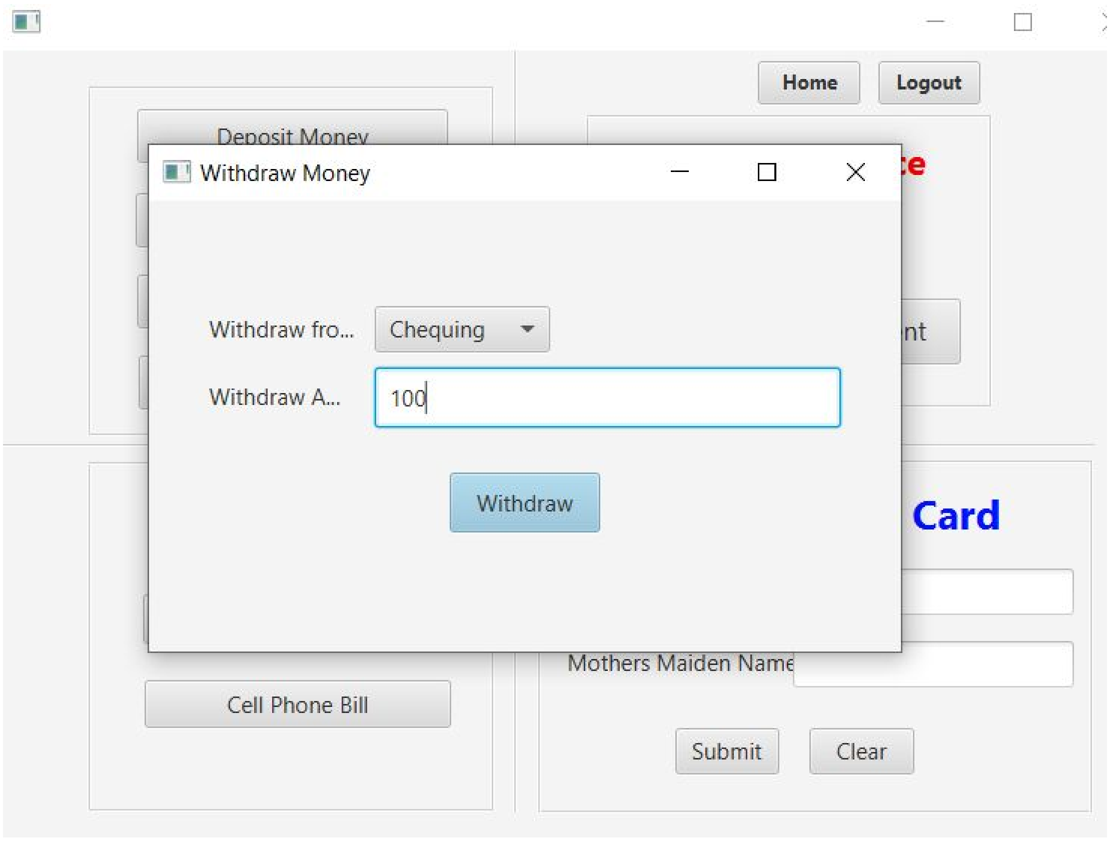
Deposit Money

Pay To : Chequing

Pay Amount : 100

Deposit

Withdraw Money



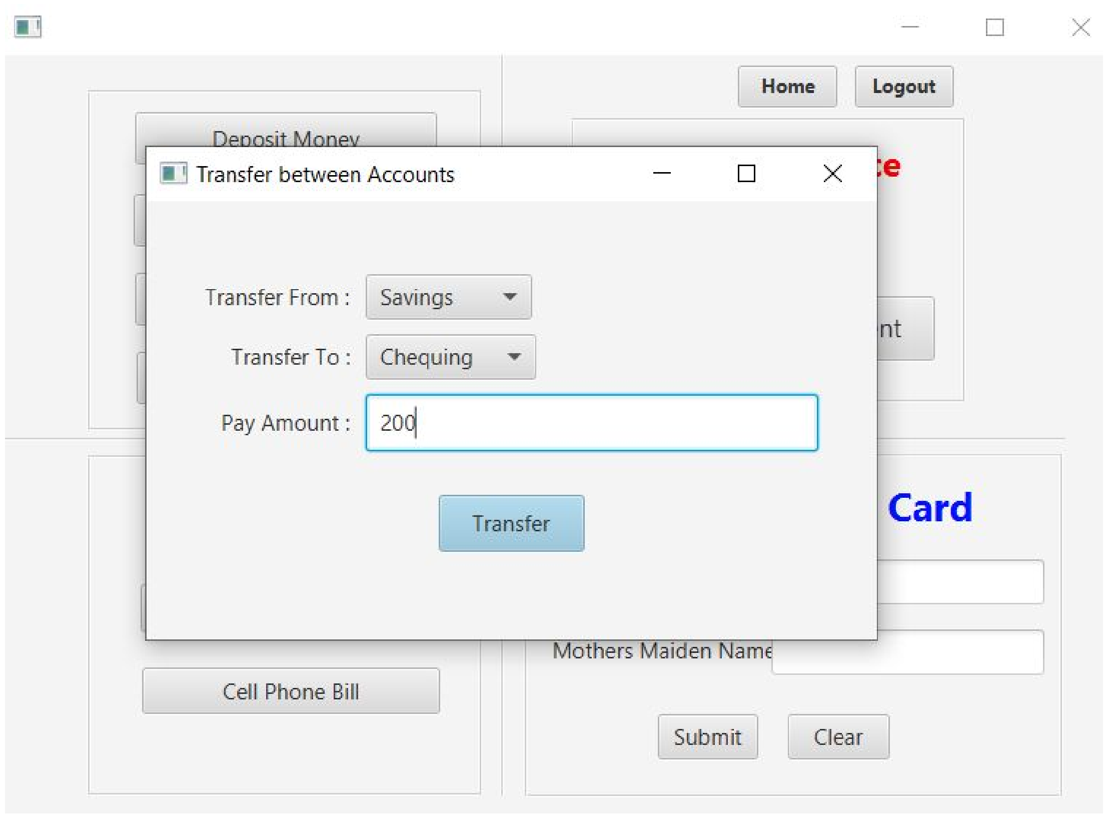
Withdraw Money

Withdraw fro... Chequing

Withdraw A... 100

Withdraw

Transfer Between Accounts

