Safe Bank

[1]Sri Harsha Peri, [1]Rajesh Kaireddy, [2]Varun Vuppala

[1]Information Systems, Northeastern University
[2]Software Engineering Systems, Northeastern University

{peri.s, kaireddy.r, vuppala.v}@northeastern.edu

ABSTRACT

Scope

This project, named "Safe Bank Internet Banking," aims to be a secure Internet banking application that solves problems for banks and customers. The application offers multiple banking services on a single platform, providing convenience and saving time for customers. Additionally, it ensures the security of customer data by preventing fraudulent or unauthorized access to it.

Implementation

The workflow to integrate the backend services into the event handlers in the JavaFx UI involves several classes and interfaces from several packages, which indicate a layered application. As soon as an event gets triggered, we initially invoke methods from classes defined in the services package, which implicitly call the protected methods defined in the respective DAO class (Data Access Object) in the dao package. The methods in the DAO classes implement all the CRUD operation transactions and manipulate the data in the database. A validation middleware from the validations package is used before processing any service to ensure every transaction gets valid input. A class named DatabaseConnectionFactory implements two major protected methods named executeQuery() and executeUpdate(), which perform the operations based on the query input. All the DAO classes implement DatabaseConnectionFactory to process all the transactions. We have used functional interfaces for two major transaction modes: credit and debit. All the transactional services implement these two methods anonymously, which doesn't generate a separate.class file, thereby optimizing the execution time. Several models were also implemented as a blueprint for the database schemas to provide more convenience in adding, updating, and retrieving data from the database. In validation logic and business logic, the Java Collection Framework has been primarily used as a data structure. Some of the most frequently used classes and interfaces are ListArrayList, Stack, MapHashMap, and SetHashSet. The Java Streams API has been used primarily for streaming the collection of data. Mapping to produce the required results using stream(). map() and filtering a subcollection from the given collection using stream(). filter() has been mostly used to simplify the process. ObservableList is mostly used in the UI classes to represent items in comboboxes. used the Date Time API to generate timestamps for specific fields in the schemas and models and to compare dates and make respective changes to reflect some crucial columns like the payment status, last payment date, and valid field values in the database.

I. PROBLEM DESCRIPTION

The main scope of this project is to provide seamless experience to users with convenient and secure banking experience and enhance customer experience with easy to use features. If there was no provision of internet banking, users would have to go to the bank for processing transactions like deposit and payments to self-account or other beneficiary accounts or to make some business transactions. But that would be consuming a lot of time causing inconvenience to users, inaccuracies and delays in reflecting payment statuses. To overcome challenges like these, online internet banking was introduced to solve the problems faced by the customers in offline banking and this project majorly aims to implement some of the functionalities that are being used in an online banking application system as per the industry standards. The transactions offered in this application are deposit to user owned accounts, fund transfers to self and beneficiary accounts, online payment using savings accounts and credit cards, minimum, custom and total amount bill payment of credit cards. Other features to provide more convenience to users are creating multiple savings accounts, requesting for a credit card, upgrading existing credit cards, reset password in and out of the user session and viewing all credit and savings transactions.

II. ANALYSIS (RELATED WORK)

As part of the analysis, the security used in this application was inspired from the functionality implementations that are used in real time banking websites. Security of this application was developed with paramount concern and hence the name SafeBank defines that this is a secured system providing solutions to customers and keeping their accounts and credit related information absolutely safe. Some of the interesting analyses include verification of email address rather than just validating it with email regex validation. The email verification involves a 2 factor authentication where a high security code is generated and sent to the provided email address. Users need to enter that code within a dialog with confirmation input. The provided input is verified against the generated code and all necessary input validations are done prior to code verification. Upon successful verification, a new user is created. There is no other way to proceed further for registration of a new user without verifying the security code. This adds a more secure layer to the application ensuring there are users with verified and trusted email addresses. The same technique was used in processing all transactions in the application.

