**Bank Management System Requirements Specification**

**Version 3.0**

**March 8th, 2025**

***Project Overview***

1. Project Title: Bank Management System

2. Introduction:The Bank Management System (BMS) is a software solution designed to automate and streamline banking operations, ensuring efficient customer service, secure transactions, and regulatory compliance. This system will allow customers to manage their accounts, transfer funds, apply for loans, and receive notifications while enabling bank employees to manage transactions, loan approvals, and account security.

3. Objectives:

•To provide secure and efficient banking services for customers.

•To automate fund transfers, account management, and loan applications.

•To enhance security with encryption and fraud detection mechanisms.

•To improve customer experience through an intuitive online banking platform.

•To ensure compliance with financial regulations and reporting requirements.

4. Scope of the Project:

Included Features:

✅ Customer Features:

•Secure Login & Authentication (with OTP)

•Account Overview (Balance, Transaction History)

•Fund Transfers (Internal & External)

•Loan Application & Repayment Tracking

•Notifications & Alerts

✅ Bank Employee Features:

•Customer Management (New Accounts, Profile Updates)

•Transaction Monitoring & Approvals

•Loan Processing & Approval System

•Report Generation

✅ Admin Features:

•User & Role Management

•Fraud Detection & Security Logs

•Compliance & Audit Reporting

Excluded Features:

❌ Cryptocurrency transactions

❌ Stock market trading integration

❌ ATM management

5. Target Users:

•Bank Customers: Individuals & businesses managing their finances.

•Bank Employees: Customer service agents, loan officers, and account managers.

•System Administrators: IT personnel managing security and compliance.

**Product/Service Description**

**1. Product Context:**

The Bank Management System operates as an independent platform but may optionally interface with external systems such as payment gateways, regulatory reporting APIs, and notification services (email/SMS). It is self-contained in handling core banking features but supports integration points for scalability. The system consists of modules like account management, transaction processing, loan handling, reporting, and security, which communicate through well-defined interfaces.

You may optionally show a context diagram in your final documentation with components such as:

* External Notification Service
* Regulatory API
* Database System
* Admin Control Panel

**2. User Characteristics:**

**Customer (Individual/Business):**

* General computer/mobile app users
* Varying levels of digital literacy
* Expect a secure and user-friendly interface

**Bank Employee:**

* Staff-level users with moderate technical expertise
* Familiar with banking processes and regulations
* Require efficiency and access to filtered customer data

**System Administrator:**

* Advanced IT personnel
* High technical expertise in security, systems, and compliance
* Responsible for backups, logs, and role-based access control

**3. Assumptions:**

* The system will run on a modern web infrastructure with SSL support.
* A secure, relational database system is available (e.g., MySQL, PostgreSQL).
* All users have stable internet access and devices capable of running web applications.
* Employees are trained in basic system use before deployment.
* Operating system and browser compatibility will be limited to common platforms (Windows/macOS/Linux and Chrome/Edge/Firefox).

**4. Constraints and Dependencies:**

* The system must generate and store audit logs for all financial and administrative operations.
* It must support role-based access control with layered permissions (Admin > Employee > Customer).
* Daily backup operations are required to ensure system integrity.
* Account creation and loan approval modules are dependent on validation modules being successfully integrated.
* Fraud detection features must operate in parallel with core transaction flows to flag suspicious behavior in real time.
* System should integrate with a notification API (email/SMS) for real-time alerts.
* This new product will require a daily download of data from X,
* Module X needs to be completed before this module can be built.

***Requirements***

Functional Requirements

(1-4 Enklajd Hodo)

1. Customer shall be able to create a bank account and view account balance.

2. Customer shall be able to transfer money and receive transaction alerts.

3. Customer shall be able to apply for and view their loan details.

4. Customer shall be able to manage cash (withdraw/deposit) and update security options like PIN.

(5-8 Florjon Allkaj)

5. Bank employee shall handle loan logic (eligibility, interest, and repayment schedule).

6. Bank employee shall manage and approve/reject customer loan applications.

7. Customer shall be able to request a checkbook.

8. Bank employee shall be able to approve or reject account creation requests.

(9-12 Elvis Hamati)

9. Bank employee shall be able to view all client accounts.

10. Bank employee shall be able to update client account information.

11. Bank employee shall be able to check client loan status.

12. Administrator shall be able to manage employee accounts.

(13-16 Alonso Tiko)

13. Administrator shall be able to monitor all transactions.

14. Administrator shall be able to freeze or close accounts.

15. Administrator shall be able to generate reports on transactions.

16. System shall be able to validate account details during creation.

(17-20 David Kapxhiu)

17. System shall be able to generate monthly account statements for customers.

18. System shall be able to apply security measures for online transactions.

19. System shall be able to perform automatic currency conversion for international transactions.

20. System shall be able to block suspicious transactions.

(21-24 Erdi Meci)

21. Customer shall be able to request account suspension.

22. System shall provide chatbot assistance for FAQs.

23. System shall auto-log out users after inactivity for security.

24. Bank employee shall be able to reset customer login credentials.

(25-28 Fabian Sulo)

25. Customer shall be able to request a debit card online.

26. Bank employee shall be able to schedule appointments with customers for financial consultations.

27. Administrator shall be able to perform system backups and restore operations.

28. Customer shall be able to provide feedback on banking services via an integrated feedback portal.

***Non-Functional Requirements***

**Product Requirements**.

**3.2.1.1    Usability Requirements**

* The system should be easy to use for customers, requiring minimal training to perform common actions
* The user documentation and help section should be complete and accessible to help users understand all system features
* The system should have an intuitive interface that minimizes the learning curve for new users
* The system should provide helpful error messages when an invalid action is performed
  + - 1. **Performance Requirements**
* The system should be able to handle at least 1000 simultaneous users without degradation in performance
* 95% of transactions shall be processed in less than 1 sec
* The system should be able to process at least 10000 transactions per day during break periods
* The system should handle data storage requirements of up to 1TB of transactionsal data without performance issues

**3.2.1.3       Availability**

* The system should have an availability of 99.9%, ensuring minimal downtime
* The system should support access from multiple geographic regions, including remote branches and online banking services
* Scheduled maintenance should be communicated to users at least 24 hours in advance, and downtime should not exceed 2 hours per month
* The maximum allowed downtime for unscheduled outages should be 4 hours per year.
* The system should be able to recover from failure within 5 minutes to minimize impact on user activities

**3.2.1.4        Security**

* All user data and transactions should be encrypted using AES-256 encryption
* The system should log all user activity for auditing and troubleshooting purposes, with logs being stored for a minimum of 1 year.
* Access to the system’s backend should be restricted to authorized bank employees only, with role-based access control
* The system should conduct integrity checks on user data and transactional records to ensure no unauthorized modifications occur
* Two-factor authentication should be required for all admin-level users   
    
  + 1. **Organizational Requirements**
* The bank’s internal policies must be adhered to during the development, ensuring compliance with financial regulations
* The system should be compatible with the bank’s existing infrastructure, including hardware and software systems
* The development process must follow the Agile methodology for continuous improvement and rapid feedback cycles
* The system should support integration with existing third-party systems, such as payment gateways and credit scoring providers  
  + 1. **External Requirements**
* The system should comply with the GDPR for data protection and privacy of EU customers
* The system must comply with industry standards such as PCI DSS for secure handling of payment card data.
* The system should integrate with government systems for tax reporting and financial audits where necessary.
* The system should support interoperability with other banks for cross-bank transfers (e.g., SWIFT, SEPA).
* The system must ensure that all software components comply with local and international banking regulations.