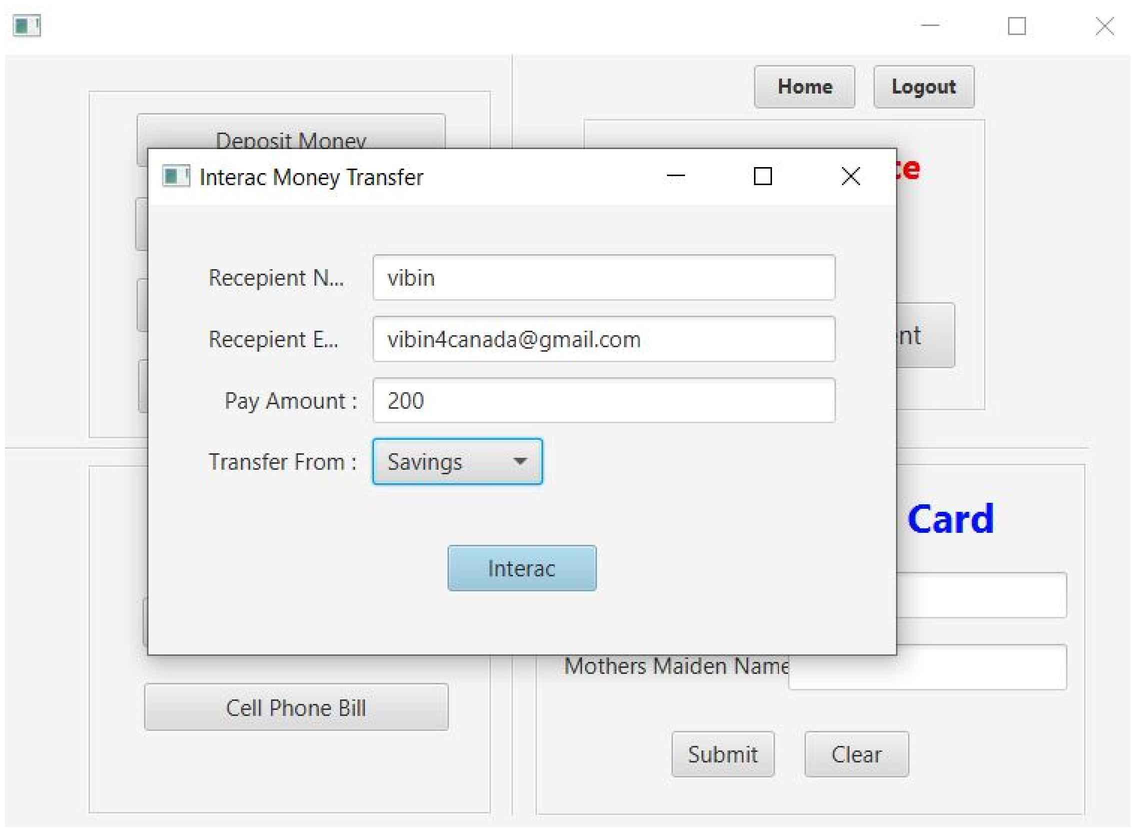

Transfer between Accounts

Transfer From : Savings

Transfer To : Chequing

Pay Amount : 200

Interac Money


Interac Money Transfer

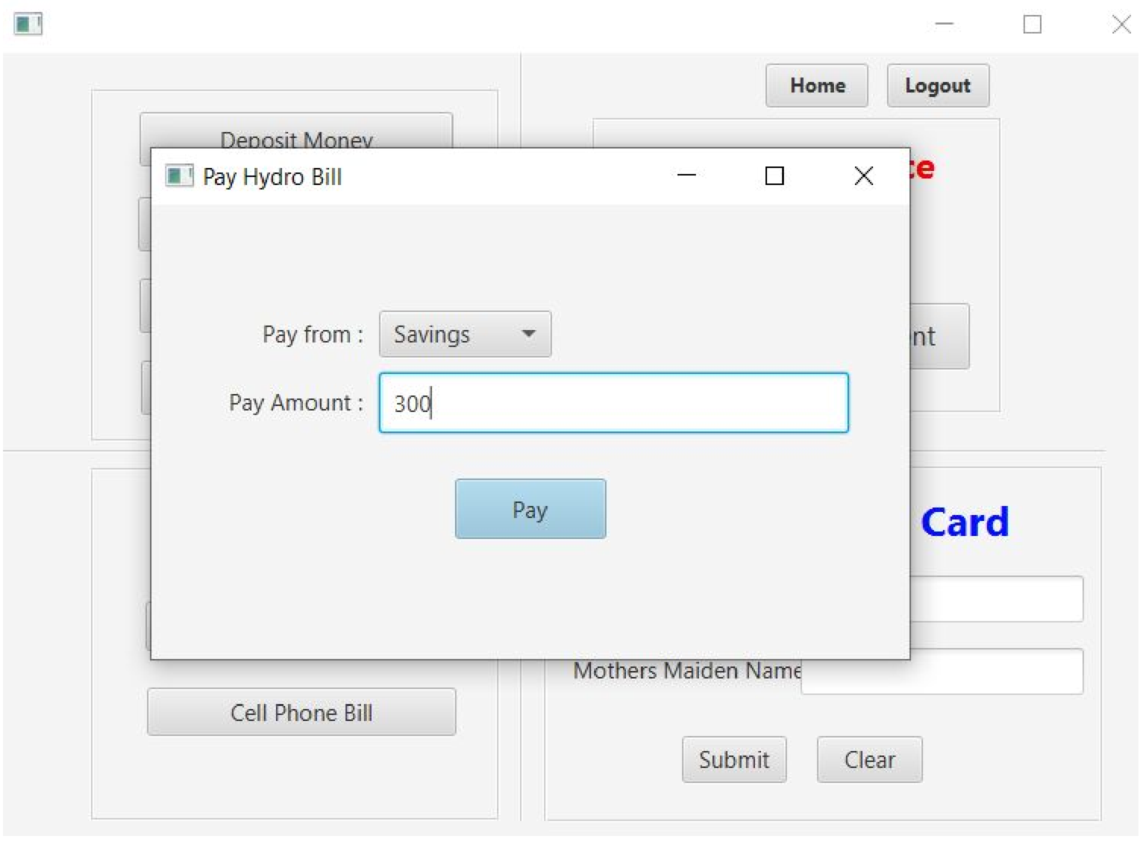
Recepient N... vibin