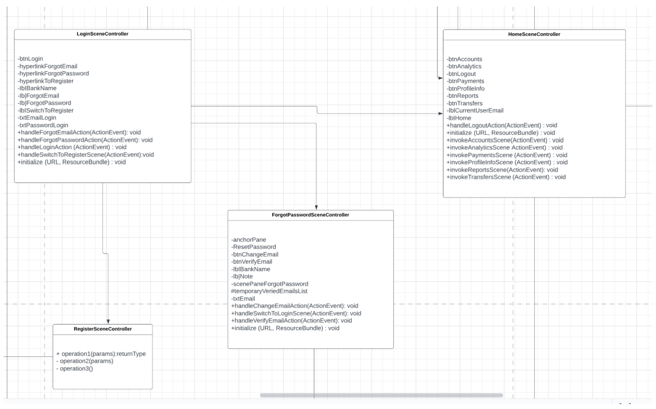

LoginSceneController
-btnLogin
-hyperlinkForgotEmail
-hyperlinkForgotPassword
-hyperlinkToRegister
-lblBankName
-lb|ForgotEmail
-lb|ForgotPassword
-lblSwitchToRegister
-txtEmailLogin
-txtPasswordLogin
+handleForgotEmailAction(ActionEvent): void
+handleForgotPasswordAction(ActionEvent): void
+handleLoginAction (ActionEvent) : void
+handleSwitchToRegisterScene(ActionEvent):void
+initialize (URL, ResourceBundle) : void
HomeSceneController
-btnAccounts
-btnAnalytics
-btnLogout
-btnPayments
-btnProfileInfo
-btnReports
-btnTransfers
-lblCurrentUserEmail
-lblHome
+handleLogoutAction(ActionEvent) : void
+initialize (URL, ResourceBundle) : void
+invokeAccountsScene(ActionEvent) : void
+invokeAnalyticsScene ActionEvent) : void
+invokePaymentsScene (ActionEvent) : void
+invokeProfileInfoScene (ActionEvent) : void
+invokeReportsScene(ActionEvent): void
+invokeTransfersScene (ActionEvent) : void
ForgotPasswordSceneController
-anchorPane
-ResetPassword
-btnChangeEmail
-btnVerifyEmail
-lblBankName
-lb|Note
-scenePaneForgotPassword
#temporaryVeriedEmailsList
-txtEmail
+handleChangeEmailAction(ActionEvent): void
+handleSwitchToLoginScene(ActionEvent): void
+handleVerifyEmailAction(ActionEvent): void
+initialize (URL, ResourceBundle) : void
RegisterSceneController
+ operation1(params):returnType
- operation2(params)
- operation3()