1. User Scenarios/Use Cases

UC 1-4 Enklajd Hodo

|  |  |
| --- | --- |
| UC Name | *UC\_01 Request for Account Creation* |
| Summary | *This use case describes the process by which a customer requests to create a bank account and the system processes the account creation.* |
| Dependency | *None* |
| Actors | *Customer, Banking System, Bank Administrator (for approval if required)* |
| Preconditions | 1. *The customer must provide valid identification and personal details.* 2. *The banking system must be operational.* |
| Description of the Main Sequence | ***Request for Account Creation***   * *Step 1: Customer navigates to the "Request Account Creation" section on the banking platform.* * *Step 2: Customer fills out the account application form with required details (e.g., name, address, contact information, ID proof, initial deposit amount if applicable).* * *Step 3: System verifies the entered details.* * *Step 4: System submits the request for processing.* * *Step 5: System provides a confirmation message with a request ID.* * *Step 6: If manual approval is required, the bank administrator reviews the request and approves or rejects it.* * *Step 7: If approved, system proceeds with account creation.* |
| Description of the Alternative Sequence | *1a. If customer enters incomplete or incorrect details:*   * *System prompts the customer to correct the information.* * *Customer resubmits the request.*   *2a. If system fails due to technical issues:*   * *System notifies the customer and suggests retrying later.*   *3a. If manual approval is required and the request is rejected:*   * *System notifies the customer with the reason for rejection.* * *Customer may reapply with corrected details.* |
| Non functional requirements | * *The account request process should take no more than 5 minutes.* * *The system should process approved account creation within 10 seconds.* * *All customer data must be securely stored and encrypted.* * *The system should send real-time notifications for request updates.* * *The system should be available 24/7.* |
| Postconditions | 1. *A new bank account is successfully created if the request is approved.* 2. *The customer receives a confirmation notification.* 3. *The customer can access their account details upon successful login.* |
| UC Name | *UC\_02 Manage Bank Account* |
| Summary | *This use case describes the process by which a customer creates a bank account, views their account balance, transfers money, and receives transaction alerts.* |
| Dependency | *None* |
| Actors | *Customer* |
| Preconditions | 1. *The customer must be a registered user of the banking system.* 2. *The customer must provide valid personal information to create an account.* 3. *The customer must have sufficient balance to initiate a transfer.* |
| Description of the Main Sequence | * *Step 1: Customer navigates to the "Create Account" section of the banking system.* * *Step 2: Customer enters required details such as name, address, contact information, and initial deposit amount.* * *Step 3: System validates the entered details.* * *Step 4: System creates a new bank account and generates an account number.* * *Step 5: System displays a confirmation message along with the newly created account details.* * *Step 6: Customer logs in to the banking system.* * *Step 7: Customer navigates to the "Account Balance" section.* * *Step 8: System retrieves and displays the customer’s current account balance.* * *Step 9: Customer navigates to the "Transfer Money" section.* * *Step 10: Customer enters the recipient’s account details and the amount to transfer.* * *Step 11: System verifies the account details and checks for sufficient balance.* * *Step 12: System processes the transaction and transfers the amount.* * *Step 13: System generates a transaction alert and notifies both the sender and recipient.* |
| Description of the Alternative Sequence | *1a. If customer enters invalid details:*   * *System prompts customer to re-enter correct details.* * *Customer re-enters details and submits the request again.*   *2a. If system fails to create an account due to technical issues:*   * *System notifies customer of the failure and suggests retrying later.*   *3a. If customer enters incorrect login credentials:*   * *System displays an error message and prompts for correct credentials.*   *4a. If customer enters an invalid recipient account:*   * *System notifies the customer and prompts for correct details.*   *5a. If customer has insufficient funds:*   * *System alerts the customer and cancels the transfer.* |
| Non functional requirements | * *The system should process account creation within 5 seconds.* * *The account balance retrieval should not exceed 3 seconds.* * *The money transfer should be completed within 10 seconds.* * *The system must encrypt all sensitive customer data.* * *The system should be available 24/7.* * *The system must send transaction alerts in real time.* |
| Postconditions | 1. *A new bank account is created if the request is valid.* 2. *The customer successfully views their account balance.* 3. *Money is transferred to the intended recipient if all validations pass.* 4. *Both the sender and the recipient receive transaction alerts.* 5. *All sensitive data remains secure through encryption.* 6. *System availability and response times meet performance standards.* |

|  |  |
| --- | --- |
| UC Name | *UC\_03 Loan Details* |
| Summary | *This use case describes the process by which a customer applies for a loan and views their loan details through the banking system.* |
| Dependency | *None* |
| Actors | *Customer* |
| Preconditions | 1. *The customer must be a registered user of the banking system.* 2. *The customer must have a valid bank account.* 3. *The customer must meet the eligibility criteria for loan application (e.g., credit score, income proof).* |
| Description of the Main Sequence | * *Step 1: Customer logs in to the banking system.* * *Step 2: Customer navigates to the "Loan Application" section.* * *Step 3: Customer selects the type of loan and enters required details (e.g., loan amount, duration, purpose).* * *Step 4: System validates the entered details and checks eligibility criteria.* * *Step 5: System processes the application and generates a loan application ID.* * *Step 6: System notifies the customer of the loan application status (approved, pending, or rejected).* * *Step 7: If approved, customer navigates to the "Loan Details" section.* * *Step 8: System retrieves and displays the loan details (e.g., principal amount, interest rate, repayment schedule, outstanding balance).* |
| Description of the Alternative Sequence | ***4a. Invalid Details:***   * *The system prompts the customer to re-enter the correct information.* * *The customer re-enters and resubmits the application.*   ***2a. System Processing Failure:***   * *The system notifies the customer of a technical issue and suggests trying again later.*   ***3a. Loan Application Rejected:***   * *The system displays the reason for rejection.* * *The system suggests alternative options (e.g., applying for a different loan type).*   ***7a. No Active Loan Found:***   * *If the customer tries to access loan details without an active loan, the system notifies them that no active loans exist.* |
| Non functional requirements | * *The system should process loan applications within 10 seconds.* * *The loan details retrieval should not exceed 5 seconds.* * *The system must encrypt all sensitive customer data.* * *The system should be available 24/7.* * *The system must notify the customer via SMS/email regarding loan application status.* |
| Postconditions | * *The loan application is successfully submitted and recorded in the banking system.* * *The customer can view loan details if the application is approved.* * *The system updates the loan status and notifies the customer accordingly.* |

