Online Banking System - Bank of Computer Science (BCS)

CPS 406 - Introduction to Software Engineering

Final Report

Team 08, Section 01

Final Phase: System Documentation

Prepared by:
Mohamad Nakouzi (501094772)
Danny Guan (501091498)
Shivesh Mahato (501111096)
Mykhaylo Batashov (501092777)
Kai Adams (501080302)
Reynold Villasenor-Doan (501031610)

April 1, 2023

TABLE OF CONTENTS

Part I: Requirements Analysis

1.1 System Domain and Objectives (Introduction)

The goal/purpose of this online banking system is to allow customers to perform financial transactions and online banking via the internet without having to visit a branch. This system is within the domain of banking, and it will involve functionality to allow users to perform various banking needs without having to leave the comfort of their homes.

The system is set to enable encryption of data upon further releases to ensure users' privacy and security. The emails, card numbers, and bank account numbers are unique for every user, and password requirements are set to ensure that passwords are not compromised. This system is created to set the privacy and security of its customers as a top priority.

The main objective is to provide the users (specifically computer science student users) of our system with as much banking functionality as possible in order to facilitate managing accounts and transactions.

This project was designed in the Java programming language using best practices of object oriented programming in order to create a modular implementation of the banking system (least coupling and maximum cohesion). Task division and gantt charts are included at the end of this report.

1.2 System Actors/Components

The following is a sample list of the major actors/components of the online banking system:

Major Actors/Components	Major Responsibilities and Roles
Clients with Account (CA)	CA are the main actors of our system; they have access to all the functionality within the system, from managing their personal accounts and making and receiving payments, to viewing current balance and transaction history. CA will also be able to book meetings with admins, request assistance from a customer service representative, make system complaints, and report suspicious activity, and more.
New Clients (NC)	NC would be able to register for an account if they do their banking with the Bank of Computer Science (BCS). Registering for an account would allow them full functionality of the system, and they would become CA.
Admins (AD)	AD receive and view meeting requests from users requesting a virtual/in-person meeting, as well as authorize and check suspicious activity reported.
Customer Service Representative (CSR)	CSR agents will be able to view and assist any customers' requests for assistance, and support the customer and fulfill their request.

Maintenance Team (MT)	MT deals with system complaints made by CA; they need to respond to the complaint/request and do any changes they find would improve the system.
Recipients (RC)	RC will have any transfers sent to them automatically deposited into their chequing account as long as they are a member of our bank (BCS).
Bank Automated (BA)	BA is the automated section of the system, and it will deal with all the logic and automatic authorization done by the system.

1.3 Discussion about the Solution

The online banking system was designed such that it provides customers with an easy-to-use interface in order to allow all users the convenience of being able to perform the transactions/account management necessary.

While the system does not currently implement a design where investments can be made online, this can be implemented in the future, depending on how users find the online system to be. The current design maintains a minimalistic, accessible design with the major functionalities, and can be further improved and extended as more users begin utilizing the system online.

1.4 Content of the Current Document and Glossary

Category	**Identifier**
Use Case	UC#
Assumption	A#
Responsibilities	R#
Functional Requirements	FR#
Non Functional Requirements	NFR#
Scenario Textual Description (Use-Case Scenarios)	STD#

1.5 Table of Functional Requirements (FR)

ID	Description: FRs for Client With Account (CA)	Traceability
FR#01	Each CA with an account can login through the initial login page	R#01
FR#02	Each CA can monitor and control the funds of the bank account(s) they have registered under their account (e.g. chequing, savings)	R#10
FR#03	Each CA can transfer funds to their other accounts	R#07
FR#04	Each CA can pay bills through online bank transfers to any recipient with a valid bank number	R#08
FR#05	Each CA can view their transaction history by account	R#09
FR#06	Each CA can contact a customer service representative (CSR) support via service requests	R#03
FR#07	Each CA can report system problems to the maintenance team	R#02
FR#08	Each CA can logout of their account	R#01
FR#09	Each CA can manage their settings, whether profile, privacy and security, notification settings.	R#11
ID	**Description: FRs for New Clients (NC)**	**Traceability**
FR#10	New Clients without accounts can create register for accounts through the login page to become a client with account	R#12
ID	**Description: FRs for Admins (AD)**	**Traceability**
FR#11	Admins/Managers can view client data to check for suspicious activity	R#14
FR#12	Admins can view and access meeting requests from a client through a secondary connection	R#13

ID	Description: FRs for Customer Service Representative (CSR)	Traceability
FR#13	Each CSR can view a service request sent in by a client and help them	R#21
FR#14	Each CSR can access limited client information to help them aid the client (e.g. account type, number of accounts, email address)	R#21
FR#15	Each CSR can connect with a client through a secondary connection to provide remote assistance (e.g. phone, email)	R#21
ID	**Description: FRs for Maintenance Team (MT)**	**Traceability**
FR#16	The Maintenance Team can view client complains about system problems	R#15
FR#17	The Maintenance Team can send a response to the client who issued a complaint through the customer's preferred notification settings	R#15
FR#18	The Maintenance Team will solve system issues based on client submissions as is deemed necessary	R#15
ID	**Description: FRs for Recipients (RC)**	**Traceability**
FR#19	Recipients get money automatically deposited into their chequing account as long as they are part of BCS.	R#16
ID	**Description: FRs for Bank AUTOMATED (BA)**	**Traceability**
FR#20	The bank (automated) will authorize logins, registers, transfers, and bill payments	R#19
FR#21	The bank (automated) will calculate balances after transfers and deposits, and send them back for display, as well as update transaction history.	R#17, R#18
FR#22	The bank (automated) will use logic to determine if an invalid transaction is occurring (e.g., a transfer larger than the account balance)	R#17

Functional Requirements of the Client With Account (CA):

1. Each CA with an account can login through the initial login page
2. Each CA can monitor and control the funds of the bank account(s) they have registered under their account (e.g. chequing, savings)
3. Each CA can transfer funds to their other accounts
4. Each CA can pay bills through the service and set payees
5. Each CA can view their transaction history by account
6. Each CA can contact a customer service representative (CSR) support via service requests
7. Each CA can report system problems to the maintenance team
8. Each CA can logout of their account
9. Each CA can manage their settings, whether profile, privacy and security, or notification settings

Functional Requirements of New Clients (NC):

1. New Clients without accounts can create register for accounts through the login page to become a client with account

Functional Requirements of the Admin (AD):

1. Admins/Managers can view client data to check for suspicious activity
2. Admins can view meeting requests from a client through a secondary connection

Functional Requirements of the Customer Service Representative (CSR):

1. Each CSR can view a service request sent in by a client and help them
2. Each CSR can access limited client information to help them aid the client (e.g. account type, number of accounts, email address)
3. Each CSR can connect with a client through a secondary connection to provide remote assistance (e.g. phone, email) as per the notification preferences of the user

Functional Requirements of the Maintenance Team (MT):

1. The Maintenance Team can view client complains about system problems
2. The Maintenance Team can send a response to the client who issued a complaint as per the notification preferences of the user (e.g., SMS or email)
3. The Maintenance Team will solve system issues based on client submissions

Functional Requirements of the Recipients (RC):

1. Recipients get money automatically deposited into their chequing account when they receive a bank or e-transfer that is valid

Functional Requirements of Bank AUTOMATED (BA):