Figure 1. Login, Register and Forgot Password UML

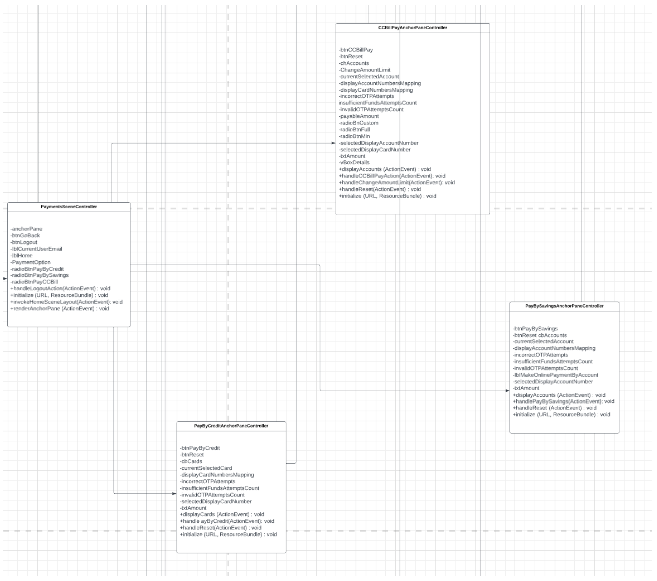

CCBillPayAnchorPaneController
-btnCCBillPay
-btnReset
-chAccounts
-ChangeAmountLimit
-currentSelectedAccount
-displayAccountNumbersMapping
-displayCardNumbersMapping
-incorrectOTPAttempts
insufficientFundsAttemptsCount
-invalidOTPAttemptsCount
-payableAmount
-radioBnCustom
-radioBtnFull
-radioBtnMin
-selectedDisplayAccountNumber
-selectedDisplayCardNumber
-txtAmount
-vBoxDetails
+displayAccounts (ActionEvent) : void
+handleCCBillPayAction(ActionEvent): void
+handleChangeAmountLimit(ActionEvent): void
+handleReset(ActionEvent) : void
+initialize (URL, ResourceBundle) : void
PaymentsSceneController
-anchorPane
-btnGoBack
-btnLogout
-lblCurrentUserEmail
-lblHome
-PaymentOption
-radioBtnPayByCredit
-radioBtnPayBySavings
-radioBtnPayCCBill
+handleLogoutAction(ActionEvent) : void
+initialize (URL, ResourceBundle) : void
+invokeHomeSceneLayout(ActionEvent): void
+renderAnchorPane (ActionEvent) : void
PayBySavingsAnchorPaneController
-btnPayBySavings
-btnReset cbAccounts
-currentSelectedAccount
-displayAccountNumbersMapping
-incorrectOTPAttempts
-insufficientFundsAttemptsCount
-invalidOTPAttemptsCount
-lblMakeOnlinePaymentByAccount
-selectedDisplayAccountNumber
-txtAmount
+displayAccounts (ActionEvent) : void
+handlePayBySavings(ActionEvent): void
+handleReset (ActionEvent) : void
+initialize (URL, ResourceBundle) : void
PayByCreditAnchorPaneController
-btnPayByCredit
-btnReset
-cbCards
-currentSelectedCard
-displayCardNumbersMapping
-incorrectOTPAttempts
-insufficientFundsAttemptsCount
-invalidOTPAttemptsCount
-selectedDisplayCardNumber
-txtAmount
+displayCards (ActionEvent) : void
+handle ayByCredit(ActionEvent): void
+handleReset(ActionEvent) : void
+initialize (URL, ResourceBundle) : void

Figure 2. Payments UML

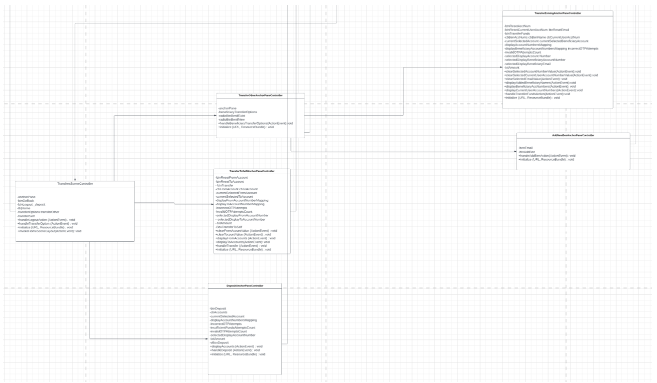


Figure 3. Transfers UML

III. System Design

As soon as the user is signed in, the user is provided with four options. Profile Information, Transactions History {Credit and Debit}, Payments{By Savings, By Credit, Pay Credit Card Bills} and Transfers{To Self, To Beneficiaries{Add New, Transfer to Existing}}.

UI Design

The UI consists of .fxml files and their respective Controller.files. All the files ending with Scene.fxml represents main features of this appication and files ending with AnchorPane.fxml represents implementation of sub features rendered inside the nested components of the main features layout.

Backend Design

The backend has multiple layers involved starting from taking the inputs from the user interface to completing the transactions into the database. The events in the JavaFX correspond to triggering a particular service defined in the services package. This service implicitly calls the methods from the DAO package to process the transaction. The implementation of database connection object to process transactions has been optimized to be less redundant. Initially every method in the DAO class was instantiating Connection, Statement and ResultSet objects and were being closed before finishing the execution of method. Later, we found that boiler plate code was repetitive and we have decided to encapsulate connection object within a class named DatabaseConnectionFactory and extended every DAO class to the same. Also replacing Statement object with PreparedStatement improved the performance of queries become for multiple record insertions, the query with PreparedStatement object is compiled only once.