|  |  |
| --- | --- |
| UC Name | *UC\_04 Update Security Options* |
| Summary | *This use case describes how a customer can update their security options, specifically changing their PIN.* |
| Dependency | *None* |
| Actors | *Customer* |
| Preconditions | 1. *The customer must be a registered user of the banking system.* 2. *The customer must have an active bank account.* |
| Description of the Main Sequence | *Update Security Options (Change PIN):*   1. *The customer logs in to the banking system.* 2. *The customer navigates to the "Security Settings" section.* 3. *The customer selects the "Change PIN" option.* 4. *The customer enters the current PIN and the new PIN.* 5. *The system verifies the current PIN.* 6. *If the current PIN is correct, the system updates the PIN.* 7. *The system sends a confirmation message to the customer regarding the successful change.* |
| Description of the Alternative Sequence | ***Incorrect Current PIN:***   * *If the customer enters an incorrect current PIN while attempting to change the PIN, the system prompts the customer to re-enter the correct PIN.*   ***PIN Update Failure (Security Policies):***   * *If the new PIN does not meet security requirements (e.g., complexity or length), the system notifies the customer and provides guidelines for creating a valid PIN.* |
| Non functional requirements | * + *PIN updates should be completed within* ***3 seconds****.*   + *The system must encrypt sensitive data, including PINs.*   + *Real-time alerts should be sent immediately after a successful PIN change.*   + *The system must be available* ***24/7****.* |
| Postconditions | 1. *The customer's PIN is successfully updated after the change process.* 2. *The customer receives a notification confirming the successful PIN update.* |

UC 5-8 Florjon Allkaj

|  |  |
| --- | --- |
| UC Name | UC\_5 Bank employee shall handle loan logic (eligibility, interest, and repayment schedule). |
| Summary | This requirement specifies that the banking system must automatically manage the loan processing logic. It involves checking whether a customer meets the eligibility criteria, calculating the applicable interest based on defined parameters, and generating a detailed repayment schedule. Essentially, it streamlines and automates the evaluation and structuring of loans, ensuring consistent, accurate, and timely processing of loan applications. |
| Dependency | For this requirement the dependency primarly involves : Customer loan application  Customer account data(validated user or not) |
| Actors | Primary actor, the customer |
| Preconditions | There should be a valid loan application, a verified customer and the eligibility criteria and loan parameters should be added |
| Description of the main sequence | 1. The Account submits a loan request with a specified amount.  2. The LoanRequest is created and initialized with the amount.  3. The BankEmployee reviews the loan request using the request ID.  4. If the customer is eligible, the LoanRequest creates a new Loan with the specified amount.  5. The Loan calculates the new loan balance and updates the Account accordingly. |
| Description of the alternative sequence | 1. The Account submits a loan request with a specified amount.  2. The LoanRequest is created.  3. The BankEmployee reviews the loan request.  4. If the customer is **not eligible**, the BankEmployee sends a notification message to the Account indicating the rejection. |
| Non functional requirements | * Performance: The system shall process loan applications within 5 seconds to ensure efficiency. * Security: All customer financial data shall be encrypted to protect sensitive information * Accuracy: The system shall be accessible 24/7 to handle loan applications without downtime. * Scalability: The system should support simultaneous loan evaluations for multiple customers without performance degradation. |
| Postconditions | * If the customer is eligible, the loan details are stored in the system and await approval from bank employees. * If the customer is not eligible, the system logs the rejection and notifies the customer. * The system has recorded loan request for future reference, regardless of the outcome. |

|  |  |
| --- | --- |
| UC Name | UC\_6 Bank employee shall manage and approve/reject customer loan applications. |
| Summary | This use case involves the bank employee reviewing customer loan applications and deciding whether to approve or reject them. |
| Dependency | |  | | --- | |  |   Depends on the "Handle Loan Logic" use case for the customer loan data being available. |
| Actors | Bank Employee |
| Preconditions | |  | | --- | | - The bank employee is logged in to the system.  Customer loan application has been submitted and is available for review after the system approved or rejected the loan application of the customer | |
| Description of the main sequence | 1. Bank employee logs into the system.  2. Bank employee views customer loan applications in the system.  3. Bank employee reviews the loan application details.  4. Bank employee approves or rejects the loan application.  5. System updates the loan application status to approved/rejected.  6. Customer is notified of the decision. |
| Description of the alternative sequence | * If the bank employee needs more information, they may request additional documents from the customer before making a decision. * If the employee rejects the application, the system notifies the customer with a rejection reason. |
| Non functional requirements | * Performance: The system shall process approval/rejection within 5 seconds to ensure efficiency. * Security: The system must ensure that customer data is handled securely during the review process. * Accuracy: The system shall be accessible 24/7 to handle approvals/rejections without downtime. * Scalability: The system should support simultaneous approvals/rejections for multiple bank employees without performance degradation. |
| Postconditions | * The loan application status is updated in the system as either approved or rejected. * The customer is notified of the outcome of the application. |

|  |  |
| --- | --- |
| UC Name | UC\_7 Customer shall be able to request a checkbook. |
| Summary | This use case involves the customer requesting a checkbook through the banking system. The bank processes the request and sends the checkbook to the customer. |
| Dependency | |  | | --- | |  |   none |
| Actors | Bank Employee, Customer |
| Preconditions | |  | | --- | | - The customer has a valid account.  - The customer is logged into the banking system. | |
| Description of the main sequence | 1. Customer logs into the system.  2. Customer navigates to the checkbook request section.  3. Customer submits a request for a checkbook.  4. Bank employee reviews the request and processes it.  5. System updates the checkbook request status.  6. The customer is notified once the request is approved and the checkbook is sent. |
| Description of the alternative sequence | -If the customer does not meet the criteria for receiving a checkbook, the request is rejected, and the customer is notified.  -If the customer is not eligible, a message is displayed explaining the reason. |
| Non functional requirements | * Performance: The system should process checkbook requests within 5 seconds. * Security: The system must ensure secure handling of customer data during the request process, customer data should be stored securely and be kept private. * Scalability: The system should support simultaneous checkbook requrests for multiple bank employees without performance degradation. |
| Postconditions | * The checkbook request status is updated in the system. * The customer is notified of the outcome of the request. |

|  |  |
| --- | --- |
| UC Name | UC\_8 Bank employee shall be able to approve or reject account creation requests.. |
| Summary | This use case involves the customer requesting a checkbook through the banking system. The bank processes the request and sends the checkbook to the customer. |
| Dependency | |  | | --- | |  |   The customer must have submitted a valid account creation request |
| Actors | Bank Employee, Customer |
| Preconditions | |  | | --- | | - The customer has a valid account.  - The customer is logged into the banking system. | |
| Description of the main sequence | |  | | --- | | 1. The customer submits an account creation request. |  |  |  | | --- | --- | |  | 2. The system processes the request and validates the provided details. |  |  |  | | --- | --- | |  | 3. The bank employee reviews the request and checks customer details. |  |  |  | | --- | --- | |  | 4. The bank employee approves or rejects the account creation request. |  |  |  | | --- | --- | |  | 5. The system updates the customer account status (approved or rejected) and notifies the customer. | |
| Description of the alternative sequence | -If the customer’s details are invalid, the system rejects the request and notifies the customer.  -if the customer’s details are valid and the bank employee approved the account creation, the system creates the account and the customer is notified, also a PIN is created.  - If the bank employee decides to reject the request, the system notifies the customer with a rejection notice |
| Non functional requirements | * Performance: The system should process the account creation requests within 5 seconds. * Security: The system must ensure secure handling of customer data during the request process, customer data should be stored securely and be kept private. * Scalability: The system should support simultaneous account requests for multiple customers without performance degradation. |
| Postconditions | * The customer request status is updated in the system. * The customer is notified of the outcome of the request. |