1. The bank (automated) will authorize logins, new client registering, transfers, and bill payments
2. The bank (automated) will calculate balances after transfers and deposits, and update transaction history
3. The bank (automated) will use logic to determine if an invalid transaction is occurring (e.g., a transfer larger than the account balance

1.6 Table of Non Functional Requirements (NFR)

ID	Description	Traceability
NFR#01	**Usability:** The system should be user-friendly and easy to navigate, with a clear and intuitive interface.	STD#14, STD#15, STD#17, STD#18
NFR#02	**Reliability and availability:** The system should be available 24/7 and have minimal downtime, with reliable and secure transactions.	STD#14, STD#15, STD#18
NFR#03	**Performance:** The system should respond quickly to user requests (Low Cost functions), with low latency and fast transaction processing times (Optimizing the database design for fast data retrieval and storage, and using caching techniques to reduce the number of database queries).	STD#08, STD#14, STD#15, STD#16
NFR#04	**Robustness:** The system should be able to handle incorrect and unexpected input, and have built-in safeguards to prevent errors and fraud.	STD#01, STD#02, STD#08, STD#09, STD#10, STD#11, STD#12, STD#13
NFR#05	**Security:** The system should protect sensitive financial information and have robust security measures in place, such as encryption and secure authentication.	STD#01, STD#02, STD#04, STD#05, STD#06, STD#07, STD#08, STD#10, STD#11, STD#12, STD#13, STD#16, STD#17
NFR#06	**Maintainability:** The system should be easily maintainable, with clear documentation and regular software updates.	STD#06, STD#07, STD#18
NFR#07	**Extendibility:** The system should be designed to be easily extendable to accommodate future changes and growth by being as modular as possible.	STD#06, STD#07, STD#09, STD#10, STD#11

1.7 Table of Assumptions (A)

ID	Description	Traceability
A#01	NC and CA will have a stable internet connection when utilizing the online banking system	
A#02	NC and CA will have a device that can connect to the internet and search the web for the online banking system	
A#03	AD have access to user accounts and their information and have the ability to change them at user's request	
A#04	NC who make a new account are already affiliated with the bank, i.e. they have an account with the bank and a debit card and use those to sign up for their new accounts	STD#01
A#05	Etransfer and bank transfer recipients only have the options for the auto-deposit functionality (money is automatically deposited to their chequing account). The system could be extended to enable security questions for non-automatic deposits in the future.	STD#06, STD#10, STD#11
A#06	CA can only have a maximum of one chequing and one savings account. The system could be extended to enable more accounts in the future.	

1.8 Table of Responsibilities (R)

ID	Description	Traceability
	Responsibilities of the Client with Account (CA)	
R#01	CA will be able to log into (and out of) their accounts using their personal credentials to access their bank accounts.	STD#02, STD#03
R#02	CA has the ability request system changes that can be implemented by the maintenance team	STD#18
R#03	CA will be able to request assistance from a customer service representative.	STD#17
R#04	CA will be able to locate a nearby branch to visit.	STD#09
R#05	CA will be able to request a meeting with AD, which will be received by AD.	STD#08
R#06	CA will be able to report any activity that they deem suspicious on their account, and AD will be able to review it and determine the next course of action, if necessary.	STD#16
R#07	CA will be able to transfer money between their chequing and saving account.	STD#07
R#08	CA will be able to make/receive payments via e-transfer and bank transfer from other users.	STD#10, STD#11, STD#12, STD#13
R#09	CA will be able to view all prior transactions done from their accounts in the last month.	STD#15
R#10	CA will be able to view their current balance in both their chequing and saving accounts.	STD#14
R#11	CA will be able to change their account settings such as their profile details, notifications and security settings.	STD#03, STD#04, STD#05, STD#06

	Responsibilities of the New Clients (NC)	
R#12	NC will be able to make an account to access the online banking services.	STD#01
	Responsibilities of the Admins (AD)	
R#13	AD views and responds to meeting requests for the CA.	STD#08
R#14	AD investigates the CA suspicious activity reports and determines the next course of action to be taken.	STD#16
	Responsibilities of the Maintenance Team (MT)	
R#15	Review the CA's requests for system changes and develop solutions to those requests.	STD#18
	Responsibilities of the Recipients (RC)	
R#16	Receive the payments made by CA.	STD#10
	Responsibilities of the Bank Automated (BA)	
R#17	Transfers exact amount of money to the account requested by CA and carries out all necessary logic.	STD#07, STD#15
R#18	BA adds the money that CA receives to the appropriate account.	STD#11
R#19	BA verifies the authenticity of the new and existing accounts by comparing them to bank accounts/login credentials in its database.	STD#01, STD#02, STD#06
R#20	BA transfers money sent by CA to the appropriate account	STD#10
	Responsibilities of the Customer Service Representative (CSR)	
R#21	CSR assists CA with any tasks when help is requested	STD#17

1.9 Table of Use Cases (UC)

ID	Name	Description	Traceability
UC#01	Make New Account (Register)	NC can register for a new account and choose their email and password. This account will be linked to their bank accounts and give them access to the functionalities of the system.	STD#01
UC#02	Login to Account	CA can log into their personal accounts using the email and password they set when they registered for the account.	STD#02
UC#03	Manage Account Settings	CA can manage their account settings and preferences on the system.	STD#03, UC#04, UC#05, UC#06
UC#04	Edit Profile	Extending 'Manage Account Settings', CA can edit their profile, e.g. they could change their address, change their phone number, and/or change their email.	STD#04, UC#03
UC#05	Manage Notifications	Extending 'Manage Account Settings', CA can set their notification preferences within settings, e.g. a user can choose to receive an email when their account balance is below a certain threshold.	STD#05, UC#03
UC#06	Privacy and Security	Extending 'Manage Account Settings', CA can change their privacy and security settings, e.g. a CA can change their password.	STD#06, UC#03
UC#07	Transfer Funds Between User Accounts	CA can transfer funds between the different accounts they have, e.g. CA can transfer money from their chequing account to their savings account	STD#07
UC#08	Request Meeting	CA can request a meeting with AD in order to discuss various banking issues, including opening new accounts, investing, or closing an account.	STD#08

UC#09	Find Nearby Branch	CA can find nearby branches/ATMs of the Bank Of Computer Science (BCS) to visit.	STD#09
UC#10	Make Payment	CA can make payment (either e-transfer or bank transfer) using the funds in their accounts.	STD#10, UC#12, UC#13
UC#11	Receive Payment	CA can receive payment (either e-transfer or bank transfer) from other people, whether people using the same bank or different bank.	STD#11, UC#12, UC#13
UC#12	E-transfer	CA can make use of the e-transfer functionality in order to send/receive money using their email as an identifier.	STD#12, UC#10, UC#11
UC#13	Bank Transfer	CA can make use of the bank transfer functionality in order to send or receive money from other users using their bank accounts number	STD#13, UC#10, UC#11
UC#14	View Balance	CA can view their current balance in each of their accounts, and the balance updates with each transaction being made.	STD#14
UC#15	View Transaction History	CA can view their transaction history for each of their accounts by selecting the account. The last 6 transactions are shown per account.	STD#15
UC#16	Report Suspicious Activity	CA can report suspicious activity on their account (e.g., a purchase they did not make) to AD by submitting a report	STD#16
UC#17	Request Assistance	CA can contact a CSR in order to request assistance, such as making an inquiry.	STD#17
UC#18	Request System Changes	CA can request system changes on the system and make suggestions to make the system better. This request is reviewed by the MT and dealt with case-by-case.	STD#18

1.10 Use Case Diagrams

Below are 2 use case diagrams of the system, one with secondary links to secondary actors, and one without. Since all the use cases involve bank automated, whether for verification or for logic, the secondary links make diagram 1.10.1 difficult to read. However, that diagram is complete with all the links.

