|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project Documentation OP BANK**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | |
|  |  |  | |  |
| **IT Project**  **OP-Bank**  Project Documentation  Group Name: DreamTeam Date: 10 November, 2023 | | |
|  | |  |

Table of Contents

[1. Introduction 3](#_Toc150463253)

[2. Business Rules 3](#_Toc150463254)

[3. Functional and Non-functional Requirements 4](#_Toc150463255)

[4. System Modeling 8](#_Toc150463256)

[1.1 Use Case 8](#_Toc150463257)

[1.2 Domain Class Diagram 20](#_Toc150463258)

[1.3 Activity Diagrams 20](#_Toc150463259)

[5. User Interfaces 26](#_Toc150463260)

[6. Implementation & Testing 28](#_Toc150463261)

[7. Conclusion 37](#_Toc150463262)

[8. References 37](#_Toc150463263)

# Introduction

This project is aimed at developing a prototype of an intuitive, user-friendly online banking web application. Henceforth titled ‘OP-Bank’, the application is intended to meet the specific needs and preferences of older adults.

OP-Bank, therefore, will focus on a simple, text-focused user-centric design targeted at older people to help them manage their finances confidently and independently in the digital age. The prototype application will be accessible through an internet browser and have an integrated chatbot assistant to help guide and navigate users through the onboarding process and utilizing the application’s primary functions.

**Keywords**: Older People (OP), OP-Bank, online banking, prototype application.

# Business Rules

Following are the business rules identified for this project:

1. Provide a clean user interface and experience consisting of large, easy-to-read fonts, high contrast colours and minimal distraction.
2. Use standard encryption techniques for secure, easy authentication.
3. Allow users to view their account balance easily with minimal number of steps.
4. Provide an accurate, easy-to-read transaction history with the ability to view details for each transaction.
5. Provide the ability to transfer money from one account to another.
6. Notify users of any important banking activities in a notification centre.
7. Provide a chat bot to help navigate around the application and assist with basic user queries.
8. (Important) Joint Accounts: Allow for joint accounts, catering to older couples who wish to manage their finances together.
9. Transaction Limits: Implement daily transaction limits to prevent unauthorized or excessive spending for security.
10. Preferred Communication: Give users the option to choose their preferred mode of communication, such as emails, SMS, or paper statements.
11. Personalization: Enable users to customize their user interface based on their visual and cognitive preferences.
12. User-Centric Design: Prioritize the needs and preferences of older adults in all design and functionality decisions.
13. Accessibility: Ensure that the application adheres to the highest standards of accessibility, catering to the diverse needs of older adults.
14. Security: Implement the highest standards of encryption and security protocols to protect user data and financial transactions.
15. Chatbot Integration: The application should have an integrated chatbot to assist users in navigating the platform and answering queries.
16. Age Limitation: Only individuals aged 65 and above are eligible to register for the OP-Bank application.
17. Transaction History: Users should have access to their transaction history for a minimum of 5 years.
18. Data Retention: User data should be retained for a specific period as per regulatory requirements.
19. Feedback Mechanism: Users should be able to provide feedback on the application's usability and functionality.

**Assumption:**

* Not all of the above business rules have been implemented in the prototype due to time and resource constraints. However, they have been considered and listed here to reflect if the application were to be implemented for real-world use.

# Functional and Non-functional Requirements

**Functional Requirements**:

These requirements are based off the business rules, and are mentioned below:

1. User must be able to register and log in securely with PIN/biometric authentication.
2. User must be able to view their account balance on the app home screen.
3. User must be able to view their transaction history, and view details for a particular transaction.
4. User must be able to transfer funds from one account to another. They should be notified of success/failure operation.
5. Users must be notified of any major change to their account. e.g., fund transfer, change password etc.
6. Customer support- The web app should provide easy access to customer support, in case older people need help using it. (Chatbot)
7. User authentication - Provide instructions for user registration and login process clearly.
8. Search and filter past transactions by date. Provide transaction details including date and account name where the transaction occurred.
9. Enable users to add, edit, and delete bill payees.
10. Provide a feature to schedule one-time or recurring bill payments with reminders.
11. Search functionality that allows users to search for specific transactions, payees, or account details.
12. Account Overview: Users should have a clear view of their account balance and recent transactions on the home screen.
13. Transaction Details: Users should be able to click on individual transactions to view detailed information.
14. Fund Transfers: Users should be able to transfer funds between accounts, with clear notifications indicating the success or failure of the transaction.
15. Notifications: Users should receive notifications for significant account activities, such as fund transfers or password changes.
16. Bill Payments: Users should be able to schedule and manage bill payments.
17. Search Functionality: Users should be able to search for specific transactions, payees, or account details.
18. Profile Management: Users should be able to edit their profile, including contact information and preferences.
19. OP-Bank should provide a bill price function allowing customers to pay online. Users must be capable of adding, controlling, and agenda bill payments for utilities, leases, loans, and other charges.
20. The included chatbot ought to be available to assist customers 24/7. It must be able to answer common questions, guide users through various application features, and offer support for account-related issues.

**Non-Functional Requirements**:

Non-functional requirements ensure that the app can deliver the required functionality for older people:

1. The app should adhere to WCAG guidelines for it to be usable by people with visual and cognitive impairments.
2. Efficiency - The app should contain features such as large icons, simple, clear, and short text display which improves efficiency. Use of simple navigations in the app.
3. Errors - Less errors should be encountered by the user when using the app and they should be able to recover from the error easily. Besides, to increase the understanding of the error messages, app should contain less complex error messages.
4. Memorability - The app may include photos and illustrations to indicate or remind users in recalling the steps involved in the app usage.
5. Response Time: Ensure the application responds within 2 seconds for all user interactions to maintain a smooth experience.
6. Biometric Security: Offer biometric authentication options, facial recognition, to enhance security.
7. Disaster Recovery: Develop a robust disaster recovery plan to minimize downtime and data loss.
8. A/B Testing: Continuously conduct A/B testing to refine the UI and features based on user feedback.
9. Concierge Support: Provide personalized customer support with dedicated representatives for elderly users.
10. Cross-Browser Compatibility: Guarantee compatibility with major browsers and screen readers to accommodate various user preferences.
11. Regular Accessibility Audits: Perform regular accessibility audits to meet evolving standards and ensure a user-friendly experience.
12. Geographical Availability: Roll out the service in regions with a significant older adult population first to prioritize accessibility.
13. The net utility must have fast response times to ensure a smooth consumer experience. It must take care of concurrent user periods efficiently without massive performance degradation.
14. The software must be designed with scalability in mind to deal with destiny and increase in-person numbers and capabilities. Scalability ought to consist of adding new banking services and expanding server sources as wanted.

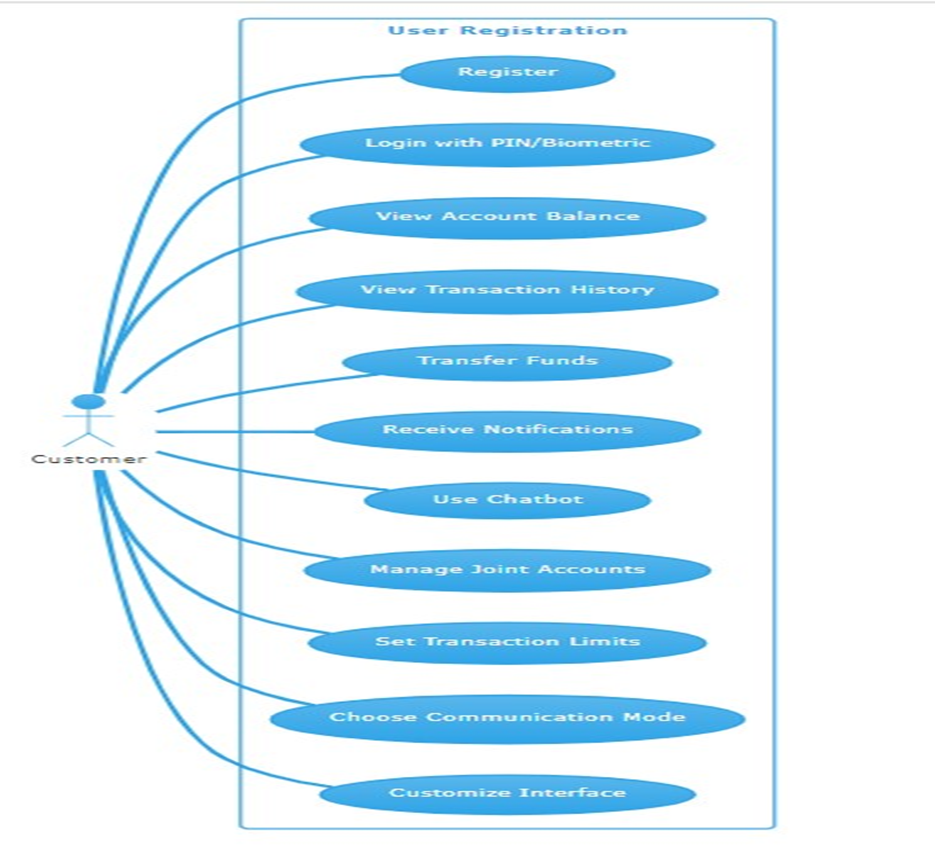
**Assumption:**

* Not all the above requirements have been implemented in the prototype due to time and resource constraints. However, they have been considered and listed here to reflect if the application were to be implemented for real-world use.