Recepient E... vibin4canada@gmail.com

Pay Amount : 200

Transfer From : Savings

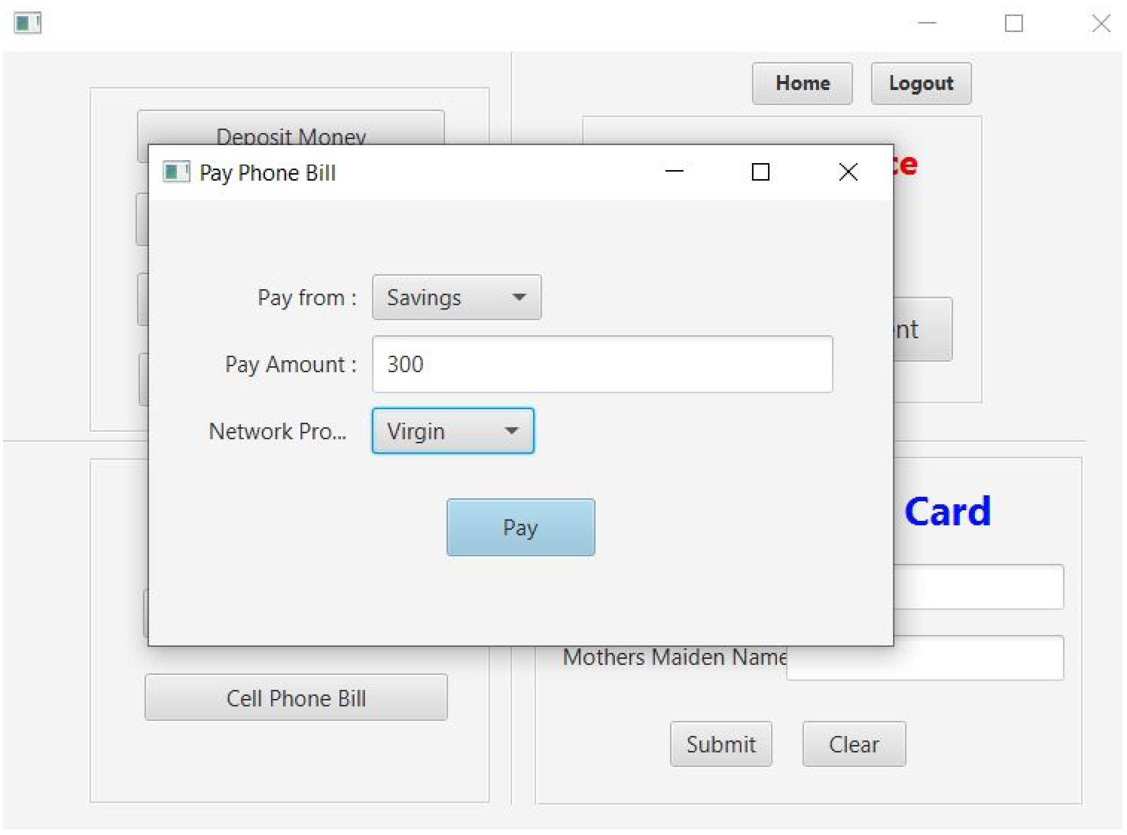
Pay Hydro Bill



Pay from : Savings

Pay Amount : 300

Pay Phone Bill



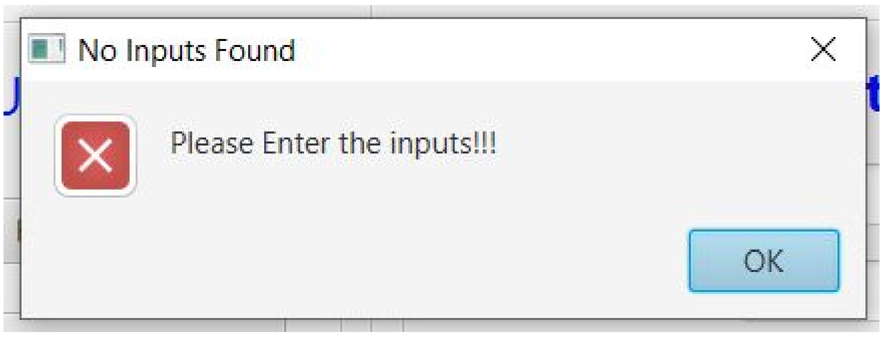
Pay from : Savings