UC 9-12 Elvis Hamati

|  |  |
| --- | --- |
| UC Name | UC\_9: Bank Employee Views Client Accounts |
| Summary | This use case allows a bank employee to view all client accounts. The employee must be authorized to access account details, and the system ensures secure retrieval and display of information.. |
| Dependency | This use case depends on the authentication and authorization system to verify the employee's credentials. |
| Actors | * **Primary Actor:** Bank Employee |
| Preconditions | 1. The bank employee must be logged into the banking system. 2. The employee must have the necessary permissions to access client accounts. 3. The banking system must be online and operational. |
| Description of the main sequence | 1.The bank employee logs into the system.  2.The employee selects the option to view client accounts.  3.The employee asks for permission from the client to access his/her rights.  4.The system verifies the employee’s access rights.  5.The system retrieves and displays a list of all client accounts.  6.The employee can select a specific client account to view details.  7.The system displays the detailed information of the selected client account. |
| Description of the alternative sequence | 1. If the employee is not logged in, they are prompted to log in. 2. If the employee does not have sufficient permissions, an error message is displayed. 3. If the system is offline, an appropriate error message is shown.. |
| Non functional requirements | * **Performance:** The system should retrieve and display client accounts within 2 seconds. * **Security:** Only authorized employees should access client account details. * **Availability:** The system should be accessible 99.9% of the time. |
| Postconditions | * The bank employee successfully views the client accounts. * The system logs the employee’s access for auditing purposes. * No unauthorized data modifications occur during the process. |

|  |  |
| --- | --- |
| UC Name | UC\_10 Update Client Account Information |
| Summary | This use case allows a bank employee to update client account information. The employee must be authorized to modify account details, ensuring secure and accurate updates. |
| Dependency | This use case depends on the authentication and authorization system to verify the employee's credentials. It may also depend on the "View Client Accounts" use case to retrieve account details before updates.. |
| Actors | * **Primary Actor:** Bank Employee |
| Preconditions | 1. The bank employee must be logged into the banking system. 2. The employee must have the necessary permissions to update client account information. 3. The banking system must be online and operational. 4. The client account must exist in the system. |
| Description of the main sequence | 1. The bank employee logs into the system. 2. The employee selects the option to update client account information. 3. The system verifies the employee’s access rights. 4. The system retrieves and displays the client’s current account information. 5. The employee makes the necessary changes to the account details. 6. The system validates the updated information. 7. The employee confirms the changes. 8. The system updates the client account in the database. 9. The system logs the update for audit purposes. T 10. The system notifies the employee that the update was successful |
| Description of the alternative sequence | * If the employee is not logged in, they are prompted to log in. * If the employee does not have sufficient permissions, an error message is displayed. * If the system is offline, an appropriate error message is shown. * If the account does not exist, the system displays an error message. * If the updated information is invalid, the system prompts the employee to correct it. * If the update fails due to a system error, the employee is notified, and the update is logged as unsuccessful. |
| Non functional requirements | * **Performance:** The system should process account updates within 3 seconds. * **Security:** Only authorized employees should be able to modify client account details. * **Auditability:** Every update must be logged for tracking and compliance. * **Availability:** The system should be accessible 99.9% of the time |
| Postconditions | * The client account information is successfully updated. * The system logs the update for auditing purposes. * The employee receives confirmation of the successful update. * The updated information is available for future queries and transactions |

|  |  |
| --- | --- |
| UC Name | UC\_11: Bank Employee Checks Client Loan Status. |
| Summary | This use case allows a bank employee to check the loan status of a client. The employee must have appropriate authorization to access the loan details, ensuring secure and accurate retrieval of information.. |
| Dependency | |  | | --- | |  |   This use case depends on the authentication and authorization system to verify the employee's credentials. It may also depend on the "View Client Accounts" use case to retrieve client details before accessing the loan status |
| Actors | * **Primary Actor:** Bank Employee |
| Preconditions | |  | | --- | | 1. The bank employee must be logged into the banking system. 2. The employee must have the necessary permissions to access client loan details. 3. The banking system must be online and operational. 4. The client must have an active loan account in the system. | |
| Description of the main sequence | 1. The bank employee logs into the system. 2. The employee selects the option to check client loan status. 3. The system verifies the employee’s access rights. 4. The employee enters or selects the client’s account. 5. The system retrieves and displays the loan status, including outstanding balance, due dates, and payment history. 6. The employee reviews the loan details. 7. The system logs the access request for audit purposes.. |
| Description of the alternative sequence | 1. If the employee is not logged in, they are prompted to log in. 2. If the employee does not have sufficient permissions, an error message is displayed. 3. If the system is offline, an appropriate error message is shown. 4. If the client does not have a loan, the system displays a message stating no active loans. 5. If the loan details cannot be retrieved due to a system error, the employee is notified, and the access attempt is logged as unsuccessful.. |
| Non functional requirements | * **Performance:** The system should retrieve and display loan details within 3 seconds. * **Security:** Only authorized employees should be able to access client loan details. * **Auditability:** Every loan status check must be logged for tracking and compliance. * **Availability:** The system should be accessible 99.9% of the time. |
| Postconditions | * The employee successfully retrieves and views the client’s loan status. * The system logs the access request for auditing purposes. * No unauthorized modifications are made to the loan data. * The retrieved information is available for further processing or customer inquiries. |

|  |  |
| --- | --- |
| UC Name | UC\_12: Administrator Manages Employee Accounts. |
| Summary | This use case allows an administrator to manage employee accounts, including creating, updating, deactivating, and viewing employee information. The administrator must have the necessary permissions to perform these actions securely and efficiently. |
| Dependency | |  | | --- | |  |   This use case depends on the authentication and authorization system to verify the administrator's credentials. It may also depend on the "Employee Authentication" use case for login and access control |
| Actors | * **Primary Actor:** Administrator |
| Preconditions | |  | | --- | | 1. The administrator must be logged into the banking system. 2. The administrator must have the required permissions to manage employee accounts. 3. The banking system must be online and operational. | |
| Description of the main sequence | 1. The administrator logs into the system. 2. The administrator selects the option to manage employee accounts. 3. The system verifies the administrator’s access rights. 4. The administrator can perform the following actions:    * Create a new employee account by entering relevant details.    * Update an existing employee's information.    * Deactivate an employee account if necessary.    * View employee account details. 5. The system validates and saves the changes. 6. The system logs the changes for audit purposes. 7. The system notifies the administrator that the action was successful.  |  |  | | --- | --- | |  |  | |
| Description of the alternative sequence | 1. -If the administrator is not logged in, they are prompted to log in. 2. If the administrator does not have sufficient permissions, an error message is displayed. 3. If the system is offline, an appropriate error message is shown. 4. If the entered employee information is invalid, the system prompts the administrator to correct it 5. If an update or deactivation fails due to a system error, the administrator is notified, and the attempt is logged as unsuccessful |
| Non functional requirements | * **Performance:** The system should process employee account actions within 3 seconds. * **Security:** Only authorized administrators should be able to manage employee accounts. * **Auditability:** Every account management action must be logged for tracking and compliance.   **Availability:** The system should be accessible 99.9% of the time. |
| Postconditions | * The administrator successfully performs the requested account management action. * The system logs all changes for auditing purposes. * No unauthorized modifications occur. * The updated employee account information is available for future access and verification. |