For readability, diagram 1.10.2 was created, which removes the secondary links in order to allow clear viewing of the primary actors' use cases and what they include and extend. All of CA's use cases include log in, as the user cannot carry out any of the actions they want without logging into the system.

It can be seen by the two diagrams below which use cases are carried out, and which actors are involved in which. It encompasses the whole system's use cases described in 1.9, and further explained in 1.11.

1.10.1 THIS PAGE WILL BE EXCLUDED FROM THE SUBMISSION TO MAKE SPACE FOR THE USE CASE DIAGRAMS PDF TO BE MERGED.

1.10.2 THIS PAGE WILL BE EXCLUDED FROM THE SUBMISSION TO MAKE SPACE FOR THE USE CASE DIAGRAMS PDF TO BE MERGED.

1.11 Use Case Scenarios (STD)

STD#01: Allow user to create a unique account	**Traceability**
Use Case Name: UC#01	
Scope & Description: Accepts information inputted by the user. This information is then used to create an account for the user and link their bank account to it.	
Primary Actor: New Clients (NC)	
Stakeholders & Interests: NC: to enter information for a new, unique account BA: to process the information entered to create a unique account	
Pre-Condition: N/A	
Triggering Event: Initiated by the user through the UI interface, when the NC selects 'Register' on the login screen	
Main Scenario Steps: 1. **(R#12)** NC opens to the login screen. 2. NC clicks the 'Register' button. 3. Inputs their personal information, email, debit card number, phone number, and selects a new password. 4. **(R#19)** BA verifies card information, email, and password 5. (i) If it's all valid , it adds user information to its database and gives access to the account (NC becomes CA). (ii) If it does not exist, the user is prompted to either try again.	R#12, R#19
Post-Condition: An account is created for the client	
Extensions: No Alternatives.	
Non-Functional Requirements: Security, Robustness	NFR#04, NFR#05
Frequency of Occurrence: Executes whenever a new user wishes to create an account	
Miscellaneous Comments: N/A	

STD#02: Allows user to log in to their account	**Traceability**
Use Case Name: UC#02	
Scope & Description: Accepts login information from the user. If correct, allows the user access to their account	
Primary Actors: Clients with Account (CA)	
Stakeholders & Interests: CA: to enter in login information through the UI interface. BA: to review the login information and determine if it's valid. If valid, allows the user access to their account.	
Pre-Condition: An account has been made previously	
Triggering Event: User accesses the login screen and enters their credentials	
Main Scenario Steps: 1. **(R#01)** User logs in to account 2. **(R#19)** BA verifies if it a valid account (email and password exist in database) (i) If it is, user is redirected to the home page of their account (ii) If not, user is asked to try again	R#01, R#19
Post-Condition: The screen is updated and shows the account menu of the user.	
Extensions: No Alternatives.	
Non-Functional Requirements: Security, Robustness	NFR#04, NFR#05
Frequency of Occurrence: Whenever the user wishes to login	
Miscellaneous Comments: N/A	

STD#03: Allow User to Manage their Account	**Traceability**
Use Case Name: UC#03	
Scope & Description: Provides a multitude of options to user in regards to managing their account and accepts selections and inputs from the user to manage their account	
Primary Actors: Clients with Account (CA)	
Stakeholders & Interests: CA: to navigate the UI and select what actions they would like to perform in regards to managing their account	
Pre-Condition: User has successfully logged in	
Triggering Event: Client logs into their account and selects the 'Settings' option.	
Main Scenario Steps: 1. **(R#01)** User logs in 2. Goes to setting and user is given three options: edit profile settings (**STD#04**), manage notification settings (**STD#05),** manage privacy & security settings (**STD#06**).	R#01, STD#04, STD#05, STD#06, R#11
Post-Condition: User has made a selection on what they want to do in regards to managing their account and saved those changes.	
Extensions: No Alternatives.	
Non-Functional Requirements: N/A	
Frequency of Occurrence: Whenever the user logs into their account and chooses to change their settings.	
Miscellaneous Comments: This is extended by STD#04, STD#05, STD06	

STD#04: Allow User to Edit Profile	**Traceability**
Use Case Name: UC#04	
Scope & Description: Provides the user with the capability to change their email, address, and/or phone number.	
Primary Actors: Clients with Account (CA)	
Stakeholders & Interests: CA: to navigate the settings menu and select which option they want to edit in their profile	
Pre-Condition: User has successfully logged in and selected 'Settings'	
Triggering Event: Client selects the 'Edit Profile' option within 'Settings'	
Main Scenario Steps: 1. From manage user account (**STD#03**), the user clicks Edit Profile 2. CA changes their information and saves.	STD#03, R#11
Post-Condition: User has edited their profile and saved those changes.	
Extensions: No Alternatives.	
Non-Functional Requirements: Security	NFR#05
Frequency of Occurrence: Whenever the user logs into their account and chooses to edit their profile.	
Miscellaneous Comments: This extends STD#03	

STD#05: Allow User to Manage Notifications Settings	**Traceability**
Use Case Name: UC#05	
Scope & Description: Provides the user with the capability to change their notification preferences.	
Primary Actors: Clients with Account (CA)	
Stakeholders & Interests: CA: to navigate the settings menu and select their notifications' preferences	
Pre-Condition: User has successfully logged in	
Triggering Event: Client logs into their account and selects the 'Settings' option	
Main Scenario Steps: 1. From manage user account (**STD#03**), user clicks Manage Notification Settings. 2. CA selects their desired notification and alert preferences and saves.	STD#03, R#11
Post-Condition: User has made a selection on what they want to do in regards to managing their account and saved those changes.	
Extensions: No Alternatives.	
Non-Functional Requirements: Security	NFR#05
Frequency of Occurrence: Whenever the user logs into their account and chooses to manage their notification preferences.	
Miscellaneous Comments: This extends STD#03	

STD#06: Allow User to Manage Privacy & Security Settings	**Traceability**
Use Case Name: UC#06	
Scope & Description: Provides the user with the capability to change their privacy and security settings.	
Primary Actors: Clients with Account (CA)	
Stakeholders & Interests: CA: to navigate the UI and select what actions they would like to perform in regards to managing their account	
Pre-Condition: User has successfully logged in	
Triggering Event: Client logs into their account and selects the 'Privacy & Security' option.	
Main Scenario Steps: 1. From manage user account (**STD#03**), user clicks 'Privacy Settings' 2. CA selects what information they would like to be shared and also has the option to change their password 2.1. a) If they change their password they input their old password and their new password twice b) BA checks to see if the current password is the same as the old password. If it is, the password is replaced with the new password	STD#03, R#19 R#11
Post-Condition: User has made a selection on what they want to do in regards to changing their privacy and security settings and saved those changes.	
Extensions: If functionality for e-transfer auto-deposit to be disabled was added, settings could be updated to allow the user to control that.	A#05
Non-Functional Requirements: Security, Extendability, Maintainability	NFR#05, NFR#06, NFR#07
Frequency of Occurrence: Whenever the user logs into their account and chooses to change their privacy & security settings.	
Miscellaneous Comments: This extends STD#03	