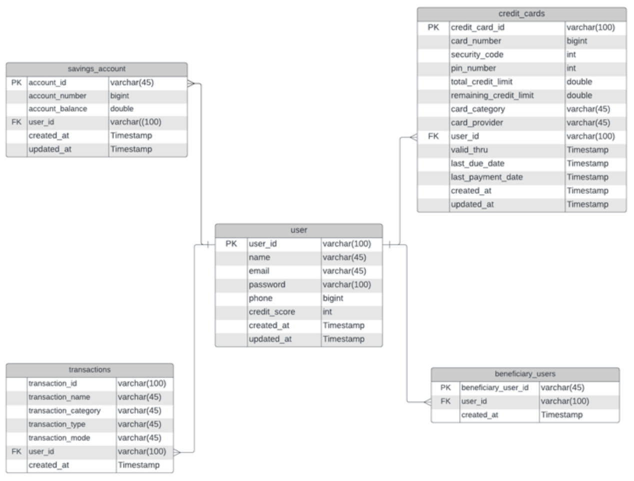


Figure 4. Database Schemas Design

IV. Implementation

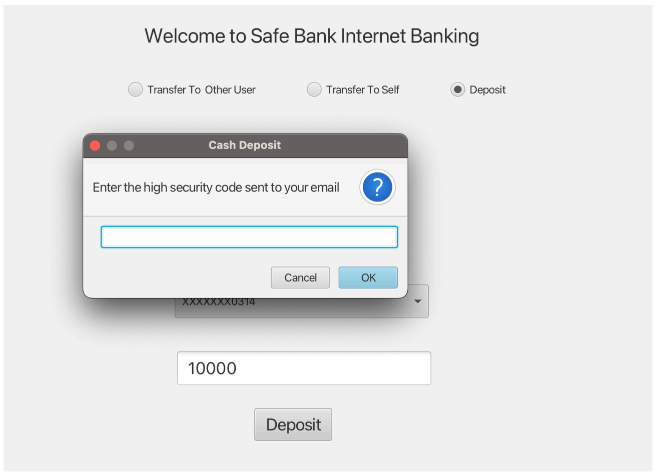
1. The login layout contains links to register new user and forgot password layout.
2. The register layout contains fields name, phone, email and password, two buttons "verify email" and "register" The register button is freezed until the user email is verified. After entering the email input and clicking on verify email, a randomly generated 6 digit high security code is sent to the input email. User need to enter the code sent in the email and it is verified against the code generated in the application.
3. After email gets verified, new user can be registered using register button and corresponding input validations are done
4. There is another option called forgot email clicking on which a dialog with input prompt appears where user need to enter

the registered contact number. If the input matches a record in the database, registered user email appears.

5. For forgot password, user need to verify the email first with OTP verification and then reset password window appears. For resetting password within the logged in user session, a dialog prompt appears to type the current password. Upon typing correct password, user gets the option to reset password. Failing to enter correct password for 3 times results in automatic session timeout due to too many incorrect password attempts and login screen appears back.
6. In the Profile Information section, there are number of options like viewing credit score, requesting for new credit card, changing name, phone and password, and creating new savings account.
7. User can create a maximum of 6 savings accounts under one registered email address. Attempting to create further results in a dialog popup giving user information that no more further accounts can be created.
8. Initially user is given 620 credit score and for creating every new account, credit score gets boosted by 20 points. After creating 6 accounts, user gets a credit score of 720. Then user can request for a new credit card. Upon requesting, user gets a BURGUNDY card approved by the bank.
9. There are certain criteria for approval of credit card. There are three categories of credit card offered to the user upon request. BURGUNDY with a total credit limit of USD 10,000, GOLD with a limit of USD 20,000 and PLATINUM with a limit of USD 30,000.
10. If a user has no credit card but a credit score eligible to get a PLATINUM card, still the user would be offered a BURGUNDY card first. Depending on the credit transactions and timely repayments, credit score gets boosted further and then user can upgrade step by step as in from BURGUNDY to GOLD to PLATINUM
11. User can view the credit and account transactions in the transactions panel. Account transactions are grouped by account number. Account numbers in the schema have unique key constraint and a 11-digit account number is randomly generated automatically while creating a new account. A 16-digit card number is generated randomly getting categorizing under CARD_PROVIDERS like VISA, MASTER CARD, DISCOVERY and AMERICAN EXPRESS.
12. In the payments section, user can make online payments through accounts, credit card and also pay credit card bills (if credit card gets approved). The three comfortable options to pay the credit card bill are Pay Total, Pay Minimum and Pay Custom amount.
13. Every credit transaction has the due date timestamp two mins after the timestamp of the transaction created. Based on the timestamp of the repayment of user, the value of a column named payment_status in the transactions table changes accordingly.
14. If a user has some amount left to pay and if the due date is yet to come, payment_status reflects PENDING. If a user has paid full amount before the due date, the payment_status reflects IN_TIME. If a user has some amount left to pay and if the due date has crossed, payment_status reflects LATE_YET_PENDING. If a user has paid full amount after the due date, the payment_status reflects LATE.
15. In the transfers section, user can transfer money to self and to beneficiary. In transfer to beneficiary, user has an option to add new beneficiary and then send to existing. However, adding a beneficiary is a must before transferring the funds to beneficiary.
16. User can also deposit the amount to owned accounts.
17. For all transactions including online payment by credit, by debit, paying credit card bill, deposit, transfer to beneficiary and self, verification of the high security code sent to the email is a must. Considering these factors, we believe we have considered few of the many crucial factors that are being practiced in the real time banking system design with major concern of consistency of transactions data.