# UC 13-16 Alonso Tiko

# Use Case: UC\_Admin\_13 – Monitor All Transactions

|  |  |
| --- | --- |
| Summary | Administrator monitors all transactions in the system for auditing, security, or analysis purposes. |
| Actors | Primary Actor: Administrator |
| Preconditions | Admin is logged in. Logging system is active. Admin panel is accessible. |
| Main Flow | 1. Admin logs in. 2. Admin navigates to “Transaction Monitor.” 3. Admin applies filters (date, type, user ID, etc.). 4. System displays filtered transactions. 5. Admin selects a transaction to view full details or flags it. |
| Alternative Flow | 1. Admin skips filters → system shows all transactions. 2. If result is too large → system prompts to refine filters. 3. Admin switches to real-time monitoring. |
| Postconditions | Transactions are viewed/exported/flagged. All monitoring actions are logged. |
| Dependencies | N/A |
| Non-Functional Requirements | N/A |

# Use Case: UC\_Admin\_14 – Freeze or Close Accounts

|  |  |
| --- | --- |
| Summary | Administrator freezes or closes customer accounts due to suspicious activity or based on a customer request. |
| Actors | Primary Actor: Administrator |
| Preconditions | Admin is authenticated. Account exists and is modifiable. |
| Main Flow | 1. Admin logs in. 2. Admin searches for a customer account. 3. Admin selects 'Freeze' or 'Close.' 4. System validates action (business rules). 5. Admin confirms action. 6. System updates account status and logs the event. |
| Alternative Flow | 1. If account is already closed → system shows error. 2. If under investigation → system disables freeze/close options. 3. Admin cancels before confirming. |
| Postconditions | Account is updated (frozen/closed). Customer notified. Action logged. |
| Dependencies | N/A |
| Non-Functional Requirements | N/A |

# Use Case: UC\_Admin\_15 – Generate Transaction Reports

|  |  |
| --- | --- |
| Summary | Administrator generates reports based on transaction data for analysis, auditing, or exporting purposes. |
| Actors | Primary Actor: Administrator |
| Preconditions | Admin is authenticated. Transaction data is available. Reporting module is functional. |
| Main Flow | 1. Admin logs in. 2. Navigates to 'Reports' section. 3. Selects report type (daily, monthly, by user...). 4. Applies filters. 5. System compiles report. 6. Admin previews/downloads it. |
| Alternative Flow | 1. No filters → default 30-day report. 2. Large report → system compresses. 3. Admin cancels generation. |
| Postconditions | Report generated/exported. Activity logged. |
| Dependencies | N/A |
| Non-Functional Requirements | N/A |

# Use Case: UC\_16 – Validate Account Details During Creation

|  |  |
| --- | --- |
| Summary | The system checks and verifies all the information provided by the customer when creating a new account. |
| Actors | Primary Actor: Customer |
| Preconditions | Customer has started the sign-up process. Form and validation services are online. |
| Main Flow | 1. Customer opens account creation form. 2. Enters personal details. 3. Uploads KYC documents. 4. Submits form. 5. System validates format (email, phone). 6. System checks for duplicates. 7. System verifies document quality. 8. System gives success/failure feedback. |
| Alternative Flow | 1. Missing fields → highlighted in red. 2. Duplicate email/ID → prompt for change. 3. Bad docs → request new upload. |
| Postconditions | On success → customer proceeds to next step. Logs saved for audit. |
| Dependencies | N/A |
| Non-Functional Requirements | N/A |

# UC 17-20 David Kapxhiu

|  |  |
| --- | --- |
| UC Name | UC\_17 Generate Monthly Account Statements |
| Summary | This use case describes how the system generates and provides customers with their monthly account statements. |
| Dependency | None |
| Actors | Primary Actor: Customer |
| Preconditions | - The customer must have an active bank account. - The system must have transaction data for the customer. |
| Main Sequence | 1. Customer logs into the banking system. 2. Customer navigates to the 'Statements' section. 3. System compiles customer transaction history for the month. 4. System formats the data into a structured statement. 5. System generates the monthly account statement. 6. System notifies the customer via email/SMS. 7. System displays and allows the customer to download the statement. |
| Alternative Sequence | 3a. If no transactions exist, the system generates a 'No Activity' statement. 6a. If notification fails, the system retries after a predefined interval. |
| Non-functional Requirements | The statement generation should not exceed 10 seconds. Statements should be available in PDF format. The system must ensure secure access. |
| Postconditions | The customer receives a monthly statement. The statement is securely stored for future reference. |

|  |  |
| --- | --- |
| UC Name | UC\_18 Apply Security Measures for Online Transactions |
| Summary | This use case describes how the system enforces security measures for online banking transactions. |
| Dependency | None |
| Actors | Primary Actor: Customer |
| Preconditions | - The customer must have an active online banking account. - The system must have security protocols in place. |
| Main Sequence | 1. Customer initiates an online transaction. 2. System verifies authentication via login credentials. 3. System prompts for multi-factor authentication (MFA). 4. Customer provides required authentication (OTP, biometrics, etc.). 5. System validates authentication data. 6. System securely processes the transaction. 7. System notifies the customer of the transaction status. |
| Alternative Sequence | 3a. If MFA verification fails, the system denies the transaction. 5a. If authentication fails, the system logs the attempt and alerts the customer. |
| Non-functional Requirements | Transactions must be encrypted end-to-end. Authentication should not exceed 5 seconds. The system must support multiple security mechanisms. |
| Postconditions | Online transactions are securely processed. Unauthorized access attempts are logged. |

|  |  |
| --- | --- |
| UC Name | UC\_19 Perform Automatic Currency Conversion for International Transactions |
| Summary | This use case describes how the system automatically converts currencies for international transactions. |
| Dependency | None |
| Actors | Primary Actor: Customer |
| Preconditions | - The customer must have an active bank account. - The transaction must involve a foreign currency. |
| Main Sequence | 1. Customer initiates an international transaction. 2. System detects currency mismatch. 3. System fetches latest exchange rate. 4. System converts transaction amount to recipient's currency. 5. System processes transaction and deducts amount. 6. System notifies customer of transaction details and conversion rate. |
| Alternative Sequence | 3a. If exchange rate is unavailable, system retries after a delay. 5a. If conversion fails, the system cancels transaction and notifies customer. |
| Non-functional Requirements | Exchange rate retrieval should not exceed 3 seconds. System must use real-time exchange rates from a reliable source. System must provide transparent fee information. |
| Postconditions | Transaction is completed with correct currency conversion. Customer is informed of transaction details. |

|  |  |
| --- | --- |
| UC Name | UC\_20 Block Suspicious Transactions |
| Summary | This use case describes how the system identifies and blocks suspicious transactions. |
| Dependency | None |
| Actors | Primary Actor: Customer Secondary Actor: Bank Security Team |
| Preconditions | - The customer must have an active bank account. - The system must have fraud detection mechanisms. |
| Main Sequence | 1. Customer initiates a transaction. 2. System analyzes the transaction against fraud detection rules. 3. If flagged as suspicious, system temporarily blocks it. 4. System notifies customer and requests verification. 5. Customer confirms or denies transaction. 6. If confirmed, system processes transaction. 7. If denied, system permanently blocks it and alerts security team. |
| Alternative Sequence | 3a. If system incorrectly flags a transaction, customer can appeal for review. 5a. If customer does not respond, transaction remains blocked. |
| Non-functional Requirements | Suspicious transaction detection should occur in real-time. System must use machine learning or rule-based detection. Customer verification requests should not exceed 5 seconds. |
| Postconditions | Fraudulent transactions are prevented. Legitimate transactions are processed after verification. |