STD#07: Allow User to Transfer Funds Between Their Personal Accounts	**Traceability**
Use Case Name: UC#07	
Scope & Description: Allows client to enter in an amount and picks an account to transfer from and another account to transfer into	
Primary Actors: Clients with Accounts (CA)	
Stakeholders & Interests: Clients with Accounts (CA): to input an amount and pick which accounts to transfer amount between Bank Automated (BA): transfers the amount from one account to another	
Pre-Condition: The client has successfully logged in and has two accounts, one chequing one savings.	
Triggering Event: Client picks option to transfer funds	
Main Scenario Steps: 1. **(R#07)** CA requests funds to be moved from either their chequing to their saving and vice versa. 2. **(R#17)** BA takes this request and ensures it is possible by checking if the client has enough money 3. (i) If so, the money is transferred and the display and transaction history are updated (ii) If not, the user is given a warning message	R#07, R#17
Post-Condition: Amount is transferred and the accounts update with the new balances in them	
Extensions: If functionality was added to include more than 2 user accounts, then this use case can be extended to include transfer between various accounts, potentially including credit.	
Non-Functional Requirements: Security, Extendability, Maintainability	NFR#05, NFR#06, NFR#07
Frequency of Occurrence: Whenever the client chooses to transfer funds	
Miscellaneous Comments: N/A	

STD#08: Allow User to Request a Meeting With an Admin	**Traceability**
Use Case Name: UC#08	
Scope & Description: Allows users to request a meeting through the online system to meet with AD to discuss various topics and perform functionalities that might not be available through the online system.	
Primary Actors: Clients with Accounts (CA), Admins (AD)	
Stakeholders & Interests: CA: to submit a request to meet with an advisor AD: to view the request and reach out to the CA to set up a meeting	
Pre-Condition: Client has successfully logged in	
Triggering Event: Client has submitted a meeting request	
Main Scenario Steps: 1. **(R#05)** CA clicks to request a meeting 2. **(R#13)** AD receives the request and either accepts or declines the meeting 3. AD contacts the CA according to their notification preferences to reply to their request.	R#05, R#13
Post-Condition: AD reaches out to client according to notification preferences	
Extensions: No Alternatives.	
Non-Functional Requirements: Security, Robustness, Performance	NFR#03, NFR#04, NFR#05
Frequency of Occurrence: Whenever client requests a meeting	
Miscellaneous Comments: N/A	

STD#09: Allow User to Locate Nearby Branches	**Traceability**
Use Case Name: UC#09	
Scope & Description: Provides a list of bank branches in a certain area that is entered by the user	
Primary Actors: New Clients (NC) Clients with Accounts (CA)	
Stakeholders & Interests: NC & CA: enters in a location on the locator screen	
Pre-Condition: N/A	
Triggering Event: User navigates the the locator page on the website	
Main Scenario Steps: 1. **(R#04)** CA clicks the 'Find Us' button 2. A selection of branch locations across canada and the US are presented to the user	R#04
Post-Condition: A list of branches in the given area is returned and displayed.	
Extensions: As new branches are made available, they can be added to the system to give users more options.	
Non-Functional Requirements: Robustness, Extendibility	NFR#04, NFR#07
Frequency of Occurrence: Whenever the user wishes to locate branches and ATMs	
Miscellaneous Comments: The user does require an account to carry this out	

STD#10: Allow User to Make Payments	**Traceability**
Use Case Name: UC#10	
Scope & Description: The client can input an amount and choose to pay that amount to a bill or another recipient	
Primary Actors: Client with Account (CA), Recipients (RC)	
Stakeholders & Interests: CA: to input an amount and pick a recipient Bank Automated: Receives and processes amount and sends it to recipient RC: to accept the money sent to them	
Pre-Condition: Client successfully logged in	
Triggering Event: Client navigates to the payment page	
Main Scenario Steps: 1. User logs in and is redirected to their home page 2. **(R#08)** CA selects 'Transfer', and they pick one of two methods: (i) e-transfer or (ii) bank transfer.	STD#12, STD#13, R#08 R#16, R#20
Post-Condition: Money is sent to recipient	
Extensions: Auto-deposit enabled by default without an option to change this. If functionality was added to allow recipients to have auto-deposit enabled, this can be upgraded to allow for that functionality.	A#05
Non-Functional Requirements: Security, Robustness, Extendibility	NFR#04, NFR#05, NFR#07
Frequency of Occurrence: Whenever the client chooses to send money	
Miscellaneous Comments: This is extended by STD#12 and STD#13	

STD#11: Allow User to Receive Payment	**Traceability**
Use Case Name: UC#11	
Scope & Description: The client can receive payment by accepting the request in their account	
Primary Actors: Client with Accounts (CA)	
Stakeholders & Interests: CA: to access their account and accept the payment sent to them Bank Automated: to notify the user of the payment and deposit it for them when they accept it	
Pre-Condition: The client has successfully logged in	
Triggering Event: A sender sends the client money	
Main Scenario Steps: 1. **(R#08)** CA receives a payment through one of two methods: (i) e-transfer or (ii) bank transfer. This payment is automatically deposited into their chequing account.	STD#12, STD#13, R#08, R#18
Post-Condition: The client accepts the payment and has it deposited. The account updates with the new balance	
Extensions: Auto-deposit is the only option currently. If functionality was added to give users the option, then the receiving of the payment could involve a security check beforehand.	A#05
Non-Functional Requirements: Security, Extendibility, Robustness	NFR#04, NFR#05, NFR#07
Frequency of Occurrence: Whenever someone sends the client money	
Miscellaneous Comments: This is extended by STD#12 and STD#13	

STD#12: Allow User to Make/Receive E-Transfers	**Traceability**
Use Case Name: UC#12	
Scope & Description: The client can either send e-transfers to other people (even if they are part of a different bank) or receive e-transfers from other people (even if they are part of a different bank).	
Primary Actors: Client with Accounts (CA)	
Stakeholders & Interests: CA: to access their account and use the e-transfer functionality to send/receive money Bank Automated: (i) Receiving: to notify the user of the payment and deposit it for them when they accept it (ii) Sending: process the amount being sent to recipient and updates accounts accordingly	
Pre-Condition: The client has successfully logged in	
Triggering Event: The client receives or decides to make an e-transfer	
Main Scenario Steps: 1. Sending: a. From STD#10, if CA chose e-transfer, then they input RC email and the amount they would like to transfer b. CA selects which account they would like to transfer the money from, and clicks send to send money to RC 2. Receiving: a. From STD#11, if the sender chose to send money via e-transfer, the money is automatically deposited to the chequing account that is associated with the email of the CA (this could be seen in the system if the receiver is within BCS).	STD#10, STD#11, R#08
Post-Condition: (i) Receiving: the client accepts the money and it is added to their account (ii) Sending: money is sent to the recipient and the user's accounts are updated	
Extensions: No Alternatives.	
Non-Functional Requirements: Security, Robustness	NFR#04, NFR#05
Frequency of Occurrence: Whenever the client makes/receives an e-transfer	
Miscellaneous Comments: This extends STD#10 and STD#11	