Pay Amount : 300

Network Pro... Virgin

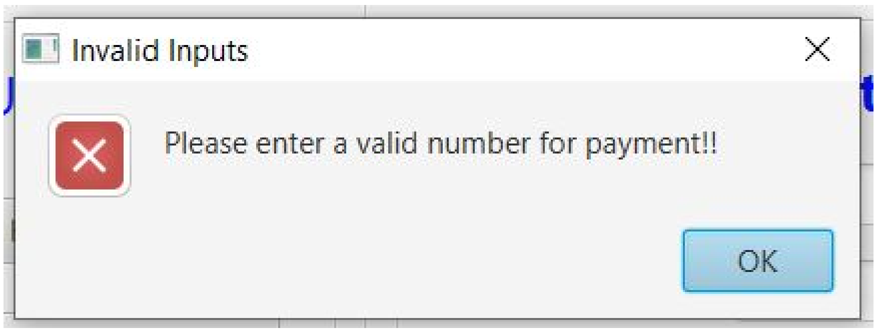
UI ERROR VALIDATIONS

Common Error popup which is used to denote that there is a mandatory value missing on click of submit


No Inputs Found

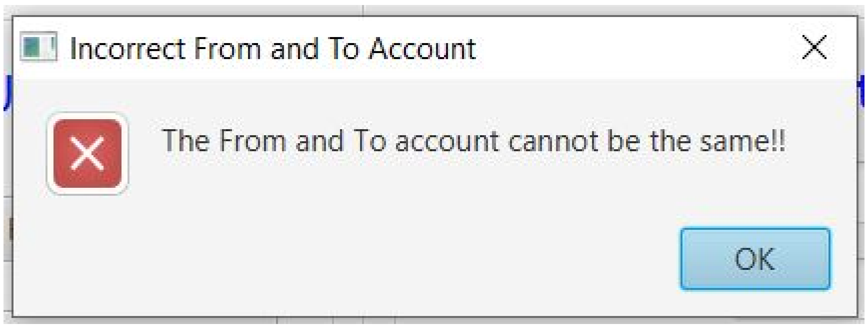
Please Enter the inputs!!!

Popup which appears when there is an invalid number entered in the field of amount for payment


Invalid Inputs

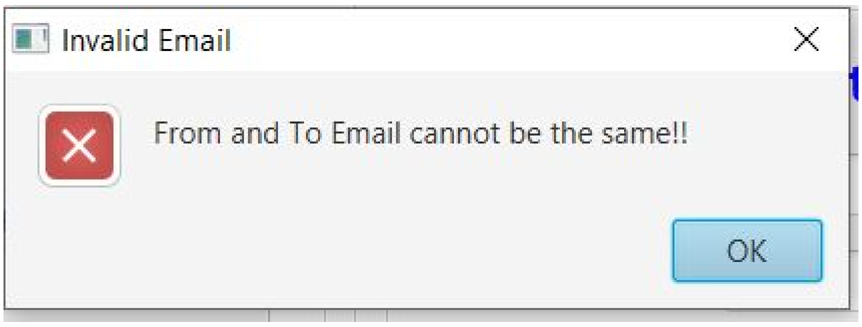
Please enter a valid number for payment!!