UC 21-24 Erdi Meci

|  |  |
| --- | --- |
| Use Case Name | *UC\_ 21 Request Account Suspension* |
| Actors | *Customer* |
| Description | *This use case allows a customer to temporarily suspend their bank account. The customer may request a suspension due to fraud concerns, inactivity, or other personal reasons. The banking system processes the request and ensures the account is suspended until reactivated.* |
| Preconditions | *1. Customer must be logged into the system. 2. Customer must have an active account. 3. The account must not already be suspended.* |
| Main Sequence | *1. The customer navigates to the "Account Suspension" section. 2. The customer selects the suspension reason and confirms the request. 3. The system verifies the account status and eligibility for suspension. 4. The system processes the suspension and sends a confirmation notification. 5. The account is marked as suspended in the system. 6. The customer receives confirmation of successful suspension.* |
| Alternative Sequence | *1a. If the account is already suspended, the system notifies the customer. 2a. If the request fails due to technical issues, the system prompts the customer to retry later.* |
| Postconditions | *1. The account is successfully suspended. 2. The customer cannot perform any transactions until reactivation.* |

## 

|  |  |
| --- | --- |
| Use Case Name | *UC\_22 Chatbot Assistance for FAQs* |
| Actors | *Customer, Chatbot System* |
| Description | *This use case enables customers to receive instant assistance through an AI-powered chatbot. The chatbot answers frequently asked questions, provides guidance on banking procedures, and redirects customers to human agents if needed.* |
| Preconditions | *1. The chatbot feature must be enabled. 2. The customer must have access to online banking or the mobile app.* |
| Main Sequence | *1. The customer accesses the chatbot from the support section. 2. The customer types a query related to banking services. 3. The chatbot searches for relevant responses and provides an answer. 4. If the response is insufficient, the chatbot offers further options. 5. If necessary, the chatbot transfers the customer to a live support agent.* |
| Alternative Sequence | *1a. If the chatbot does not recognize the question, it requests clarification. 2a. If the chatbot fails to provide a satisfactory answer, it suggests speaking with a human agent.* |
| Postconditions | *1. The customer receives an answer to their query. 2. If the chatbot cannot resolve the issue, a human agent takes over.* |

|  |  |
| --- | --- |
| Use Case Name | *UC\_23 Auto-Logout After Inactivity* |
| Actors | *Customer* |
| Description | *This use case describes the system’s ability to automatically log out users after a period of inactivity. This enhances security by preventing unauthorized access if the user forgets to log out manually.* |
| Preconditions | *1. The customer must be logged into the system. 2. A session timeout period must be configured by the bank.* |
| Main Sequence | *1. The customer logs into their account. 2. The system tracks user activity and detects inactivity. 3. If no interaction occurs for a predefined period, the system warns the user. 4. If the user remains inactive, the system logs them out. 5. The customer must log in again to continue banking.* |
| Alternative Sequence | *1a. If the user interacts with the system before timeout, the session remains active.* |
| Postconditions | *1. The customer is logged out due to inactivity. 2. The system ensures security by preventing unauthorized access.* |

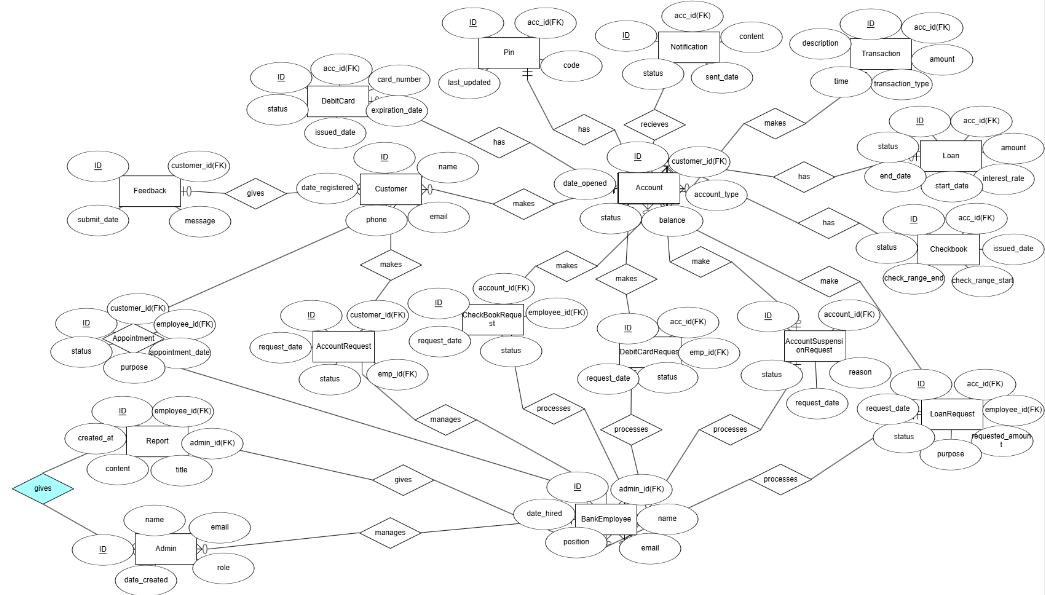
|  |  |
| --- | --- |
| Use Case Name | *UC\_24 Reset Customer Login Credentials* |
| Actors | * *Bank Employee* |
| Description | *This use case describes the process where a bank employee resets a customer’s login credentials upon request. The system generates new credentials and ensures that only authorized personnel can perform the reset.* |
| Preconditions | * *The customer must verify their identity through a security process.* * *The bank employee must have administrative privileges.* * *The customer must provide a valid reason for the reset request.* |
| Main Sequence | *1. The customer contacts customer support to request a credential reset. 2. The bank employee verifies the customer’s identity using security questions or other verification methods. 3. If verification is successful, the system generates a temporary password. 4. The employee provides the new credentials securely. 5. The customer logs in using the temporary password and sets a new password.* |
| Alternative Sequence | * *1a. If verification fails, the employee requests additional identification.* * *2a. If the system encounters an error, the employee informs the customer to retry later.* |
| Postconditions | * *The customer successfully receives new login credentials.* * *The system logs the reset request for security purposes.* |