# System Modeling

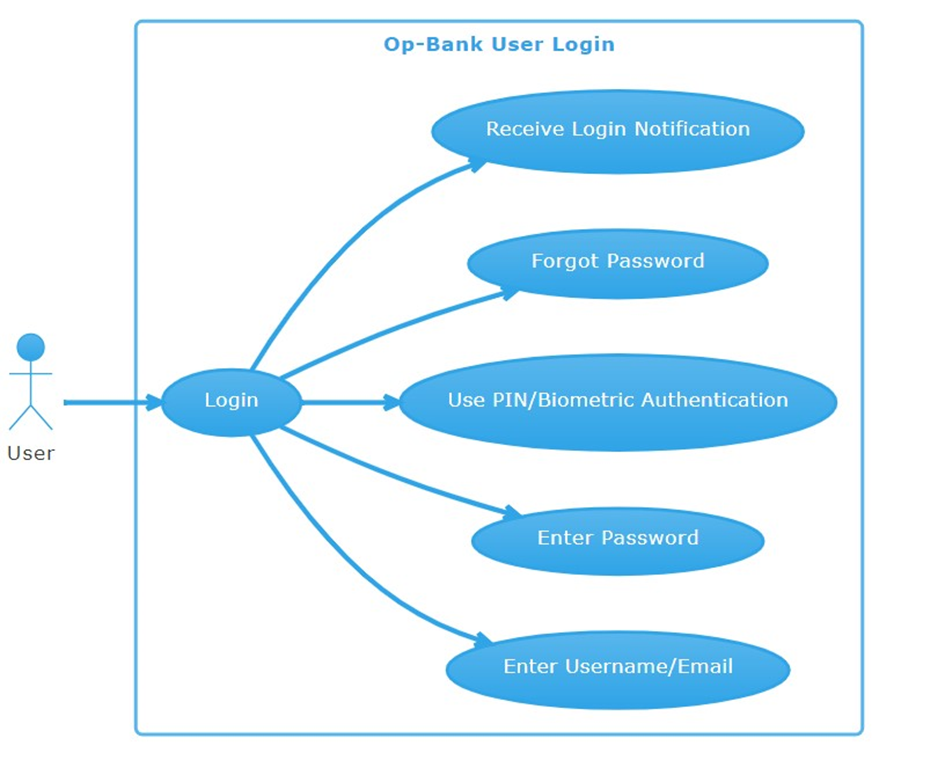
## Use Case

1. **User Registration**



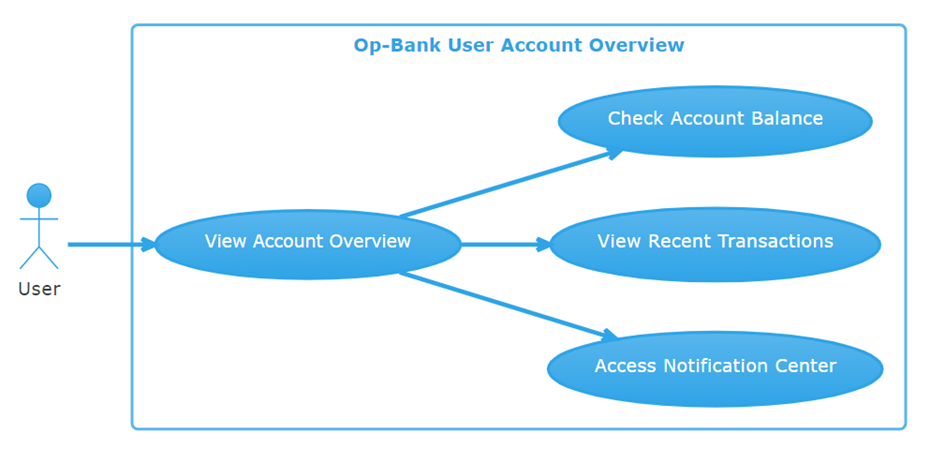
|  |  |  |
| --- | --- | --- |
| Use Case Name: | User Registration | |
| Scenario | A new user wants to register for the OP-Bank application | |
| Triggering Event: | User selects the "Register" option on the OP-Bank application homepage. | |
| Brief Description: | This use case describes the process by which a new user can register for the OP-Bank application, setting up their account and personal details. | |
| Actors: | User | |
| Stakeholders | - OP-Bank application developers  - OP-Bank application administrators  - Older adults (potential users) | |
| Preconditions: | - User has accessed the OP-Bank application.  - User is aged 60 and above (as per the age limitation for registration). | |
| Postconditions: | - User has successfully registered and has an active account in the OP-Bank application.  - User can now log in using their credentials. |  |
| Flow of Activities: | Actor | System |
| 1. Selects "Register" option. 2. Enters personal details, sets up PIN/biometric authentication. 3. Submits the registration form. | 1. Displays registration form. 2. Validates the provided details. 3. Creates a new user account and sends a confirmation notification. |
| Alternative Flows | 1. User decides to cancel the registration process. 2. System redirects the user back to the homepage. | |
| Exception Conditions: | 1. 1. User provides invalid or incomplete details.  - System prompts the user to correct the errors before proceeding.  2. User is below the age of 60.  - System informs the user that they are not eligible to register. | |

1. **User Login**



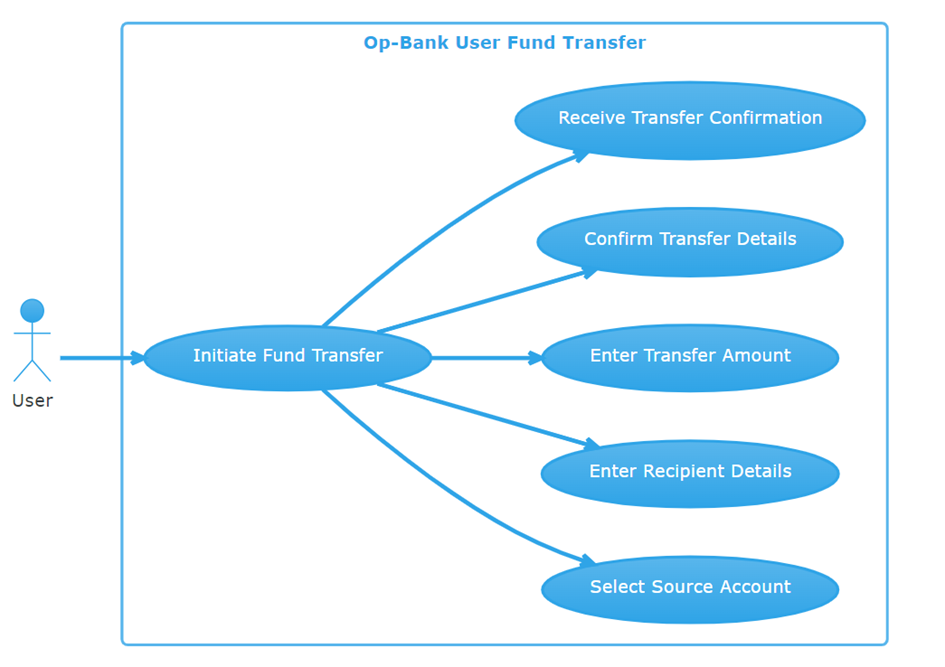
|  |  |  |
| --- | --- | --- |
| Use Case Name: | User Login | |
| Scenario | An existing user wants to log into the OP-Bank application. | |
| Triggering Event: | User selects the "Login" option on the OP-Bank application homepage. | |
| Brief Description: | This use case describes the process by which an existing user can log into the OP-Bank application using their credentials or PIN/biometric authentication. | |
| Actors: | User | |
| Stakeholders | OP-Bank application developers  - OP-Bank application administrators  - Registered users | |
| Preconditions: | - User has an active account in the OP-Bank application.  - User has either their username/email and password or PIN/biometric set up. | |
| Postconditions: | - User is successfully logged in and has access to their account dashboard. |  |
| Flow of Activities: | Actor | System |
| 1. Selects "Login" option. 2. Enters Username/Email. 3. Enters Password or uses PIN/Biometric Authentication. 4. Submits the login form. | 1. Displays login form. 2. Validates the provided username/email. 3. Validates the provided password or PIN/biometric. 4. Grants access to the user's account dashboard and sends a login notification. |
| Alternative Flows | 1. User selects "Forgot Password" option.  - System initiates the password recovery process. | |
| Exception Conditions: | 1. User provides incorrect username/email or password.  - System prompts the user to check their credentials and try again.  2. Multiple failed login attempts.  - System temporarily locks the account and informs the user. | |