V. EVALUATION

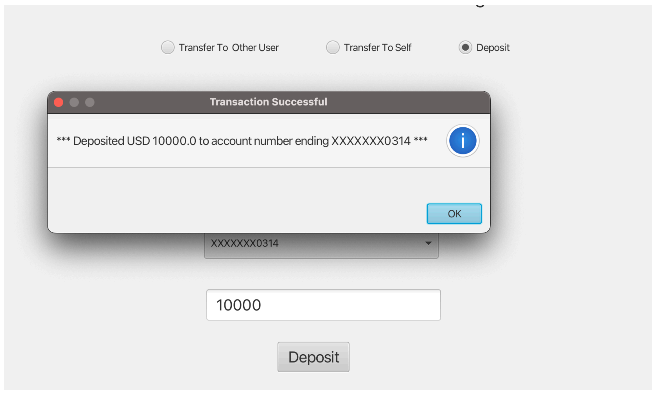
- **The screenshots of sample run and the explanation**


Welcome to Safe Bank Internet Banking

Cash Deposit

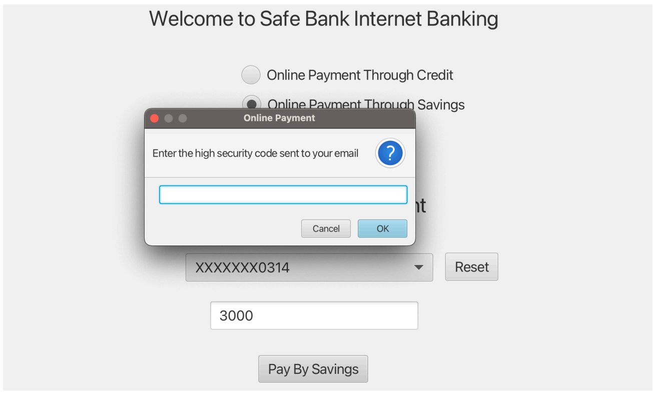
Enter the high security code sent to your email

Figure 5. Deposit OTP Dialog


Transaction Successful

*** Deposited USD 10000.0 to account number ending XXXXXXX0314 ***

Figure 6. Successful Deposit


Welcome to Safe Bank Internet Banking

Online Payment Through Credit

[✓ Online Payment Through Savings]

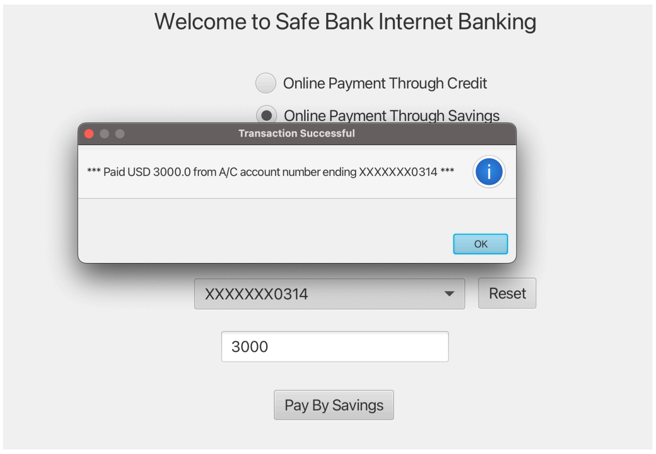
Online Payment

Enter the high security code sent to your email

XXXXXXX0314

3000

Figure 7. Pay By Savings Dialog


Welcome to Safe Bank Internet Banking

Online Payment Through Credit

[✓ Online Payment Through Savings]

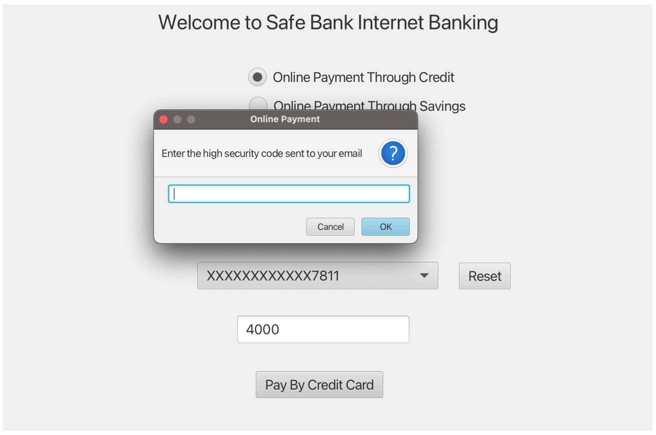
Transaction Successful

*** Paid USD 3000.0 from A/C account number ending XXXXXXX0314 ***

XXXXXXX0314

3000

Figure 8. Successful Payment by Savings


Welcome to Safe Bank Internet Banking

[✓ Online Payment Through Credit]