UC 25-28 Fabio Sulo

5. Diagrams

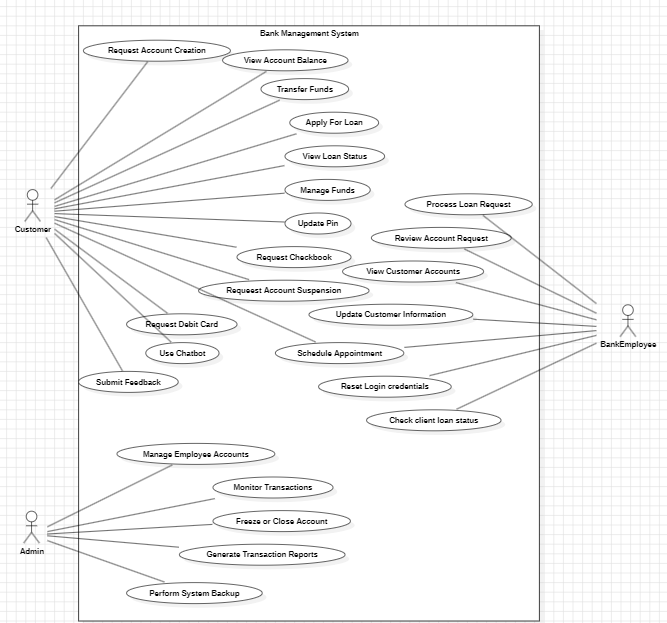
In this section you are going to place all of the diagrams that you build throughout to the course, in following with the slides presented throughout the weeks.

5.1 ER Diagram



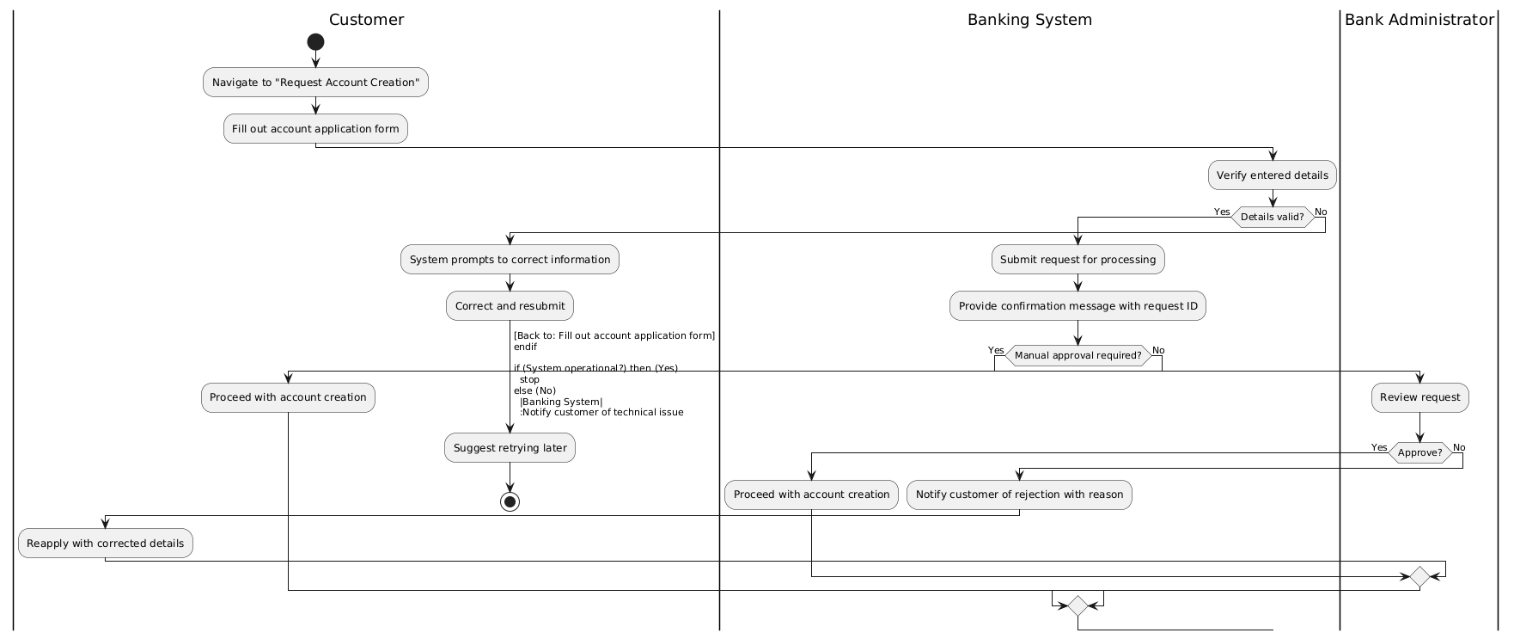
.  
  
5.2 Use Case Diagram (general)

(Florjon Allkaj)

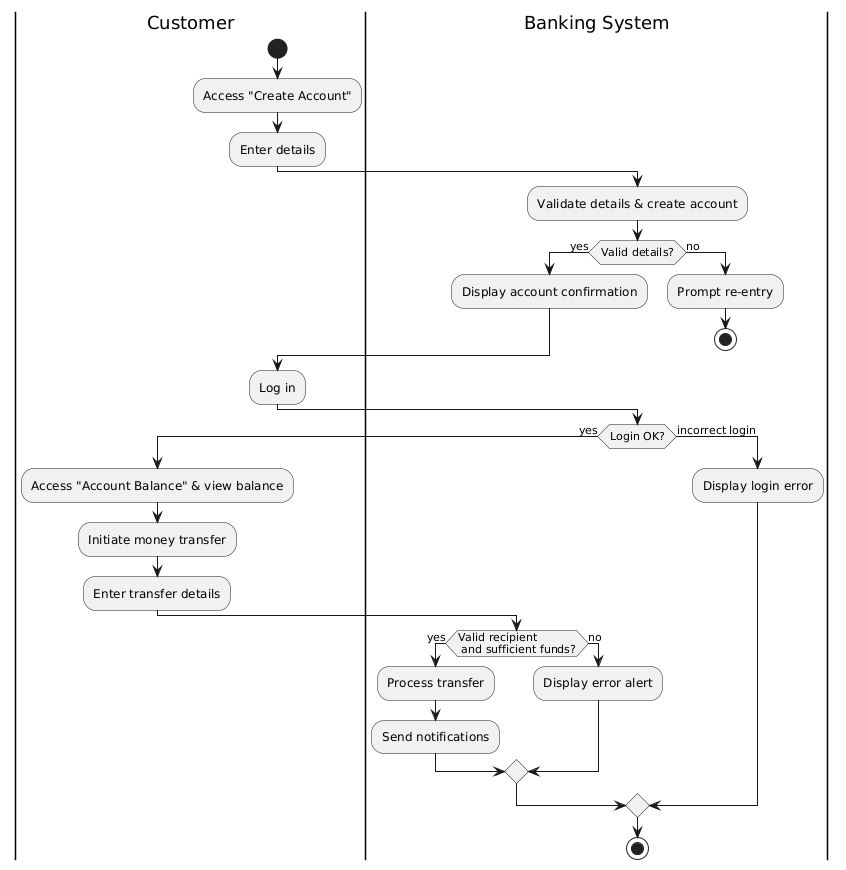


5.3 Activity Diagram

UC1(Enklajd Hodo use cases 1-4)