1. **User Account Overview**



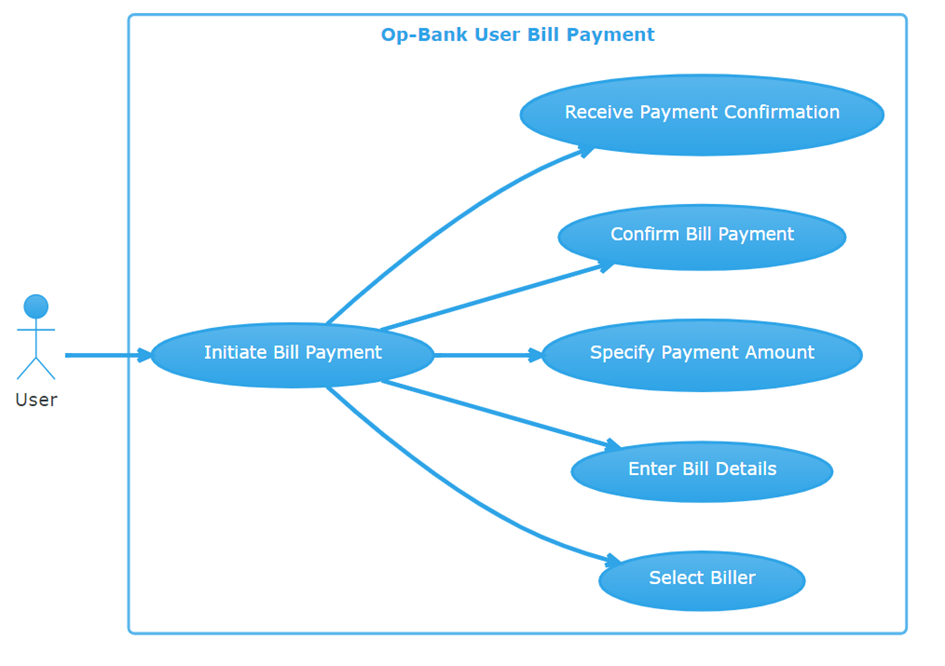
|  |  |  |
| --- | --- | --- |
| Use Case Name: | User Account Overview | |
| Scenario | A registered user wants to view their account overview, including their account balance and recent transactions. | |
| Triggering Event: | User logs into the OP-Bank application and accesses the account dashboard. | |
| Brief Description: | This use case describes the process by which a user can view their account overview, check their current account balance, view recent transactions, and access the notification center for any banking updates. | |
| Actors: | User | |
| Stakeholders | - OP-Bank application developers  - OP-Bank application administrators  - Registered users | |
| Preconditions: | - User is logged into the OP-Bank application.  - User has an active account with transaction history. | |
| Postconditions: | - User has viewed their account balance and recent transactions.  - User is informed about any important banking activities via the notification center. |  |
| Flow of Activities: | Actor | System |
| 1. Selects "Account Overview" option. 2. Views account balance. 3. Checks recent transactions. 4. Accesses the notification center. | 1. Displays account dashboard. 2. Displays the current account balance. 3. Lists the recent transactions with details. 4. Shows any important banking notifications. |
| Alternative Flows | 1. User wants to view transactions from a specific date range.  - System provides filtering options to view transactions based on selected dates. | |
| Exception Conditions: | 1. System fails to retrieve account details.  - System displays an error message and prompts the user to try again later. | |

**4. User Fund Transfer**



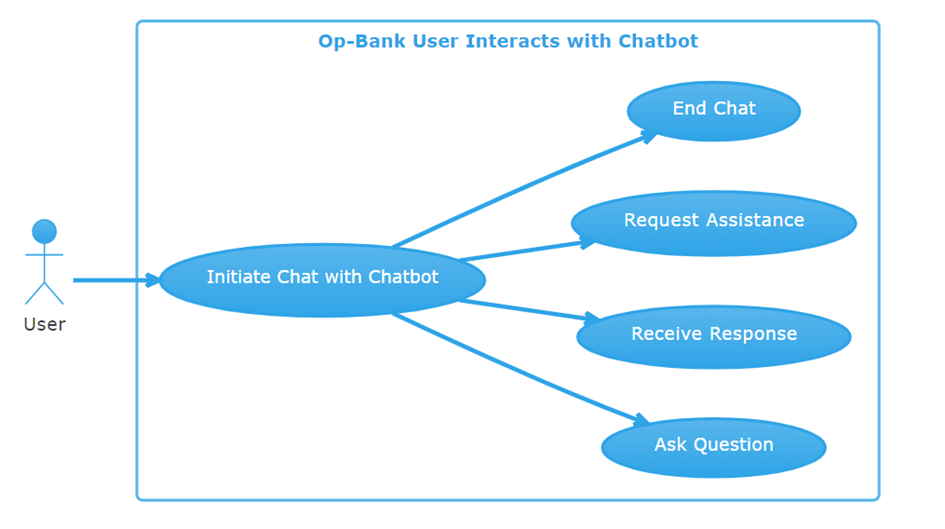
|  |  |  |
| --- | --- | --- |
| Use Case Name: | User Fund Transfer | |
| Scenario | A registered user wants to transfer funds from their account to another account. | |
| Triggering Event: | User selects the "Fund Transfer" option on the OP-Bank application dashboard. | |
| Brief Description: | This use case describes the process by which a user can initiate a fund transfer, select the source account, enter recipient details, specify the transfer amount, confirm the transfer details, and receive a confirmation of the transfer. | |
| Actors: | User | |
| Stakeholders | - OP-Bank application developers  - OP-Bank application administrators  - Registered users  - Recipients of the fund transfer | |
| Preconditions: | - Funds are successfully transferred from the user's account to the recipient's account.  - User receives a confirmation of the successful transfer. | |
| Postconditions: | - User has viewed their account balance and recent transactions.  - User is informed about any important banking activities via the notification center. |  |
| Flow of Activities: | Actor | System |
| 1. Selects "Fund Transfer" option.  2. Selects source account.  3. Enters recipient details (account number, bank, etc.).  4. Enters transfer amount.  5. Confirms transfer details.  6. Waits for confirmation. | 1. Displays fund transfer form.  2. Validates the selected account and its balance.  3. Validates the recipient details.  4. Checks if the entered amount is within the permissible limits and available balance.  5. Processes the fund transfer.  6. Sends a transfer confirmation notification to the user. |
| Alternative Flows | 1. User enters an amount exceeding their available balance.  - System prompts the user to enter a valid amount within the available balance.  2. User enters invalid recipient details.  - System prompts the user to check and correct the recipient details. | |
| Exception Conditions: | 1. System fails to process the fund transfer due to technical issues.  - System displays an error message and prompts the user to try again later.  2. Transfer amount exceeds the daily transaction limit.  - System informs the user about the exceeded limit and prompts to enter a lesser amount. | |

