Software Requirement Specification (SRS)

Project name: ASE Bank Application

Date: 07/05/24

Version: 2

By: Michael Edwards

Revision History:

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Author | Version Description | Date Completed |
| 1 | Mitchell Seng | First iteration | 12/04/24 |
| 2 | Michael Edwards | Second Iteration | 07/05/024 |

Review History:

|  |  |
| --- | --- |
| Approving Party | Michael Edwards, Mitchell Seng, James Gibson |
| Version Approved | 2 |
| Signature Date | 07/05/24 |

Approval History:

|  |  |
| --- | --- |
| Reviewer | Mitchell Seng |
| Version Reviewed | 2 |
| Signature | Mitchell Seng |
| Date | 0705/24 |

Table of contents

1. Introduction
   1. Product Scope
   2. Product Value
   3. Intended Audience
   4. Intended Use
   5. General Description
2. Functional Requirements
3. External Interface Requirements
   1. User Interface Requirements
   2. Hardware Interface Requirements
   3. Software Interface Requirements
   4. Communication Interface Requirements
4. Non-Functional Requirements
   1. Security
   2. Capacity
   3. Compatibility
   4. Scalability
   5. Maintainability
   6. Usability
   7. Other Non-Functional Requirements

Introduction   
Describe the purpose of the document.

1.1 Product Scope

* The aim of this project is to create an application for a bank. The application will offer a secure place for users to do their banking. Using two factor authentication and data encryption to keep a user's information safe. Another benefit of the project is the ability to contact support from the application directly.

1.2 Product Value

* The project aims to offer a secure place for users to do their banking, this security should add to the value they hold in the product. Another thing that will add value to the product is the ability to contact a help desk through the application. By not having to exit the banking application to get support the user gets a more cohesive experience.

1.3 Intended Audience

* The intended audience of this product is the public, users may vary in age and culture. The product should be able to be used by anyone to do their banking.

1.4 Intended Use

* When the user first opens the application, they will be prompted with a login page. After they have logged into their account they will be given various options, including edit account details, close or open an account, move money between accounts, move money into or out of the account, view a transaction history, set up recurring bills, and manage services such as check books and credit cards.

1.5 General Description

* The software will offer users a range of banking functions. Starting with account management users will be able to set up two types of banking accounts, personal or business. They will be able edit the account details including the account type, and any user information that may need updating. Finally, they will be able to close any account they have opened. Next users will be able to make financial transactions such as depositing money into an account, withdrawing money from an account, and transferring money between different accounts within the same bank. All these transactions will be able to be viewed in an account’s transaction history. For account services a user will be able to view account information, including its current balance, recent transactions, and the account type. Request and manage check books, debit, and credit cards. And set up recurring payments using direct debits. All these functions will comply with relevant and current data laws and regulations. This will be done with a robust two factor authentication system and appropriate levels of data encryption for any sensitive information. Overall, the software will provide a user friendly and comprehensive banking experience.

Functional Requirements

* 1. If/Then Behaviours:

The If/Then behaviours describe the expected if statements required for the program to run.

* If new account information is unique, then create new account.
* If username and password are correct, then begin two factor authentication.
* If two factor is valid, then allow access to account.
* If account details are correct, then allow money deposit.
* If balance has enough funds, then allow money withdrawal.
* If account details are valid, then allow transfer between accounts.
* If account details are valid, allow new credit/ debit card and/or check book request.
* If account details are valid, allow new recurring payment.
  1. Data Handling Logic:

How data moves between tables/ databases in the program.

* Moving of money between accounts during transactions.
* Depositing money into account on request & calculating new total cash funds.
* Calculating if money to withdraw < total amount of money and then withdrawing money form account on request.
* Storing of account details in relevant tables on creation of new accounts (i.e. account name, total funds, credit/ debit cards, etc.)
* Deleting relevant data on deleting an account.
* Checking two factor authenticator is matching to the account trying to be accessed.
  1. System Workflows:

How the expected system workflow should be, essentially how the user will move from one feature to another.

* How the system flows? From one section to another.
* Login screen 🡪 Create new account.
* Login screen 🡪 Two factor authentication.
* Two factor authentication 🡪 Main menu.
* Main menu 🡪 Account information.
* Account information 🡪 Edit account details.
* Main menu 🡪 Account balance.
* Account balance 🡪 Transaction history.
* Account balance 🡪 Cash deposit.
* Account balance 🡪 Cash withdraw.
* Account balance 🡪 Cash transfer menu.
* Main menu 🡪 Manage credit/ debit card menu.
* Manage credit/ debit card menu 🡪 Open new card.
* Manage credit/ debit card menu 🡪 Cancel card.
* Main menu 🡪 Close account
  1. Transaction Handling:

Logic of how transactions will move money between accounts.

* Deposit:
  + Get account balance.
  + Get deposited cash.
  + Combine current balance + deposited cash.
* Withdraw:
  + Get account balance.
  + Get withdraw amount.
  + If withdraw amount < current balance:
    - Then, subtract current balance – withdraw amount.
  + Else:
    - Then, return error.
* Transaction:
  + Get account1 balance (Withdrawer)
  + Get account2 balance (Receiver)
  + Get transaction amount.
  + If withdraw amount < account1 balance:
    - Then, subtract account1 balance – transaction amount.
    - Combine account2 balance + transaction amount.
  + Else:
    - Then, return error.
  1. Administrative Functions:

Main functions to be used by the program.

* Deposit\_Cash(self, amount)
* Withdraw\_Cash(self, amount)
* Transfer\_Cash(self, amount, second\_account)
* Create\_Account(self, username, password, amount)
* Close\_Account(self)
* Login(entered\_username, entered\_password)
* Logout(self)
* Display\_Details(self)
* Edit\_Details(self)
* Create\_Two\_Factor(self)
* Verify\_Two\_Factor(self)
  1. Regulatory and Compliance Needs:

How the system will abide by law and regulation requirements.

* All data stored is encrypted to abided by data protection acts.
* All relevant data will be deleted when account is closed to preserve data protection.
* All data will be updated correctly when a user changes their details to have data consistency.
* Prevent unauthorised access to data to follow data protection act.
  1. Performance Requirements:
* Ensuring that the program has a high level of performance on all operating systems (i.e. Windows, Mac, Linux, etc.)

External Interface Requirement

3.1 User interface requirements Describe the logic behind the interactions between users and the software (screen layouts, style guides, etc).   
3.2 Hardware interface requirements List the supported devices the software is intended to run on, the network requirements, and the communication protocols to be used.   
3.3 Software interface requirements Include the connections between your product and other software components, including frontend/backend framework, libraries, etc.   
3.4 Communication interface requirements List any requirements for the communication programs your product will use, like emails or embedded forms.

Non-Functional Requirements

4.1 Security Include any privacy and data protection regulations that should be adhered to.  
4.2 Capacity Describe the current and future storage needs of your software.   
4.3 Compatibility List the minimum hardware requirements for your software.   
4.4 Reliability Calculate what the critical failure time of your product would be under normal usage.   
4.5 Scalability Calculate the highest workloads under which your software will still perform as expected.   
4.6 Maintainability Describe how continuous integration should be used to deploy features and bug fixes quickly.   
4.7 Usability Describe how easy it should be for end-users to use your software.   
4.8 Other List any additional non-functional requirements.