Online Payment Through Savings

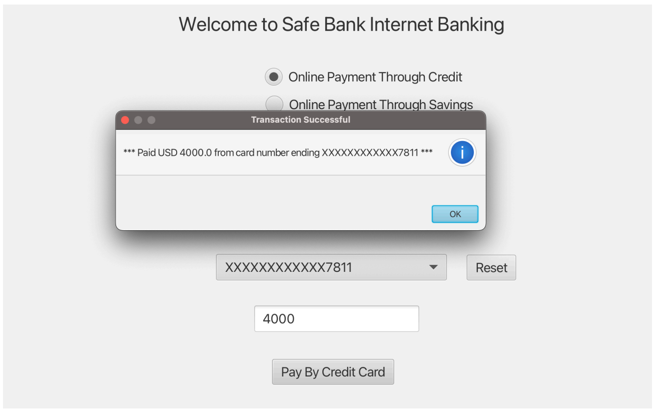
Online Payment

Enter the high security code sent to your email

XXXXXXXXXXXX7811

4000

Figure 9. Pay By Credit Dialog


Welcome to Safe Bank Internet Banking

[✓ Online Payment Through Credit]

Online Payment Through Savings

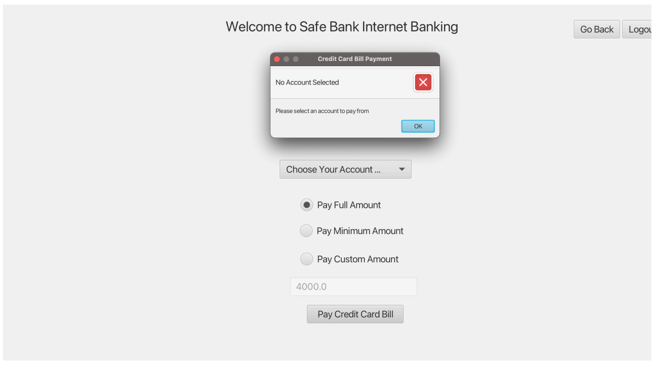
Transaction Successful

*** Paid USD 4000.0 from card number ending XXXXXXXXXXXX7811 ***

XXXXXXXXXXXX7811

4000

Figure 9. Successful Pay by Credit


Welcome to Safe Bank Internet Banking

Credit Card Bill Payment

No Account Selected

Please select an account to pay from

Choose Your Account ...

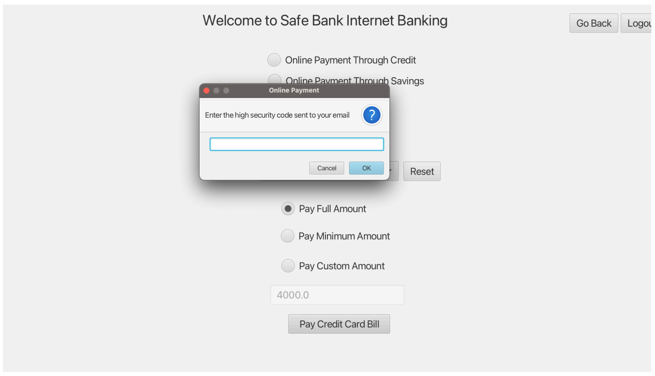
[✓ Pay Full Amount]

Pay Minimum Amount

Pay Custom Amount

4000.0

Figure 10. Error Message if no account is selected


Welcome to Safe Bank Internet Banking

Online Payment Through Credit

Online Payment Through Savings

Online Payment

Enter the high security code sent to your email

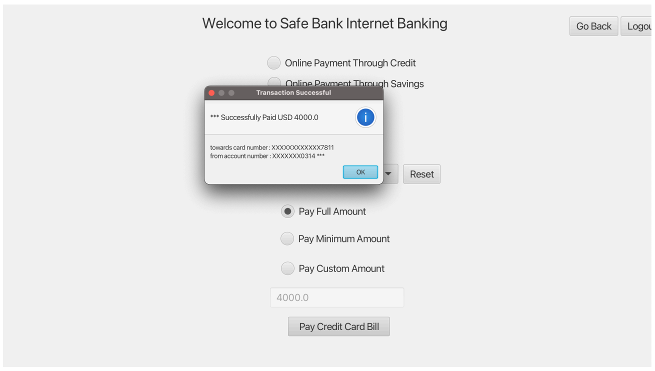
[✓ Pay Full Amount]