Popup which appears in the **Transfer between accounts** when the "from account type" option and "to account type" option has the same value


Incorrect From and To Account

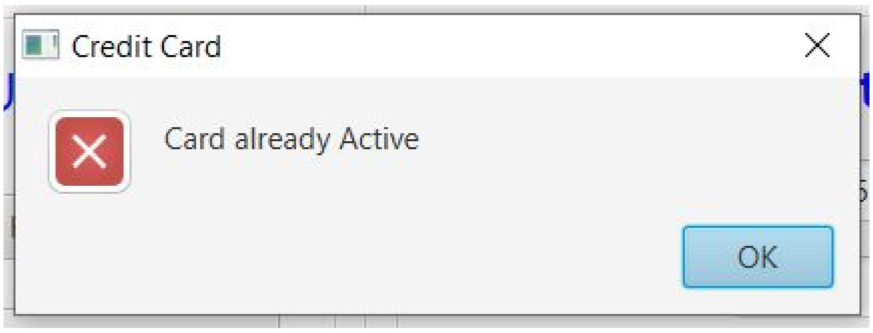
The From and To account cannot be the same!!

Popup which appears in the Interac money transfer where the customer enters his own email address in the recipient's email address textfield


Invalid Email

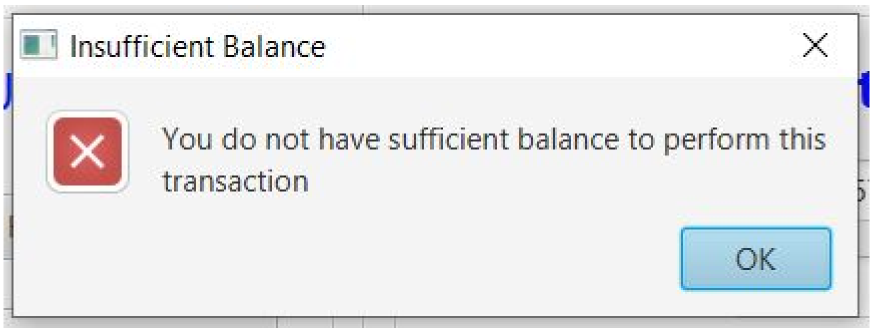
From and To Email cannot be the same!!

Popup which appears in the activate credit card option where the credit card is already active when trying to activate the credit card again


Credit Card

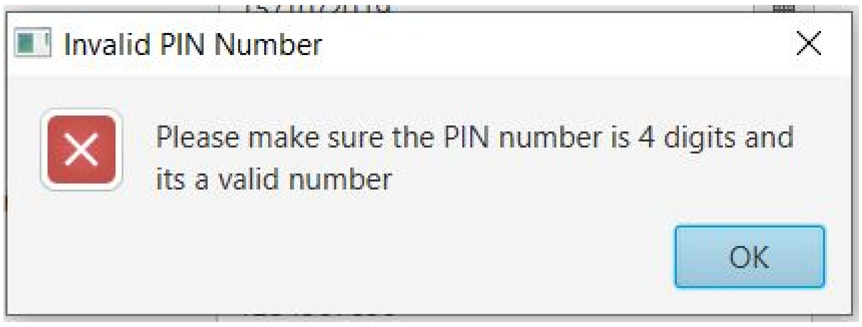
Card already Active

Popup which appears when there is insufficient balance to perform a banking transaction. This appears while doing an Interac transfer/ Transfer between accounts/ Paying utility bills, etc..