1. **User Bill Payment**



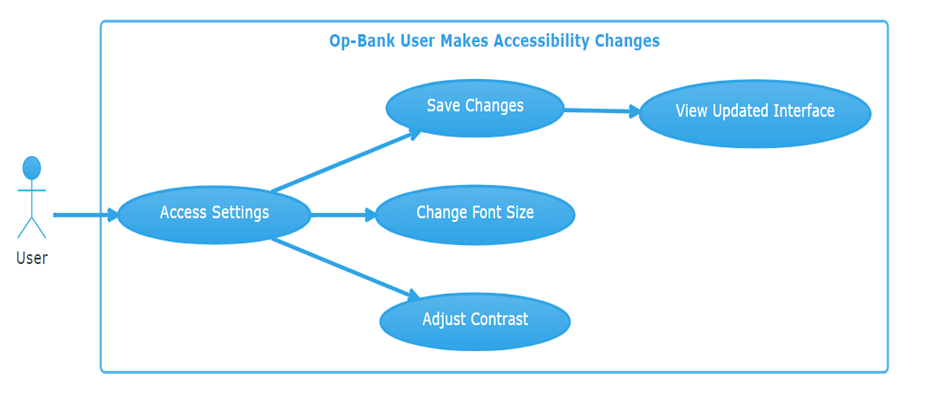
|  |  |  |
| --- | --- | --- |
| Use Case Name: | User Bill Payment | |
| Scenario | A registered user wants to pay a bill using the OP-Bank application. | |
| Triggering Event: | User selects the "Bill Payment" option on the OP-Bank application dashboard. | |
| Brief Description: | This use case describes the process by which a user can initiate a bill payment, select the biller, enter bill details, specify the payment amount, confirm the bill payment details, and receive a confirmation of the payment. | |
| Actors: | User | |
| Stakeholders | - OP-Bank application developers  - OP-Bank application administrators  - Registered users  - Billers (Utility companies, service providers, etc.) | |
| Preconditions: | - User is logged into the OP-Bank application.  - User has sufficient funds in their account for the bill payment. | |
| Postconditions: | - Bill is successfully paid.  - User receives a confirmation of the successful bill payment. |  |
| Flow of Activities: | Actor | System |
| 1. Selects "Bill Payment" option.  2. Selects biller (e.g., utility company, service provider).  3. Enters bill details (bill number, due date, etc.).  4. Specifies payment amount.  5. Confirms bill payment details.  6. Waits for payment confirmation. | 1. Displays bill payment form. 2. Validates the selected biller. 3. Validates the entered bill details. 4. Checks if the entered amount is within the permissible limits and available balance. 5. Processes the bill payment. 6. Sends a payment confirmation notification to the user. |
| Alternative Flows | 1. User enters an amount exceeding their available balance.  - System prompts the user to enter a valid amount within the available balance.  2. User enters invalid bill details.  - System prompts the user to check and correct the bill details. | |
| Exception Conditions: | 1. System fails to process the bill payment due to technical issues.  - System displays an error message and prompts the user to try again later.  2. Payment amount exceeds the daily transaction limit.  - System informs the user about the exceeded limit and prompts to enter a lesser amount. | |

1. **User interacts with Chatbot**



|  |  |  |
| --- | --- | --- |
| Use Case Name: | User interacts with Chatbot | |
| Scenario | A registered user wants to interact with the integrated chatbot for assistance or queries. | |
| Triggering Event: | User initiates a chat with the chatbot on the OP-Bank application. | |
| Brief Description: | This use case describes the process by which a user can initiate a chat with the chatbot, ask questions, receive responses, request specific assistance, and end the chat session. | |
| Actors: | User | |
| Stakeholders | - OP-Bank application developers  - OP-Bank application administrators  - Registered users  - Chatbot developers and trainers | |
| Preconditions: | - User is logged into the OP-Bank application.  - Chatbot is operational and available for interaction. | |
| Postconditions: | - User's queries or requests are addressed.  - User ends the chat session with the chatbot. |  |
| Flow of Activities: | Actor | System |
| 1. Initiates chat with chatbot.  2. Asks a question or states a request.  3. Continues interaction or asks further questions.  4. Decides to end the chat. . | 1. 1. Activates chatbot interface.      1. Chatbot processes the user's input. 2. Chatbot provides a relevant response or action.      1. Chatbot continues to provide responses or actions based on user's input.      1. Chatbot offers closing remarks and ends the chat session. |
| Alternative Flows | 1. User asks a question that the chatbot cannot answer.  - Chatbot informs the user that it doesn't have the answer and offers to redirect to customer support or provide related information.  2. User requests an action that the chatbot cannot perform.  - Chatbot informs the user of its limitations and suggests alternative solutions. | |
| Exception Conditions: | 1. Chatbot fails to understand the user's input multiple times.  - Chatbot suggests rephrasing the question or offers to redirect the user to live customer support.  2. Chatbot becomes unresponsive.  - System displays an error message and suggests the user try again later. | |

1. **User makes accessibility changes**



|  |  |  |
| --- | --- | --- |
| Use Case Name: | User makes accessibility changes | |
| Scenario | A registered user wants to adjust the accessibility settings of the OP-Bank application to enhance their user experience. | |
| Triggering Event: | User navigates to the settings or accessibility options on the OP-Bank application. | |
| Brief Description: | This use case describes the process by which a user can access the settings, change the font size, adjust the contrast, save these changes, and view the updated interface with the new accessibility settings. | |
| Actors: | User | |
| Stakeholders | - OP-Bank application developers  - OP-Bank application administrators  - Registered users | |
| Preconditions: | - User is logged into the OP-Bank application.  - Accessibility settings are available for customization. | |
| Postconditions: | - User's preferred accessibility settings are applied.  - User views the updated interface with the new settings. |  |
| Flow of Activities: | Actor | System |
| 1. Navigates to settings or accessibility options.  2. Chooses to change the font size. 2  3. Decides to adjust the contrast.  4. Saves the changes made.  5. Returns to the main interface. | 1. Displays available accessibility settings. 2. Provides options for font size adjustments. 3. Offers contrast adjustment settings. 4. Applies the new accessibility settings. 5. Displays the updated interface with the new settings applied. |
| Alternative Flows | 1. User asks a question that the chatbot cannot answer.  - Chatbot informs the user that it doesn't have the answer and offers to redirect to customer support or provide related information.  2. User requests an action that the chatbot cannot perform.  - Chatbot informs the user of its limitations and suggests alternative solutions. | |
| Exception Conditions: | 1. Chatbot fails to understand the user's input multiple times.  - Chatbot suggests rephrasing the question or offers to redirect the user to live customer support.  2. Chatbot becomes unresponsive.  - System displays an error message and suggests the user try again later. | |