Pay Minimum Amount

Pay Custom Amount

4000.0

Figure 11. Successful Payment of Credit Card Bill


Welcome to Safe Bank Internet Banking

Online Payment Through Credit

Online Payment Through Savings

Transaction Successful

*** Successfully Paid USD 4000.0

towards card number : XXXXXXXXXXXX7811
from account number : XXXXXXX0314 ***

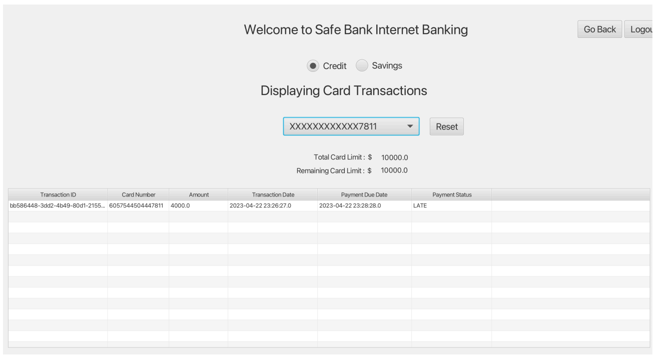
[✓ Pay Full Amount]

Pay Minimum Amount

Pay Custom Amount

4000.0

Figure 12. Successful Payment of Credit Card Bill



Welcome to Safe Bank Internet Banking

Credit [✓] Savings

Displaying Card Transactions

XXXXXXXXXXXX7811

Total Card Limit : $ 10000.0
Remaining Card Limit : $ 10000.0

Transaction ID	Card Number	Amount	Transaction Date	Payment Due Date	Payment Status
bb586448-3dd2-4b49-80d1-2155...	6057544504447811	4000.0	2023-04-22 23:26:27.0	2023-04-22 23:28:28.0	LATE

Figure 13. Payment Status of paid credit card transactions

- **Comparison between our work to other people's work**

Working on a Banking System was something which 3 of us were very curious about and that grouped us together to form a team for this project. Even though we had a well planned roadmap in our minds, we were curious enough to refer to projects done by other people on GitHub and YouTube to understand the features, the security level and the edge cases that they have implemented. After referring to several projects, we have decided to go extra miler as compared to their work and hence decided to implement OTP verification in our system. Also, many other projects were not even focusing on user authentication properly. Looking at these factors, we took the most important feature to gain a customer's trust very seriously in our application and have done password encryption in our system using Bcrypt password hashing library.

- **Our study with real time users and their feedback on our application**

We also spoke to 4 people working in the banking sector and they have significant experience working as a software engineer and have also contributed to the many features in the frontend and backend development of the bank website. They liked our validations and attention to detail in implementation of our transaction features very much. However, they have suggested a few things following which might have added more value to our application. Using test cards from a payment gateway service provider such that paying credit cards bills and repayment would have looked real. Also developing the backend in a cleaner way with all thread safe classes would have boosted up performance even way better. We are grateful to have received such valuable and useful feedback and would surely like to apply even after the project submission and make this system more reliable.

VII. Discussion (Reflection)

By carefully implementing all the features, we were able to process successful funds transfer from one account to another, observe the changes in the account balance and card balance immediately, and consistency in transaction data. For self-transfer, we get two records in the transactions table. One record is for amount debit and another record is for amount credit. We ensured that the timestamp of credit comes only after debit by putting the credit transaction thread in wait() state and making the debit transaction thread send notify() after processing. The results were also very accurate. We have handled all exceptions carefully

VIII. Conclusions and Future Work

- **Advantages or benefits of using our solution**

  For a user who is new to using our application, we are sure to win the trust of a user as we are securely storing user's credentials by using password encryption technique, for email verification, otp verification for transactions and consistency in transaction data for credit card bill payments. This makes the user believe how valuable the customer is to us. Our most favorite part of development is validating the inputs with all edge cases. For a programmer who is new to banking application development, our application would give detailed insights on what are all the edge cases which need to be considered.

- **Problems found but not yet explored in our project**

  Much more depends on factors for credit score adjustment. Currently, we are adding 20 points for account creation and payment through credit card, 10 points for deposit and online payment through savings. Reducing 20 points for late payment and 30 points for using the credit card with past payment due and payment_status from transactions as LATE_YET_PENDING.