Insufficient Balance

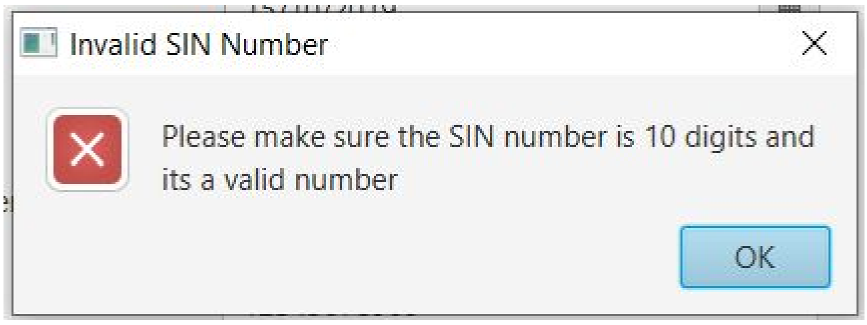
You do not have sufficient balance to perform this transaction

Popup which appears when the entered pin number is more than 4 digits or less than that during the new registration of the customer


Invalid PIN Number

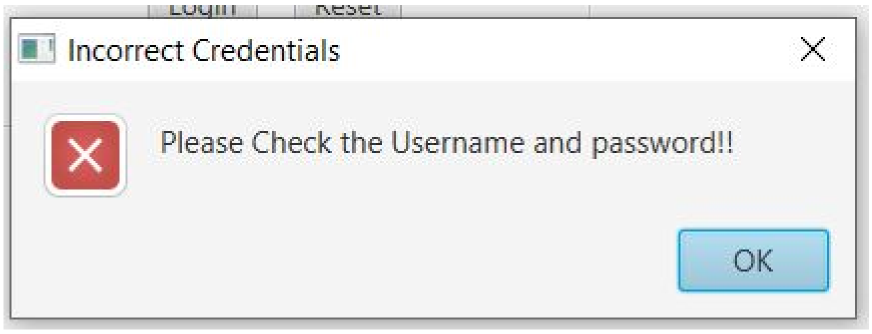
Please make sure the PIN number is 4 digits and its a valid number

Popup which appears when the entered SIN number is more than 10 digits or less than that during the new registration of the customer


Invalid SIN Number

Please make sure the SIN number is 10 digits and its a valid number

Popup which appears when the login credentials are invalid during the login page


Incorrect Credentials

Please Check the Username and password!!

CONCLUSION

This project is developed to nurture the needs of a user in banking sector by embedding most of the basic transactions taking place in a bank. This system is more efficient, fast, reliable, user friendly. We started from how customers think and what they really want from their banks and in order to understand that we checked some banks website to find out about what exactly happening in the bank. Google has also helped in getting information about different types of bank accounts. The main challenge was to mutate all the information into an application format. We took a look through the links mentioned in the references part for getting some hints. Because time limitation is a concern and this project covers only the primary transactions that happen in a bank.

Challenges:

The biggest challenge faced during the course of the project was merging the back end changes to the front end since working with Javafx was a whole new experience and a new territory for both of us.

Task Distribution:

Modules	Tasks Done By
1. Login Module	Lijosh Abraham [C0764921]
2. Main view Module	Lijosh Abraham [C0764921]
3. New customer Registration module	Vibin Erapakkal Joby [C0765779]
4. Existing customer sign in Module	Vibin Erapakkal Joby [C0765779]
5. Existing customers Transaction module	Vibin Erapakkal Joby [C0765779]

Documentation

Flow chart Designs,Introduction, Project description and Conclusion	Lijosh Abraham [C0764921]
Rest of the description with explanation for Source code	Vibin Erapakkal Joby [C0765779]

REFERENCES

https://o7planning.org/en/11533/opening-a-new-window-in-javafx

https://www.callicoder.com/javafx-registration-form-gui-tutorial/

https://coderwall.com/p/ab5qha/convert-json-string-to-pretty-print-java-gson

https://stackabuse.com/reading-and-writing-json-in-java/

https://stackoverflow.com/questions/1844688/how-to-read-all-files-in-a-folder-from-java