## Domain Class Diagram

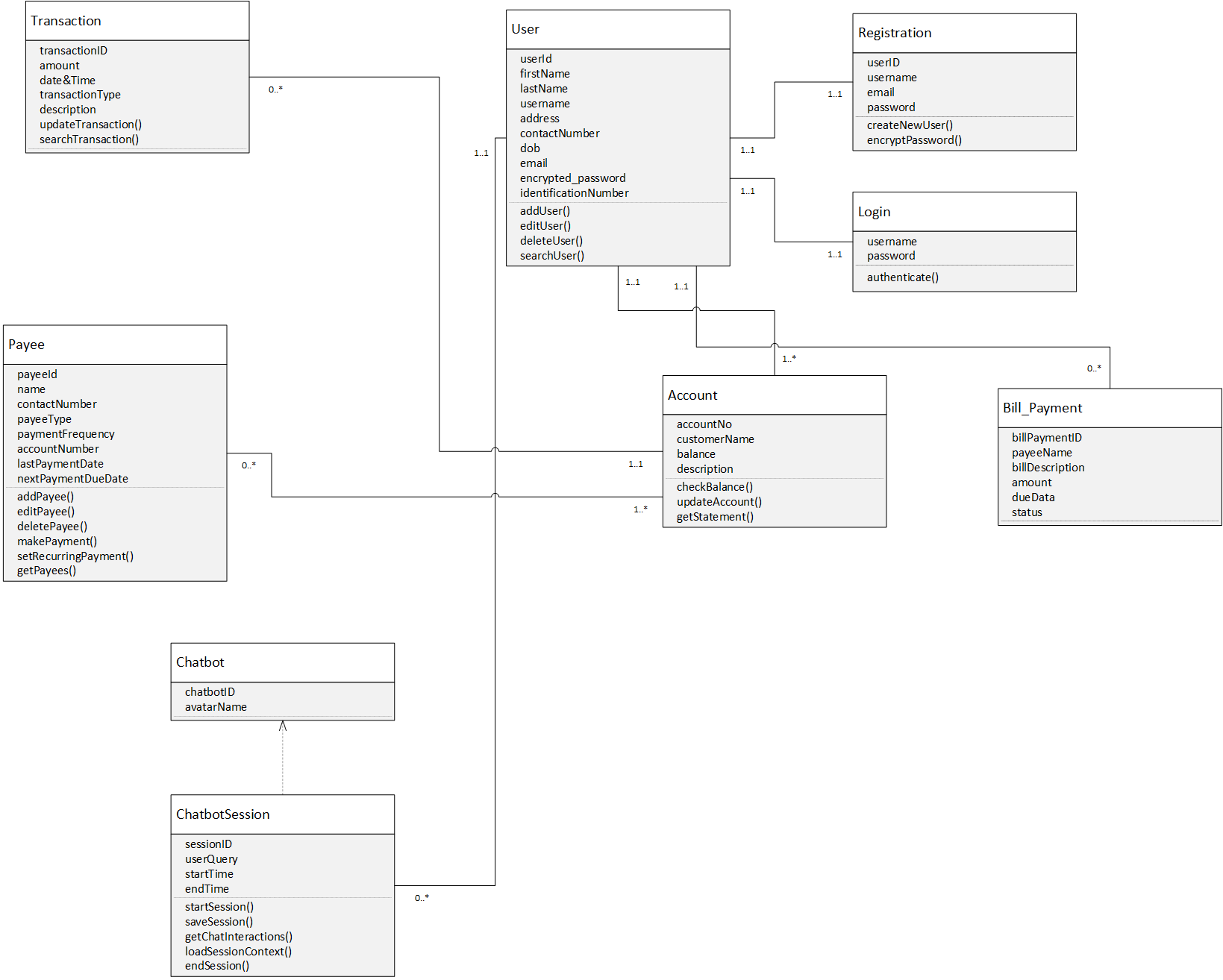


Figure 1: Domain class diagram for online banking app for older persons

## Activity Diagrams

**Activity Diagrams (Rizwan):**

1. User transfers fund into an account

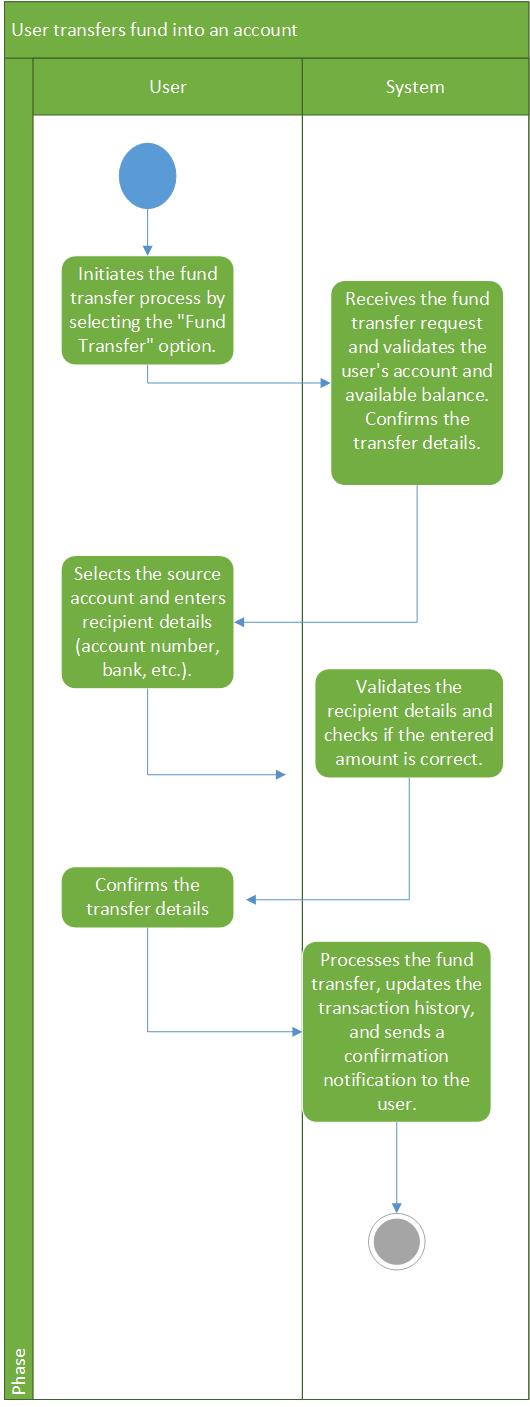


Figure 2: User transfer funds

1. User creates account

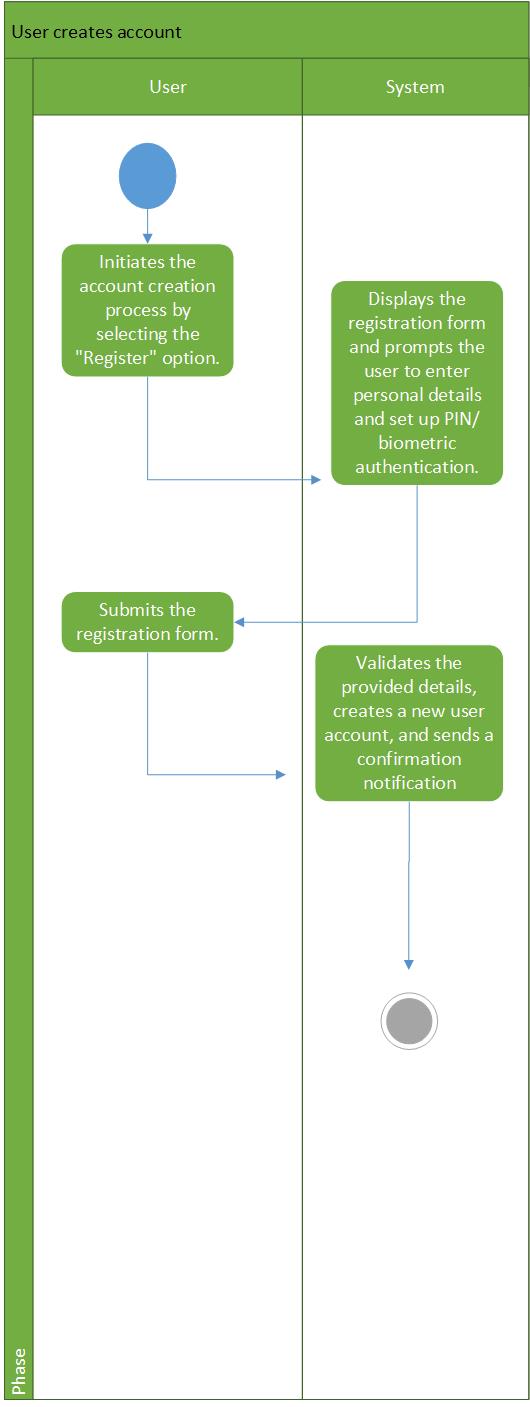


Figure 3: User creates account

1. User makes a bill payment

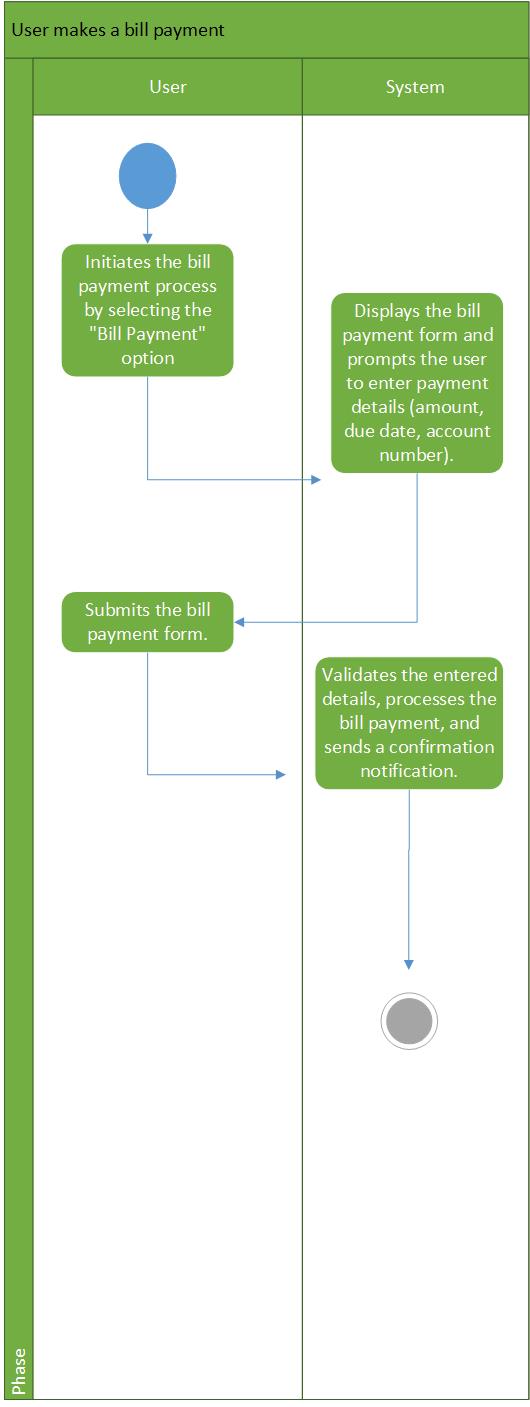


Figure 4: User makes bill payment