STD#13: Allow User to Make/Receive Bank Transfers	**Traceability**
Use Case Name: UC#13	
Scope & Description: The client can either send bank transfers to other people (even if they are part of a different bank) or receive bank transfers from other people (even if they are part of a different bank; international transferring is included).	
Primary Actors: Client with Accounts (CA)	
Stakeholders & Interests: CA: to access their account and use the bank transfer functionality to send/receive money Bank Automated: (i) Receiving: to deposit the amount into their chequing account (ii) Sending: process the amount being sent to recipient and updates accounts accordingly	
Pre-Condition: The client has successfully logged in	
Triggering Event: The client receives or decides to make a bank transfer	
Main Scenario Steps: 1. Sending: a. From STD#10 If Bank transfer was chosen CA will need to input the RC bank account number and the amount they would like to send. b. CA the clicks send and the amount will be sent to RC 2. Receiving: a. From STD#11, if the sender chose to send money via Bank Transfer, the money is automatically deposited to the chequing account that is associated with the Account number of CA (this could be seen in the system if the receiver is within BCS)	STD#10, STD#11, R#08
Post-Condition: (i) Receiving: the client accepts the money and it is added to their account (ii) Sending: money is sent to the recipient and the user's accounts are updated	
Extensions: No Alternatives.	
Non-Functional Requirements: Security, Robustness	NFR#04, NFR#05
Frequency of Occurrence: Whenever the client makes/receives a bank transfer	
Miscellaneous Comments: This extends STD#10 and STD#11	

STD#14: Allow User to View Balance	**Traceability**
Use Case Name: UC#14	
Scope & Description: The client can access their account and view their balance in that account	
Primary Actors: Client with Accounts (CA)	
Stakeholders & Interests: CA: Login to their account Bank automated: To display the current balance in the account chosen and continuously update the balance	
Pre-Condition: The client has successfully logged in	
Triggering Event: The client logs into their account	
Main Scenario Steps: 1. **(R#10)** CA logs in and they see all their accounts balances on their homepage. 2. BA updates the balance and display whenever necessary	R#10
Post-Condition: The balance of the accounts is displayed on the homepage	
Extensions: No Alternative.	
Non-Functional Requirements: Usability, Reliability and Availability, Performance	NFR#01, NFR#02, NFR#03
Frequency of Occurrence: Whenever the client logs in	
Miscellaneous Comments: After each transaction and payment made or received, the balance is updated	

STD#15: Allow User to View Transaction History	**Traceability**
Use Case Name: UC#15	
Scope & Description: The client can access their account and view their transaction history for the latest 6 transactions	
Primary Actors: Client with Accounts (CA)	
Stakeholders & Interests: CA: to click on their account Bank automated: To display the transaction history for the last 6 transactions and update with each transaction	
Pre-Condition: The client has successfully logged in	
Triggering Event: The client clicks on their account	
Main Scenario Steps: 1. **(R#09)** CA selects one of their accounts to view their transaction history 2. **(R#17)** BA gathers the data and presents to to the CA, along with the bank balance and details about the account	R#09, R#17
Post-Condition: The transaction history of the account chosen is displayed	
Extensions: No Alternative.	
Non-Functional Requirements: Usability, Reliability and Availability, Performance	NFR#01, NFR#02, NFR#03
Frequency of Occurrence: Whenever the client clicks on their account	
Miscellaneous Comments: After each transaction and payment made or received, the transaction history is updated	

STD#16: Allow User to Report Suspicious Activity on Their Account	**Traceability**
Use Case Name: UC#16	
Scope & Description: Client files a report to notify admins of suspicious activity happening to their account	
Primary Actors: Client with Accounts (CA), Admins (AD)	
Stakeholders & Interests: CA: to navigate to the report page and fill out a report AD: to examine the report and take any necessary action	
Pre-Condition: Client has an account	
Triggering Event: Client fills out report and submits it	
Main Scenario Steps: 1. CA logs in and is redirected to their homepage 2. **(R#06)** CA selects the option that allows them to report suspicious activity 3. They write out a small summary about the suspicious activity they are experiencing 4. **(R#14)** AD receives and reviews the report then contacts CA according to their notification preferences.	R#06, R#14
Post-Condition: AD receives report and takes any action needed	
Extensions: No Alternatives.	
Non-Functional Requirements: Performance, Security	NFR#03, NFR#05
Frequency of Occurrence: Whenever the client encounters suspicious activity	
Miscellaneous Comments: N/A	

STD#17: Allow User to Request Assistance From Staff	**Traceability**
Use Case Name: UC#17	
Scope & Description: The client can request assistance from a customer support representative	
Primary Actors: Clients with Account (CA), Customer Support Representative (CSR)	
Stakeholders & Interests: CA: to click the button that requests assistance from the CSR BA: to send the request to the CSR to notify them CSR: to view the notification and reach out to the client to assist them	
Pre-Condition: N/A	
Triggering Event: Client clicks button to request assistance	
Main Scenario Steps: 1. CA logs in and is redirected to their homepage 2. **(R#03)** CA selects the option to request assistance from CSR 3. CSR receives this request and reviews it 4. **(R#21)** CSR assists CA in any problems they have by responding according to their notification preferences.	R#03, R#21
Post-Condition: CSR reaches out to the client to assist them	
Extensions: No Alternative.	
Non-Functional Requirements: Security, Usability	NFR#01, NFR#05
Frequency of Occurrence: Whenever the client is in need of assistance	
Miscellaneous Comments: Can only be done when logged in	

STD#18: Allow User to Request System Change	**Traceability**
Use Case Name: UC#18	
Scope & Description: The client can request a system upgrade/change to the maintenance team in order to add functionality/report bugs that the user finds	
Primary Actors: Client with Accounts (CA), Maintenance Team (MT)	
Stakeholders & Interests: CA: to report their request and explain the functionality they want added or the bug they want fixed MT: receives the report, reviews it, and works towards fixing the issue to make the user's experience better	
Pre-Condition: The client has successfully logged in	
Triggering Event: The client selects the option to request a system change	
Main Scenario Steps: 1. CA logs in and is redirected to their homepage 2. **(R#02)** CA selects the option to request system change 3. They describe the current system problems they are facing along with any suggestions as to what can be done to make a better experience 4. **(R#15)** MT receives this request and evaluates it on whether the changes requested are reasonable and if to be implemented.	R#02, R#15
Post-Condition: A confirmation that the report was sent is shown to the user	
Extensions: No Alternative.	
Non-Functional Requirements: Usability, Reliability and Availability, Maintainability	NFR#01, NFR#02, NFR#06
Frequency of Occurrence: Whenever the client wants to request a system upgrade	
Miscellaneous Comments: N/A	

1.12 Risk Analysis and Prototype-I

A few risks in the development of the system were recognized during this phase:

1. Passwords are not encrypted. This could cause data leaks and unauthorized logins in some cases. Addressing this involves that the final release of the system must include an encryption algorithm to ensure the security of the customers.
2. Due to time constraints, the several use cases described above might be difficult to implement. Addressing this involves dedicating more time to implementing the system within the time frame specified.

Prototype-I addressed the issues above as was described. This way, it was safe to continue the implementation of the system and move on to the next phase: the design phase.

Part II: System Design

2.1 Introduction

In this phase, one class diagram overall, one sequence diagram for each use case (18 total), and one activity diagram for each use case (18 total) were designed. The class diagram shows all the objects in the system and how they are related/interact with one another. To the best of our ability, we reduced coupling and maximized cohesion.

2.2 Class Diagram

As per instructions, the class diagram contains all the classes on one page. The diagram is hard to read due to that, but it is clear to zoom in and read all the details. Every class, attribute, and method have been included, and it covers the whole system.