- **If our team has more time, what would we like to improve?**

  If our team would be given more time, we would like to implement a more complex reports downloading feature and analytics dashboard which gives the statistics based on the user's credit and debit transactions, payment_status of past bills and credit score. Based on the user's average usage of credit card, we would like to make predictions of their future usage and categories where they could be spending more and suggest a spending plan for users to ensure they maintain a balanced credit score and financial profile.

IX. Job Assignment

- **Peri Sri Harsha : Transfers (Beneficiary Transfer, Self-Transfer, Deposit).**

  Harsha has worked on layout designs of Beneficiary Transfer, Self Transfer and Deposit. He also designed the schemas of transactions and savings_accounts. He has written functions in TransactionsDAO for updating account balance of sender account holder and receiver account holder records in savings_account tables. He has also written service to run a query and add a transaction in the transactions table. He has contributed his idea of reducing the redundant code by encapsulating the DatabaseConnectionFactory with a private Connection instance and implemented the protected methods executeQuery() and executeUpdate().

- **Rajesh Kaireddy: User Authentication (email verification and password encryption from inside and outside the application)**

  Rajesh has worked on layout designs of login, register and forgot password. He has written services to create a new user in the UsersDAO. He ensured that a user's email is being verified first before registering and resetting password. He did set up Profile Information section with a variety of options like view credit score, request new credit card, update user name and phone number and create new savings account. Rajesh has considered all edge cases carefully for implementing the validation logic.

- **Varun Vuppala: Transactions(Card and Account Transactions):**

  Varun has worked on layout designs of Card Transactions and Account transactions. He has written logic for conditional rendering of the content. If user has no credit card yet, in the transactions, only account transactions will be shown. If user has a credit card, two options will be shown whether to choose credit or debit transactions. Varun has also implemented a good idea of calling updateAllPendingTransactionsPaymentStatus() while initializing the Transactions component over our group idea to run a cron job for that which saves the number of threads used in the program.

- **Group Contribution: Payments(Pay By Credit, Pay By Debit, Credit Card Bill Payment):**

  The Pay By Debit is a feature for initiating online payment through savings account. This feature has been implemented by Rajesh. The Pay By Credit is a feature for initiating online payment through credit card. This feature has been implemented by Varun. The credit card bill payment feature in this section has been implemented by Harsha. This is the most complicated part of our application where after a credit transaction, a record is inserted in transactions table with status PENDING. Then a service is written and executed such that it fetched all the PENDING status transactions from transactions table and updates the status to IN_TIME or LATE or LATE_YET_PENDING by comparing the due date with the current date.

REFERENCES

[1] **JavaFX documentation:** This is the official documentation provided by Oracle that covers everything from getting started with JavaFX to more advanced topics.

[2] **JavaFX tutorials:** There are many online tutorials available that helped us build this project. We watched many YouTube videos of Bro Code and classroom tutorials of professor. click here

[3] **JavaFX Examples:** Oracle provides several JavaFX samples that demonstrate various JavaFX features. We have referred few of the references on GitHub which gave us a deeper insight on how to switch from one scene to another. We could achieve this with the help of the following GitHub link. click here

[4] **Books**: There are several books available on JavaFX that provide a more in-depth coverage of the subject.

1) "JavaFX 8: Introduction by Example" by Carl Dea, Mark Heckler, Gerrit Grunwald, Jose Pereda and Sean Phillips - This book is a great introduction to JavaFX and covers many of its features through practical examples.
2) "Pro JavaFX 9: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients" by Johan Vos, Weiqi Gao, James Weaver and Stephen Chin - This book is an in-depth guide to JavaFX 9 and covers everything from basic concepts to advanced topics such as 3D graphics and media.
3) "JavaFX 8: Building Rich Client Applications" by Sven Ruppert and Jim Weaver - This book provides a comprehensive guide to developing JavaFX applications, including topics such as UI controls, layout, and data binding.
4) "JavaFX for Dummies" by Doug Lowe - This book is a great resource for beginners to JavaFX and covers the basics of JavaFX development in an easy-to-understand way.
5) "Core Java Volume II: Advanced Features" by Cay S. Horstmann - This book is not specifically about JavaFX, but it covers many advanced Java features that are relevant to JavaFX development, such as concurrency, network programming, and security.

[5] Open-source projects: There are many open-source projects available on GitHub that use JavaFX.
https://github.com/rissandimo/JavaFX-Bank-Application
https://github.com/dmpe/BankApp