1. User searches for a transaction in transaction history

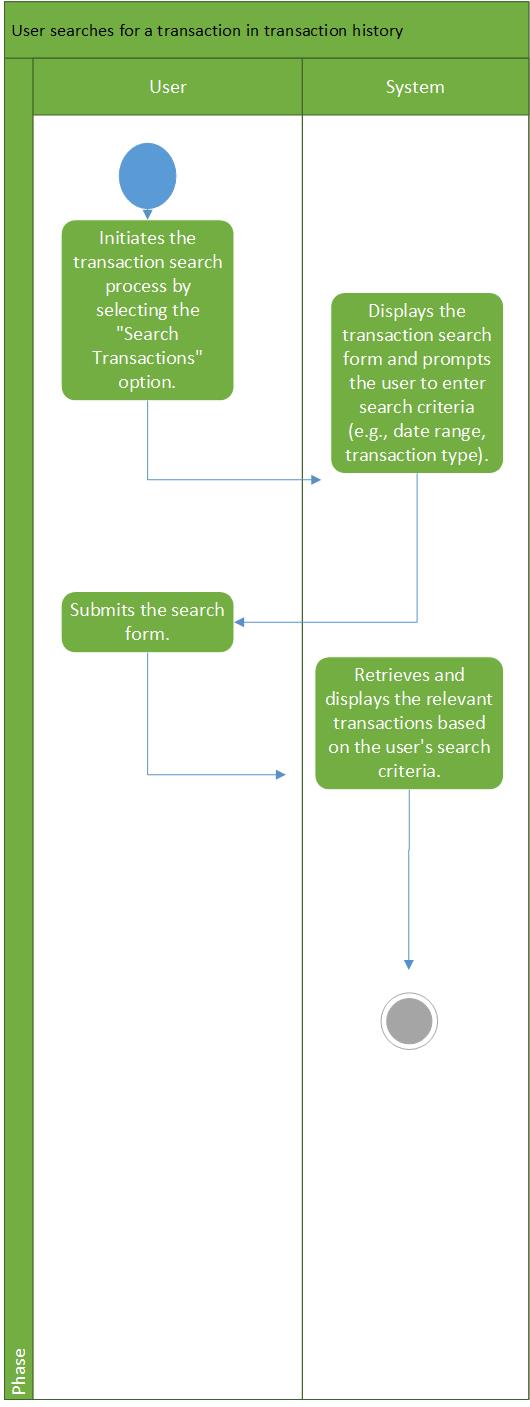


Figure 5: User searches transaction history

1. User interacts with chatbot about a query [e.g., show me my transaction history, or increase font size]

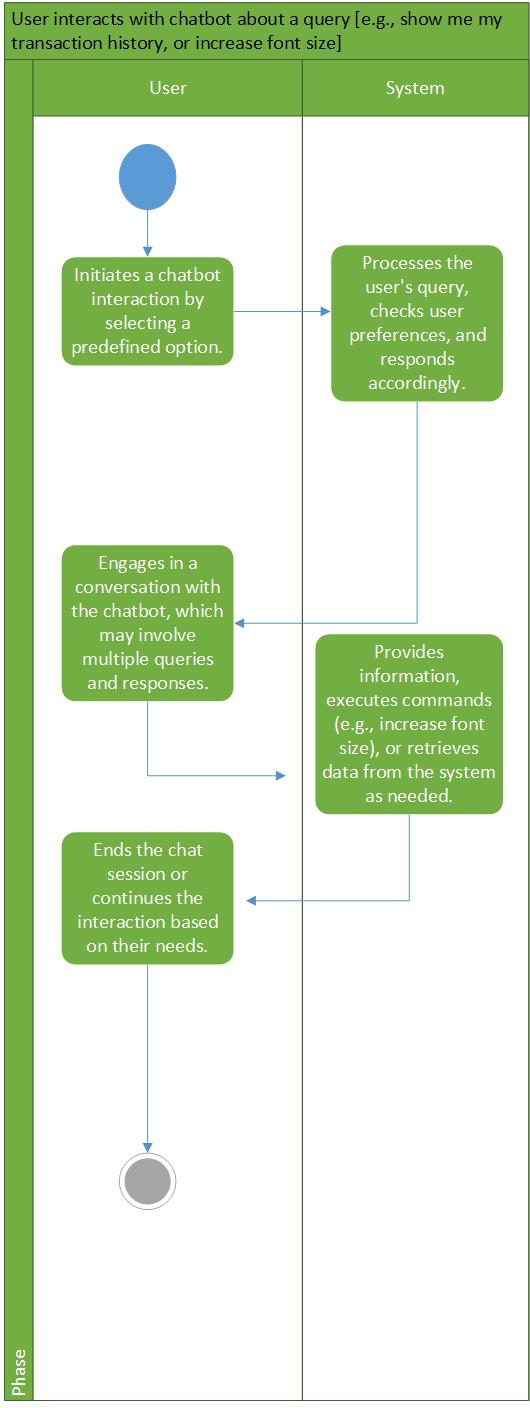


Figure 6:User chatbot interaction

# User Interfaces

Following are the application interfaces. Please note that some variation is present in the final project as updates/changes were made during implementation.