THIS PAGE IS LEFT BLANK FOR THE Class Diagram

2.3 Activity/Sequence Diagrams

For the sequence/activity diagrams, they were made to be compliments of each other, referencing other use cases for simplicity in some cases. In this document, instead of each type of diagram having its own section, we organized it in a way that, for each use case, the sequence and activity diagram of that use case are given together. This makes it easier to keep track and compare.

UC#1 SEQUENCE

UC#1 ACTIVITY

UC#2 SEQUENCE

UC#2 ACTIVITY

UC#3 SEQUENCE

UC#3 ACTIVITY

UC#4 SEQUENCE

UC#4 ACTIVITY

UC#5 SEQUENCE

UC#5 ACTIVITY

UC#6 SEQUENCE

UC#6 ACTIVITY

UC#7 SEQUENCE

UC#7 ACTIVITY

UC#8 SEQUENCE

UC#8 ACTIVITY

UC#9 SEQUENCE

UC#9 ACTIVITY

UC10 SEQUENCE

UC#10 ACTIVITY

UC#11 SEQUENCE

UC#11 ACTIVITY

UC#12 SEQUENCE

UC#12 ACTIVITY

UC#13 SEQUENCE

UC#13 ACTIVITY

UC#14 SEQUENCE

UC#14 ACTIVITY

UC#15 SEQUENCE

UC#15 ACTIVITY

UC#16 SEQUENCE

UC#16 ACTIVITY

UC#17 SEQUENCE

UC#17 ACTIVITY

UC#18 SEQUENCE

UC#18 ACTIVITY

2.4 Risk Analysis and Prototype-II

A few risks in the development of the system were recognized during this phase:

1. Due to the size of the class diagram, implementation of all these classes and rigorously testing them will take more time than was expected.

2. Graphical pages were all designed to have similar constraints and limitations. Therefore, since it was statically designed, the graphical pages are set for a screen resolution of 1920x1080 pixels. While this is likely to fit most screens, it still leads to the graphics to be represented sub-optimally on screen resolutions that don't match the 1920x1080 restriction.

Prototype-II addressed the issues above as follows:

1. Task division was reorganized to spread the work out more and ensure that the implementation and testing were given enough time and effort to include everything that was set as part of the design.

2. Future implementations of the system should include a dynamic display to tailor to almost all (if not all) screen sizes. However, as a short-term solution, the implementation can be made to be fully compatible with all screen resolutions/sizes to allow anyone to use the system, even if the graphics are to be not filling the whole screen.

Part III: System Implementation

3.1 Introduction

As previously described, the system is implemented in the Java programming language using best practices of object-oriented programming. The system has various classes, each with their own attributes and methods. A detailed documentation is provided in the supplementary submission to this report. This documentation is made by the JavaDocs standard of documenting code, and it contains all the methods and classes details. Moreover, said documentation can be found along with this submission.

To run the system, a readMe.txt file was also submitted to point out a few special running instructions/details about the system. Please read the file before running the program.

An important note: as per instructions, the implementation only covers the use cases for the customer. For example, admins should be able to view meeting requests and respond to them, but that is not applicable through the current version of the system. We were told that this is fine for the current version. The system can be further extended to cover the remaining actors in the future.

3.2 Code Documentation

In addition to the supplementary JavaDocs documentation to provide the details, here is a short overview of the classes (no details are provided here, JavaDocs contains the details).

1. **BankAutomated**: The logic/authentication unit of the system. Every use case that requires logic (such as transferring money) or authentication (such as logging in or registering) uses BankAutomated and its methods to validate everything.
2. **People**: The parent class for all the people objects in the system: AD, CA, CSR, and MT.
3. **AD**: The object to represent an admin in the system.
4. **CA**: The object to represent a customer with an account in the system.
5. **CSR**: The object to represent a customer service representative in the system.
6. **MT**: The object to represent a maintenance team member in the system.
7. **Report**: The object to represent a suspicious activity report that a customer made.
8. **Request**: The object to represent a request that a customer made, can be one of three forms.
9. **Transaction**: The object to represent a transaction that a customer made.
10. **BankSystem**: The main class of the system, it initializes a BankAutomated and a LoginPage object.
11. **LoginPage**: The graphical implementation of the login page for the user to sign into their account.
12. **RegisterPage**: The graphical implementation of the register page for the user to sign up for an account.
13. **ForgotPage**: The graphical implementation of the forgot password page for the user to request a reset.
14. **HomePage**: The graphical implementation of the user's home page, unique for each user.
15. **BankTransferPage**: The graphical implementation of the bank transfer page for the user to transfer.
16. **ETransferPage**: The graphical implementation of the e-transfer page for the user to make a transfer.
17. **TransferFundsPage**: The graphical implementation of a page that allows the user to transfer funds between their chequing and savings accounts.
18. **ChequingHistoryPage**: The graphical implementation of the chequing account transaction history page to display the 6 most recent transactions the user made with their chequing account.
19. **SavingsHistoryPage**: The graphical implementation of the savings account transaction history page to display the 6 most recent transactions the user made with their savings account.
20. **NotificationSettingPage**: The graphical implementation of the notification settings page for the user to set their notification preferences.

21. **PrivacySettingPage**: The graphical implementation of the privacy settings page for the user to set their privacy preferences or change their password.
22. **ProfileSettingPage**: The graphical implementation of the profile settings page for the user to update their profile (e.g., change their email/address).
23. **RequestsPage**: The graphical implementation of the make request page, where the user can select what type of request to make and specify details about said request.
24. **MakeReportPage**: The graphical implementation of the make report page, where the user can describe any suspicious activity on their account and send the details to an admin.
25. **FindUsPage**: The graphical implementation of the Find Us page, where the user can select a pinned location and find the address of the bank branch at that pin.

3.3 Risk Analysis and Prototype-III

A few risks in the development of the system were recognized during this phase:

1. The full implementation of the system was bigger than initially planned (not necessarily a bad thing, as implementation finished on time and covers everything described in the previous phases). However, this entails that highly rigorous testing of the system is made to avoid/address any small issues/bugs that go unnoticed.

2. Due to the size of the system, detailed documentation ended up being around 100 pages, which might be difficult to read through at once.

3. Since we do not have a database of bank users around the world, it is not possible to validate bank account numbers for bank transfers or email addresses for e-transfers. Therefore, the system currently assumes that any 5-digit number is a valid bank account number and any valid email address is valid for e-transfers.

Prototype-III addressed the issues above as follows:

1. The unit testing of the system was made to include every function that was used. In addition to that, a member of the team was set to test the system (specifically graphical controls) rigorously by creating various accounts and making various types of transactions. Several bugs were uncovered and handled.

2. The 100 paged-documentation (JavaDocs) is submitted separately (i.e. not part of this report), and less detailed, more focused documentation to provide instructions for running of the system is provided in the readMe.txt file along with the brief description in section 3.2. The JavaDocs-style documentation will still be provided to allow people to read through the details of the system, with quick links to allow faster examining of the documentation.

3. The final implementation of the system should be put against a database of bank users to ensure that only valid account numbers and registered email addresses are input by the user when making transfers.