A screenshot of a login form

Description automatically generated

Figure 7: User Welcome Page

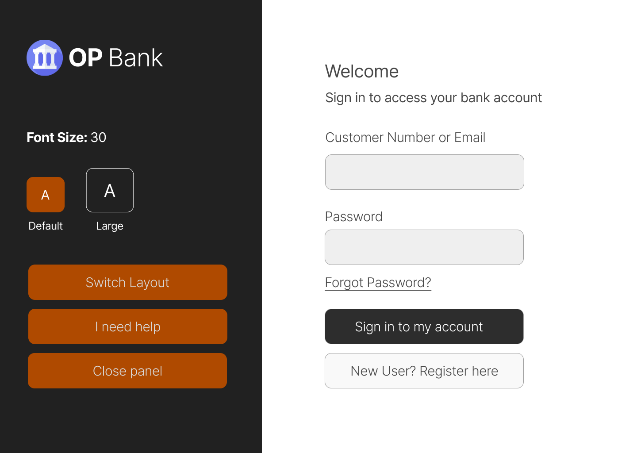


Figure 8: Accessibility Panel

A screenshot of a login form

Description automatically generated

Figure 9: Enlarge font size

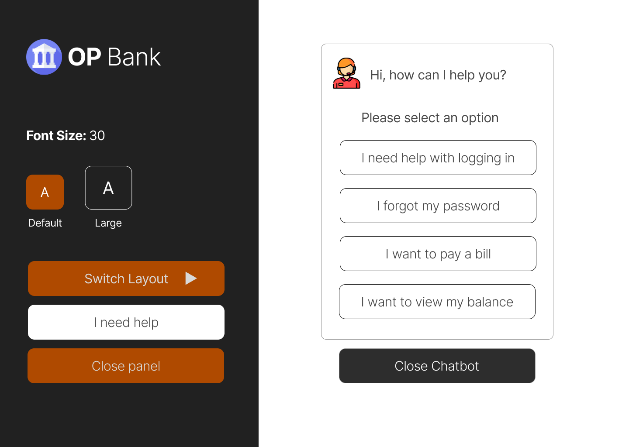


Figure 10: Chatbot

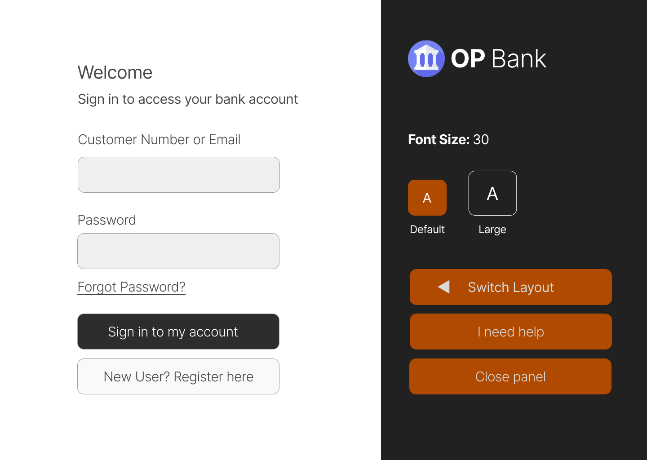


Figure 11: Switch Layout

A screenshot of a register account

Description automatically generated

Figure 12: Register Account

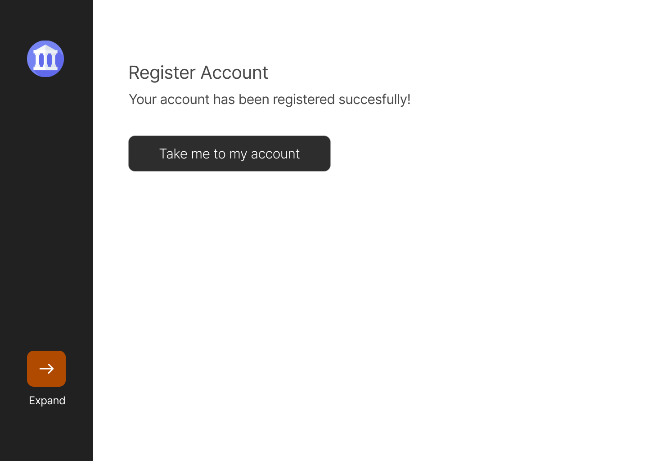


Figure 13: Registration Success

A screenshot of a account

Description automatically generated

Figure 14: User Dashboard

A screenshot of a computer

Description automatically generated

Figure 15: User Options

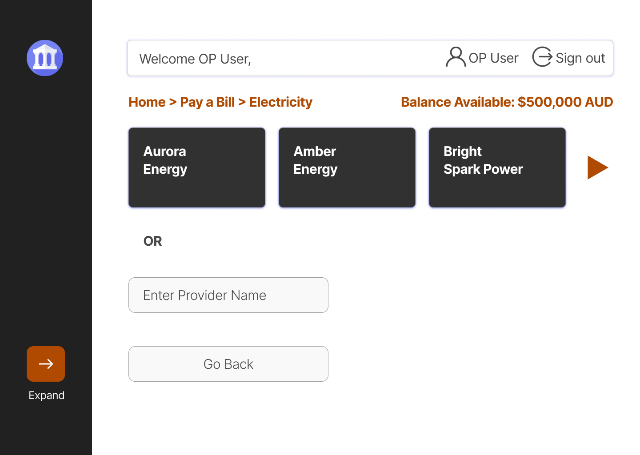


Figure 16: User Select Provider

A screenshot of a computer

Description automatically generated

Figure 17: User Bill Details

A screenshot of a computer

Description automatically generated

Figure 18: User Bill Paid Successfully

# Implementation & Testing

The Project implementation has been done in two components. The first one is the front-end that contains the code for designing and implementing the user interface. It contains the required React, Node.js and Express.js, and CSS libraries and dependencies required to complete the interface. The back-end component connects the front-end with the MySQL database.

**6.1 Overview of the Implementation:**

Table 1: OP Bank Project - Architecture and Technologies

|  |  |
| --- | --- |
| Front-end | React |
| State management | Redux with @reduxjs/toolkit |
| HTTP Requests | axios |
| Date and Time handling | moment |
| User Interface Libraries | react-icons, react-tiny-popover |
| Routing | react-router-dom |
| User Feedback/Notifications | React-toastify |
| Form Captcha | Google Captcha |
| Development Tools | Vite, Eslint, Tailwind CSS, Autoprefixer |
| TypeScript | TypeScript type definitions for React and ReactDOM are used for TypeScript support |
| Database | MySql |

**6.2 Environment Setup**

1. **Node.js and npm**: Ensure that Node.js is installed on the system, which comes with npm (Node Package Manager). Download and install Node.js from the official website: <https://nodejs.org/>

Verify that Node.js and npm are installed by running the following commands in the terminal:

bashCopy code

node -v npm -v

1. **Vite**: The project uses Vite as a development tool and build system. Install Vite globally using npm:

bashCopy code

npm install -g create-vite

Verify the installation of Vite by running:

bashCopy code

create-vite --version

1. **Eslint**: Eslint is used for code linting. Install it globally using npm:

bashCopy code

npm install -g eslint

Verify the installation of Eslint by running:

bashCopy code

eslint -v

1. **Tailwind CSS**: Tailwind CSS is used for styling the application. It doesn’t need to be installed globally, as it's managed as a project dependency. However, make sure it's listed in the project's **package.json** dependencies.
2. **Other Project Dependencies**: To install the other project-specific dependencies, navigate to the project's root directory in the terminal and run:

bashCopy code

npm install

This will install all the project dependencies specified in the **package.json** file

**6.3 Code Structure**

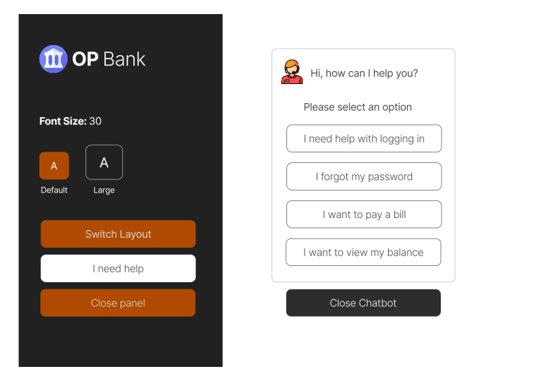
A screenshot of a computer program

Description automatically generated

Figure 19: Code directory structure

|  |  |  |
| --- | --- | --- |
| **Folder/File Name** | **Details** | |
| **Front End** | | |
| Public | Contains the vite.svg static asset | |
| src/app/store.js | Used to create and configure the Redux store | |
| src/assets | Contains assets used in the UI/app | |
| src/components/ | Contains javascript files for the following components: - BillHistory  - Dashboard  - PayBill  - ProviderPage  - Shared  - Sidebar  Details and code for these components can be viewed in a code editor (MS Visual Studio or VS Code) | |
| src/features/authSlice.js | Redux slice definition for managing authentication-related state in a Redux store. This slice is designed to handle user authentication, including login, logout, and updating user information. | |
| src/pages/ | Contains javascript files to view the for the following pages:  BillDetailsPage.jsx BillHistoryPage.jsx  ChatbotPage.jsx  ChooseProvider.jsx  ErrorPage.jsx  PayBillPage.jsx  RegisterPage.jsx  RegisterSuccessPage.jsx  SignInPage.jsx  Details and code for these pages can be viewed in a code editor (MS Visual Studio or VS Code) | |
| index.css | Defines css style of the application. The app uses Tail wind framework for responsive design, with Montserrat as main font. | |
| main.jsx | Sets up a React application with specific configurations, including Redux for state management, client-side routing, and toast notifications. It initializes the application and renders it into the specified HTML element, allowing users to interact with the React-based user interface. | |
| Routes.jsx | Defines and configures the routes and navigation structure of the React application. This includes:  - Importing components and libraries for routing, styling and user authentication  - Checks for user authentication and validates user session with user token  - Renders the sidebar that is displayed on the UI for accessibility. Also controls the layout.  Details and code for this file can be viewed in a code editor (MS Visual Studio or VS Code) | |
| .env | This is the environment file to store sensitive information. It does the following:  - Specifies the URL of the backend server/API for communication.  - Specifies the site key for Google’s reCAPTCHA service. | |
| .eslintrc.cjs | Config file for ES Lint Linter service. This specifies linting rules and settings tailored for this project. It extends recommended configurations for ESLint and React, specifies the parser options, ignores certain files and directories, and configures custom rules and plugins, including support for React Fast Refresh during development.  Details and code for this file can be viewed in a code editor (MS Visual Studio or VS Code) | |
| index.html | Serves as entry point for the web app. It contains app metadata and defines an empty div element with id ‘root’ to serve as the container for rendering the app. The following script loads and initializes the application with main.jx in the src directory as the entry point for the app:  <script type="module" src="/src/main.jsx"></script> | |
| Other files | package.json, package-lock.json, tailwind.config.js, vite.config.js. | |
| **Back-end / Database** | | |
| helpers/db.js | | Initializes the database connection using MySql |
| middleware/authorize.js | | Responsible for authenticating and authorizing incoming requests based on JSON Web Tokens  Details and code for this file can be viewed in a code editor (MS Visual Studio or VS Code) |
| middleware/error-handler.js | | For managing unexpected errors. If the back end encounters an authentication error, the middleware logs it on the console and sends a JSON response with a 500-status code and error message.  Details and code for this file can be viewed in a code editor (MS Visual Studio or VS Code) |
| middleware/validate-request.js | | Validates incoming request data against a specified schema, ensuring that the data conforms to the expected format and constraints. If validation fails, it passes an error message to the next middleware in the request pipeline. If validation succeeds, it updates the **req.body** object with the validated data and continues the request handling process.  Details and code for this file can be viewed in a code editor (MS Visual Studio or VS Code) |
| users/user.model.js | | Defines a Sequelize model for a "User" entity with specific attributes and options. It enforces constraints on the fields, such as requiring certain fields to be non-null. Additionally, it sets up a default scope to exclude the user's password when querying the model by default, enhancing security.  Details and code for this file can be viewed in a code editor (MS Visual Studio or VS Code) |
| users/user.service.js | | Provides a set of services for user authentication and management. It handles user authentication, user creation, and retrieval of user data while incorporating security measures like password hashing and JWT-based authentication.  For password hashing, the app uses ‘bcryptjs’ which is JS library for securely hashing passwords. It also has salted hashing to ensure that in case of same user password, the hashed passwords are different.  Details and code for this file can be viewed in a code editor (MS Visual Studio or VS Code) |
| users/users.controller.js | | Defines routes and controller functions for user-related operations, including authentication, registration, and data retrieval  Details and code for this file can be viewed in a code editor (MS Visual Studio or VS Code) |
| config.json | | Config file for database host, port, user details/password and db name. |
| Other files | | server.js, package.json, Readme.md. |

**Chatbot:**



On the home screen when the OP clicks the button when in need of help, the section in displayed, provided in the above image. The user can select one option at a time. The choice of help questions is limited to four to avoid confusion and jargon. When clicked on the button, the explanation is plain and simple language is provided to the OP.

The approach used to implement the chatbot was to use if-else statement in the front-end code. The two-way interaction between elderly and the application would have been implemented in larger timeframe. Because of time constraints, the training and tree style based chatbot was aborted.

**Implementation Constraints:**

There are several implementation constraints that emerged during the design, development, and implementation of the project. These are:

* There is no forgot user password flow.
* The prototype does not have a payment system. The ‘Pay a bill’ feature has only been implemented at the interface level.
* User Notification and Settings section has not been implemented.
* In the design of our chatbot, we specified a class ‘ChatbotSession’ which manages and save user interactions with the chatbot with a unique identifier; session ID. This class depends on ‘Chabot’ class which also maintain chatbot id and avatar. However due to time constraints we had to change our approach in implementing chatbot for our web app. The app has a basic implementation of chatbot.

**6.6 Test Cases**

We identified the following set of test cases as part of our testing strategy for the project:

* + - 1. User creates a new account
      2. User provides correct login details
      3. User pays a bill
      4. User Age verification
      5. User Switches Layout
      6. New user is created in the database
      7. User views bill history

Following members performed the test cases:

Fahad Dahish (FD), Shevindi Rodrigo (SR), Muhammad Rizwan (MR), and Khalid Mohammed (KM)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test id** | **Title** | **Participant** | **Planned date** | **Planned Location** | **Responsibility** | **Requirements** | **Platform** |
| 1.1 | Create a new account | FD | 05/11/2023 | Online | FD | * Laptop or PC * Local host | Any |
| 1.2 | Correct Login | FD | 05/11/2023 | Online | FD | * Laptop or PC * Internet | Any |
| 1.3 | Paying a bill | SR | 05/11/2023 | Online | SR | * Laptop or PC   Local host | Any |
| 1.4 | Age Verification: Eligibility for Users aged 60 years and above. | SR | 05/11/2023 | Online | SR | * Laptop or PC * Local host | Any |
| 1.5 | Switch Layout Button | MR | 06/11/2023 | Online | MR | * Laptop or PC * Local host | Any |
| 1.6 | Create new user in database | MR | 07/11/2023 | Online | MR | * Laptop or PC * Local host | Any |
| 1.7 | User views bill history | KM | 07/11/2023 | Online | MR | * Laptop or PC * Local host | Any |

**Test Cases:**

|  |  |
| --- | --- |
| Test ID | 1.1 |
| Title | Create a new account |
| Feature | Create a new account on OP-bank webapp |
| Objective | Confirm that the user can enter data to database and create a new account. |
| Setup | All common web browser (Chrome, Firefox, Safari) |
| Test Data | The account should be created after added the account details: (Customer Num or Email, Date of birth > 60 years, Password, human varication) and click Register button. |
| Test Actions | Access Create an account page for the OP-bank webapp.  Input the new account details: (Email Address or Customer Num, Date of birth > 60 years, Password).  Checking the human varication (captcha).  Click the Register button. |
| Expected results | The system creates the new account and store account details on the MySQL database and then the user is shown the register account successfully page take me to my account of the site (banking dashboard). |

|  |  |
| --- | --- |
| Test ID | 1.2 |
| Title | Correct Login |
| Feature | Login to OP-banking webapp using user credentials |
| Objective | Confirm that a correct email/Customer number and password yields access to the webapp as expected |
| Setup | All common web browser (Chrome, Firefox, Safari) |
| Test Data | Login information  Customer Number: 20203030  Password: asdf1234 |
| Test Actions | Access login page for the OP-banking webapp  Input Customer Number and password and click login button |
| Expected results | Verify the login credentials without receiving any error alerts. Send the user to the OP-banking web app's dashboard or account overview page. After a successful login, the user's email or customer number should be displayed |

|  |  |
| --- | --- |
| Test ID | 1.3 |
| Title | Paying a bill |
| Feature | Paying an electricity bill to a saved provider |
| Objective | Confirm that user pays the electricity bill for a pre-saved provider successfully |
| Setup | All common web browser (Chrome, Firefox, Safari) |
| Test Data | Login information  Email Address: test12@gmail.com  Password: 1234567 |
| Test Actions | 1. Access login page for the OP-Bank web app 2. Input valid email address or customer account number and valid password. 3. Human verification using captcha. 4. Access dashboard after successful login. 5. Click “Pay a bill” panel to pay the bill. 6. Choose “Electricity” pane to pay electricity bill. 7. Choose the pre-saved provider: “Aurora Energy” to make the payment. 8. Verify the bill details. 9. Click Make a Payment to pay the bill. |
| Expected results | System displays a big green tick to show that the user made the payment on the bill successfully. |

|  |  |
| --- | --- |
| Test ID | 1.4 |
| Title | Age Verification: Eligibility for Users aged 60 years and above. |
| Feature | User registration eligibility check |
| Objective | Confirm that users who are aged 60 years and older are allowed to register to OP-Bank, while those below this age are denied the access for registration process. |
| Setup | All common web browser (Chrome, Firefox, Safari) |
| Test Data | Login information  Email Address: test12@gmail.com  Password: 1234567 |
| Test Actions | 1. Access Registration page for the OP-Bank web app. 2. Enter the customer number/email address and enter the password. 3. Enter the address. 4. Enter the date of birth in which the birth year should be 1960 or older. |
| Expected results | System should allow to continue the registration process without giving an erroe. |

|  |  |
| --- | --- |
| Test ID | 1.5 |
| Title | Switch layout button |
| Feature | Checking the UI shift from left to right and back to left after clicking switch layout button. |
| Objective | Confirm that users who are aged 60 years and older are able to use the Switch layout button UI shift from left to right and back to left after clicking switch layout button. |
| Setup | All common web browser (Chrome, Firefox, Safari) |
| Test Data | Login information  Email Address: test12@gmail.com  Password: 1234567 |
| Test Actions | 1.Access Registration page for the OP-Bank web app.  2. Click the switch layout button and confirm the UI switch from left to right and again back to left after clicking for second time. |
| Expected results | Confirm that users will be able to use the Switch layout button UI shift from left to right and back to left after clicking switch layout button. |

|  |  |
| --- | --- |
| Test ID | 1.6 |
| Title | Create new User in database |
| Feature | The table in the backend can store the details provided by users at the time of registering. |
| Objective | To ensure smooth execution of user records storage and retrieval. |
| Setup | All common web browser (Chrome, Firefox, Safari) |
| Test Data | Login information  Email Address: test12@gmail.com  Password: 1234567 |
| Test Actions | Access Registration page for the OP-Bank web app.  Enter the customer number/email address and enter the password.  Enter the address.  Enter the date of birth in which the birth year should be 1960 or older.  Next, open the backend table in mySQL workbranch |
| Expected results | After opening the backend table in mySQL workbranch, check if the data is stored correctly and if it is being retrieved correctly. |

|  |  |
| --- | --- |
| Test ID | 1.7 |
| Title | View Bill History |
| Feature | View Bill History on OP-bank webapp |
| Objective | Confirm that the user can enter and see all bill history transaction |
| Setup | All common web browser (Chrome, Firefox, Safari) |
| Test Data | User credentials with existing bill payment history |
| Test Actions | Navigate to the OP-bank webapp login page.  2. Enter valid user credentials (Email Address or Customer Number, and Password).  3. Complete any required human verification (captcha).<br>4. Click the 'Login' button to access the account.  5. Navigate to the 'Bill Payment' section of the dashboard  6. Click on the 'View Bill History' link or button. |
| Expected results | The user can log in and navigate to the 'Bill Payment' section. Upon clicking 'View Bill History', the user should see a list of all past bill transactions, with details such as biller name, amount paid, date of transaction, and payment status. |

# Conclusion

The OP Bank prototype focuses on providing a simple, text-focused, and intuitive user interface to help older users manage their finances independently in the digital age. Key features include secure authentication, easy access to account balance and transaction history, fund transfers, notifications, and the integration of a chatbot assistant.

The project defines a set of business rules and functional requirements to ensure that the application caters to the unique requirements of older users. These requirements encompass user authentication, account management, notifications, search functionality, and accessibility standards compliance. Additionally, non-functional requirements emphasize efficiency, error handling, response times, and scalability to create a smooth and reliable user experience.

While not all requirements have been implemented in the prototype due to time and resource constraints, they serve as a comprehensive guide for potential real-world implementation. OP-Bank strives to provide older adults with a secure and user-centric online banking experience, promoting financial independence and accessibility for this demographic.

# References

Tajudeen, F. P., Bahar, N., Tan, M. P., Mustafa, M. B., Saedon, N. I., & Jesudass, J. (2022, October 1). *Understanding User Requirements for a Senior-Friendly Mobile Health Application*. Geriatrics; Multidisciplinary Digital Publishing Institute. <https://doi.org/10.3390/geriatrics7050110>