Part IV: System Testing

4.1 Introduction

To ensure that the system meets the rigorous requirements and specifications set by the banking company, we implemented a comprehensive unit testing system as part of our software development process. This system is designed to thoroughly test individual components and modules of the system to identify any defects or errors and ensure that the system functions as intended. Our unit testing system is built using a robust and widely-used JUNIT testing framework that provides a suite of tools for writing, executing, and reporting on tests. The framework also enables continuous integration and delivery, allowing tests to be run automatically each time a new code is committed to the system. Our unit testing system covers all the critical areas of the online banking system, including login, register, transactions, reports, requests, changing password. Each module is tested both in isolation and as part of the integrated system to ensure that all the components work together seamlessly and reliably.

This unit testing can be found fully-implemented in the TestCase.java file in the src file within the supplementary zip file submitted along with this report. The JUNIT dependency is required to run said file.

4.2 Unit Testing Documentation

TestCase #0: saveloadTest()
Test Case Name: Saving and Loading test
Objective: To ensure that the main functionality of allowing customers to login into their accounts with all of their information intact, as per the functional requirements.
Pre-Condition(s): ● The online banking system is up and running ● Several customer objects are already given unique information for the registration process. ● None of the testing objects will be actually saved to People.ser
Test Steps: ● One customer object is created ● Logout (save to file) ● Reinitiate bank automated (load from file) ● Check for success/fail ● Create 1,000,000 customer objects ● Logout (save to file) ● Reinitiate bank automated (load from file) ● Check for success/fail
Expected Result: The result should be exactly one for the first test run, then 1,000,000 for the second run.

TestCase #1: stressTest()
Test Case Name: Stress Test
Objective: To comply as closely as possible with our non-function requirements for performance and reliability, by making sure we can compute millions of customers in a reasonable amount of time. Testing caching and multithreaded streams.
Pre-Condition(s): ● The online banking system is up and running ● The customer is given a unique email and password
Test Steps: ● One million dummy accounts are registered into the banking system ● Check the time of completion ● A random account is logged in ● Check the time of completion
Expected Result: The one million accounts should be created in about half a second, with each login taking about constant time to complete.

TestCase #2: testRegister()
Test Case Name: Register Testing
Objective: To make sure our function requirement for customers being able to register an account with our online banking system works. We test if the system is able to check for wrong inputs.
Pre-Condition(s): ● The online banking system is up and running ● The customer is given a unique email and password to test for fail/success
Test Steps: ● A customer with correct inputs is registered ● Check for fail/success ● A customer with incorrect inputs is registered ● Check for fail/success ● A customer with incorrect inputs is registered ● Check for fail/success
Expected Result: It should return correct for all cases and successfully create accounts that have correct inputs.

TestCase #3: testLogin()
Test Case Name: Login Testing
Objective: To make sure our function requirement for customers being able to login to their account in our banking system. Checks for correct email and password inputs and matches their input with an account object in our system.
Pre-Condition(s): ● The online banking system is up and running ● Each customer account has already been registered
Test Steps: ● A customer with correct inputs is logged in ● Check for fail/success ● A customer with incorrect inputs is logged in ● Check for fail/success ● A customer with incorrect inputs is logged in ● Check for fail/success
Expected Result: It should return correct for all cases and successfully return a customer object for each successful register.

TestCase #4: testTransferFunds()
Test Case Name: Transfer Funds testing
Objective: To make sure our function requirement for customers being able to transfer funds between their 2 accounts (checkings and savings). Check if the amount is valid and the customer picks valid accounts.
Pre-Condition(s): ● The online banking system is up and running ● Each customer account has already been registered and given a certain amount of cash
Test Steps: ● Transfer between accounts ● Check for success/fail ● Transfer between accounts with invalid inputs ● Check for success/fail
Expected Result: Money should be successfully transferred between accounts with valid transaction reports in their respective account histories.

TestCase #5: testETransfer()
Test Case Name: Bank Transfer Testing
Objective: To make sure our function requirement for customers being able to transfer funds within the bank. Check if the amount is valid and the customer picks a valid outside bank number and outside account.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered, with a certain amount of cash ● The receiver account is already registered, with a certain amount of cash
Test Steps: ● Transfer between accounts ● Check for success/fail ● Transfer between accounts with invalid inputs ● Check for success/fail
Expected Result: Money should be successfully transferred between accounts with valid transaction reports in their respective account histories.

TestCase #6: testBankTransfer()
Test Case Name: Bank Transfer Testing
Objective: To make sure our function requirement for customers being able to transfer funds outside of the bank, with amounts from (checkings and savings). Check if the amount is valid and the customer picks a valid outside bank number and outside account.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered, with our bank number and a certain amount of cash ● The receiver account is created with a separate bank number and a certain amount of cash
Test Steps: ● Transfer between accounts ● Check for success/fail ● Transfer between accounts with invalid inputs ● Check for success/fail
Expected Result: Money should be successfully transferred between accounts with valid transaction reports in their respective account histories.

TestCase #7: testCardExpiry()
Test Case Name: Card Expiry Testing
Objective: To make sure no expired cards or invalid expiry dates are able to pass through the register.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered, with a card.
Test Steps: ● Enter a not expired card ● Check for success/fail ● Enter an expired card ● Check for success/fail
Expected Result: It should be successful if the entered expiry date is greater than the current date.

TestCase #8: testBankNumber()
Test Case Name: Bank Number Testing
Objective: To make sure our function requirement for customers being able to transfer funds outside of the bank, with amounts from (checkings and savings). Check if the given bank number is correct.
Pre-Condition(s): ● The online banking system is up and running ● The receiver account is created with a separate bank number.
Test Steps: ● Check if the bank number is valid and correct to our bank transfer function.
Expected Result: It should be successful if the bank number entered has only digits.

TestCase #9: testCVV()
Test Case Name: Card CVV Testing
Objective: To make sure no invalid CVVs are able to pass through the register.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered, with a card.
Test Steps: ● Enter a valid CVV ● Check for success/fail ● Enter an invalid CVV ● Check for success/fail
Expected Result: It should be successful if the entered CVV is of length 3 or 4 and only numeric.

TestCase #10: testPassword()
Test Case Name: Bank Password Testing
Objective: To make sure no invalid passwords are able to pass through the register.
Pre-Condition(s): ● The online banking system is up and running ● The receiver account is created with a unique password
Test Steps: ● Check if entered password is valid ● Return success/fail
Expected Result: It should be as long as it has upper and lower cases, numbers and special characters and matches the length requirement.

TestCase #11: testDOB()
Test Case Name: Date of birth testing
Objective: To make sure no invalid birthdays are able to pass through the register. Check for year of birth, nobody under 18 and has entered a valid day and month.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered, with a date of birth entered.
Test Steps: ● Check if entered date of birth is valid ● Return success/fail
Expected Result: It should be successful if the day, month and year line up and the age of the customer is at least 18 years of age.

TestCase #12: testGender()
Test Case Name: Customer Gender Testing
Objective: Just to make sure we can set the genders of individual accounts.
Pre-Condition(s): ● The online banking system is up and running ● The receiver account is created with a random gender
Test Steps: ● Change the gender of the given account ● Check for success/fail
Expected Result: It should be successful as long as the setter method in CA is working.

TestCase #13: testAddress()
Test Case Name: Customer Address Testing
Objective: Just to make sure we can set the addresses of individual accounts.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered, with an address added.
Test Steps: ● Change the address of the given account ● Check for success/fail
Expected Result: It should be successful as long as the setter method in CA is working.

TestCase #14: testCardNum()
Test Case Name: Card Number testing
Objective: To make sure no invalid card numbers are able to pass through the register. Check if it passes the Luhn algorithm, is unique (not already registered to another account) and is of valid length.
Pre-Condition(s): ● The online banking system is up and running ● The receiver account is created.
Test Steps: ● Valid Card number is entered ● Check for success/fail ● Invalid Card number is entered ● Check for success/fail
Expected Result: It should be successful if it passes all of the requirements in BankAutomated, that is it passes the Luhn algorithm, of valid length and has not been used with another account.

TestCase #15: testChequing()
Test Case Name: Customer Chequing Testing
Objective: Just to make sure we can set the chequing account amounts of individual accounts.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered with a randomly generated chequing amount.
Test Steps: ● Change the chequing account amount of given account ● Check if success/fail
Expected Result: It should be as long as the setter method in CA is working.

TestCase #16: testSavings()
Test Case Name: Customer Savings Testing
Objective: Just to make sure we can set the savings account amounts of individual accounts.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered with a randomly generated savings amount.
Test Steps: ● Change the savings account amount of given account ● Check if success/fail
Expected Result: It should be as long as the setter method in CA is working.

TestCase #17: testChequingHistory()
Test Case Name: Customer Chequing History Testing
Objective: To make sure customers are given an accurate representation of their chequing history containing their transactions asper the functional requirements.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered with a randomly generated chequing amount.
Test Steps: ● Add in a new transaction object to the chequing history ● Check if success/fail ● Check if correctly stored in customer chequing history ● Check if sender is correct ● Check if receiver is correct ● Check if amount is correct
Expected Result: The expected result should be, whatever was entered during the construction of the Transaction object is correct and is successfully added into the checking history of the customer.

TestCase #18: testSavingsHistory()
Test Case Name: Customer Savings History Testing
Objective: To make sure customers are given an accurate representation of their savings history containing their transactions asper the functional requirements.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered with a randomly generated saving amount.
Test Steps: ● Add in a new transaction object to the savings history ● Check if success/fail ● Check if successfully added into customer savings history ● Check if sender is correct ● Check if receiver is correct ● Check if amount is correct
Expected Result: The expected result should be whatever was in the transaction object as it was initialized (sender, receiver and amount).

TestCase #19: testSetterGetter()
Test Case Name: Setter/Getter Testing
Objective: Last check on setter/getters in CA
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered with randomly generated details
Test Steps: ● Change everything with new details ● Get the details ● Check to see if everything lines up
Expected Result: It should be as long as the setter method in CA is working.

TestCase #20: testRequest()
Test Case Name: Request Testing
Objective: Test to see if customers are able to send reports and the reports histories are added to their request history. Also check if requests are properly handled(received) by the admin.
Pre-Condition(s): ● The online banking system is up and running ● The sender account is already registered ● Admin account already setup
Test Steps: ● Customer sends report to admin ● Check if admin received it
Expected Result: It should be successful if a report object is successfully created and sent to an admin.

4.3 Risk Analysis and Prototype-IV

During the testing phase, several rigorous tests were made as was described in the documentation above. This testing uncovered various bugs in the system, all of which were handled and fixed with little-to-no problems.

Prototype-IV (submitted as part of the zip file, in the src folder) contains all the system specifications and more, with various graphical implementations to facilitate the use of online banking for all users of all groups (as was intended as part of the domain).

Several risks mentioned previously still persist in this prototype, but can be handled as further prototypes and the final implementation are made available (e.g., add encryption algorithms for passwords, check bank account numbers and emails against a database of bank users, etc.).

Part V: Conclusion

5.1 Reflection

The construction of the automated banking project was a difficult and time-consuming endeavor that nevertheless gave the group invaluable insight into the lifecycle of a software solution. Overall, there were aspects of the project which were straightforward, while others proved more difficult than anticipated. On the more manageable side, the requirements analysis proved simple as we were able to identify the core functionality of our system and turn those into functional and non-functional requirements. Additionally, many of the diagrams, although time consuming, were straightforward as we could create our diagrams directly based on the requirements. We found that our steps were able to flow seamlessly into one another and each part of the process could build off of what we had done previously. This showed us why the SDLC is so important and widely used.

On the other hand, there were portions of the process which required considerably more effort than the others. Most prominent was the actual implementation which took the most man hours to complete. This step was where we encountered most of our issues, and where our initial guidelines were challenged by the realities of implementation and development. This in turn created additional difficulties when a good bulk of our activity diagrams had to be revised in order to reflect the reality of the implementation as opposed to the initial proposal. Test cases and other debugging also proved to be time consuming, as nearly every aspect of the program had to be meticulously stress-tested and provided with the necessary edge-cases. Despite this, we have managed to implement and even in some cases expand our vision of the banking web application, and have cultivated numerous skills that could be carried across to a professional environment.

5.2 Task Division Table

Mo Nakouzi	● Phase 1 ○ Gantt Chart in excel ○ Table of Assumptions ○ Part of STD ○ Use Case Table ○ Organization and final submission ● Phase 2 ○ Part of Class Diagram ○ Sequence Diagrams ○ Organization and final submission ● Phase 3 ○ Implementation of all graphical pages ○ Integrating BA methods into the graphical implementation ● Phase 4 ○ Ensuring test cases were in line with functions ○ Bug fixes if any test cases failed ○ Code documentation into Javadocs
Kai Adams	● Phase 1 ○ Gantt Chart in excel ○ Glossary

	○ Part of STD ● Phase 2 ○ Activity Diagrams #8-11 ● Phase 3 ○ Created graphical classes skeleton code ○ Implementation of the settings functionality ● Phase 4 ○ Test cases for some setters and getters ○ Test the system by running it to find bugs ○ Sequence diagram fixes
Reynold Villasenor-Doan	● Phase 1 ○ Part of STD ○ Functional Requirements ● Phase 2 ○ Activity Diagrams #15-18 ● Phase 3 ○ Implementation of logic behind homepage ● Phase 4 ○ Test some functions in BA
Danny Guan	● Phase 1 ○ Gantt table ○ Non-Functional Requirements ○ Task division ● Phase 2 ○ Activity Diagrams #5-8 ○ Class Diagram ● Phase 3 ○ Login, Logout, Register, Change Password ○ Implementation of logic behind transactions/requests/reports history ○ BankAutomated methods and CA ○ Save/Load customer details ○ Documentation and bug fixes ● Phase 4 ○ Stress test, Login/Register test ○ Unit Testing Documentation ○ Saving/Loading to file test
Shivesh Mahato	● Phase 1 ○ Functional Requirements ○ Total Duration ● Phase 2

	○ Activity Diagrams #12-14 ○ Sequence Diagram Descriptions #1-7 ● Phase 3 ○ Transfer Funds functionality ○ Bank Transfer functionality ○ E transfer functionality ○ Some getter/setter functions ● Phase 4 ○ Tests for e transfer, bank transfer, funds transfer ○ Test cases for getters and setters
Mykhaylo Batashov	● Phase 1 ○ Duration and SDLC methodology ○ Functional Requirements ● Phase 2 ○ Activity Diagrams # 1-4 ○ Goals and subgoals ● Phase 3 ○ FindUs functionality ○ meetingRequest functionality ○ Report functionality ● Phase 4 ○ FindUs implementation

5.3 Submission and Source Code

All the code can be found in this github repository. Any bug fixes/updates can be found there. Along with this report, the submission will include the readME.txt file, the JAR file, and the code and testing.