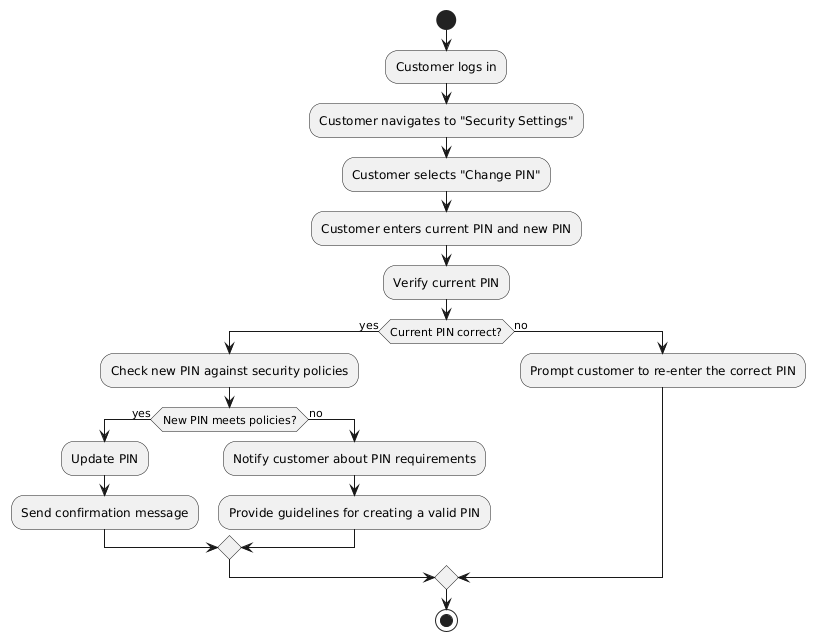
UC2



UC3



UC4



(Florjon Allkaj Use cases 5-8)

Use case 5

A diagram of a process flow

AI-generated content may be incorrect.

Use case 6

A diagram of a process

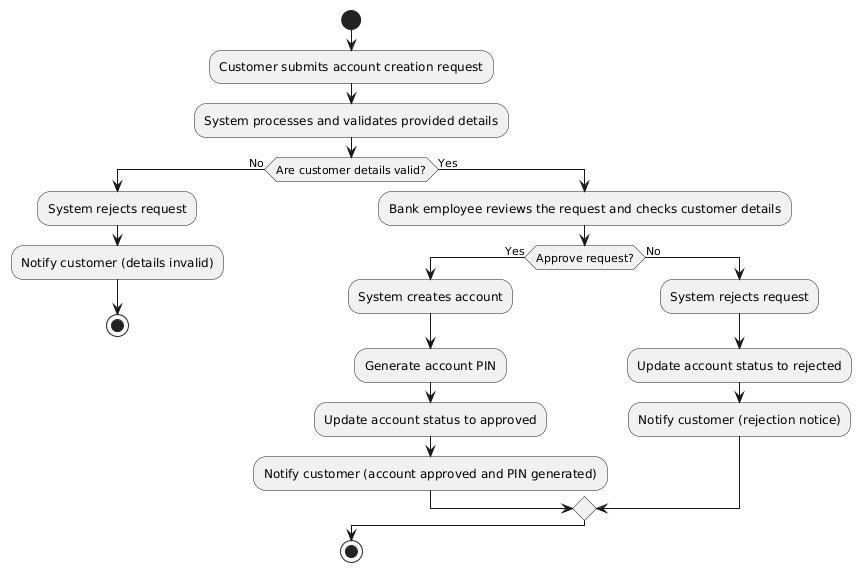
AI-generated content may be incorrect.

Use case 7

A diagram of a customer survey

AI-generated content may be incorrect.

Use case 8



(Elvis Hamati requirements 9-12

Use case 9

A flowchart with text and images

AI-generated content may be incorrect.

Use Case 10

A flowchart of information

AI-generated content may be incorrect.

Use Case 11

A flowchart of a bank

AI-generated content may be incorrect.

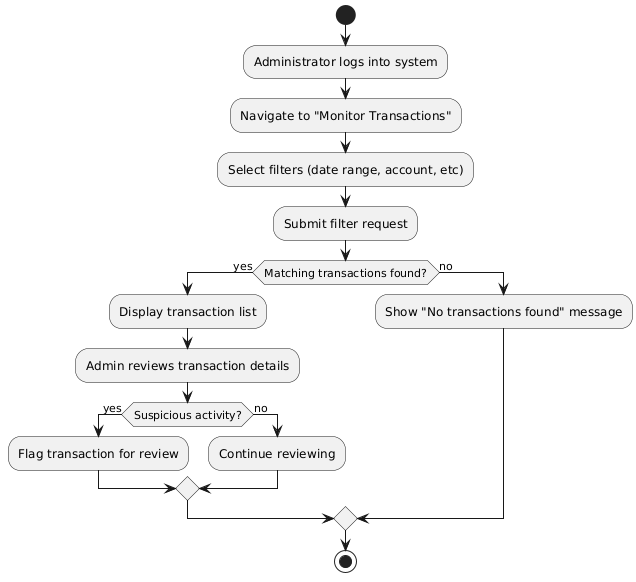
Use Case 12

A diagram of a company

AI-generated content may be incorrect.

(Alonso Tiko Use Cases 13-16)

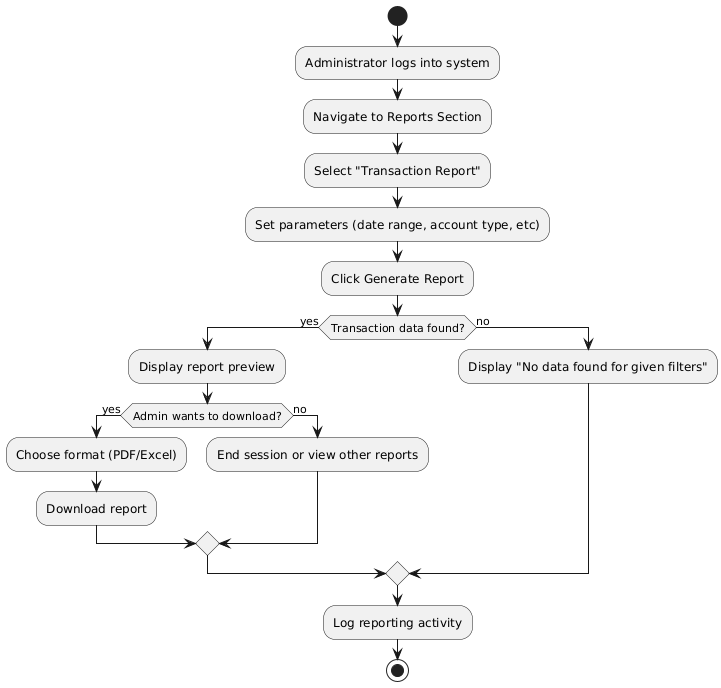
Requirement number 13: “Administrator shall be able to monitor all transactions.”

****

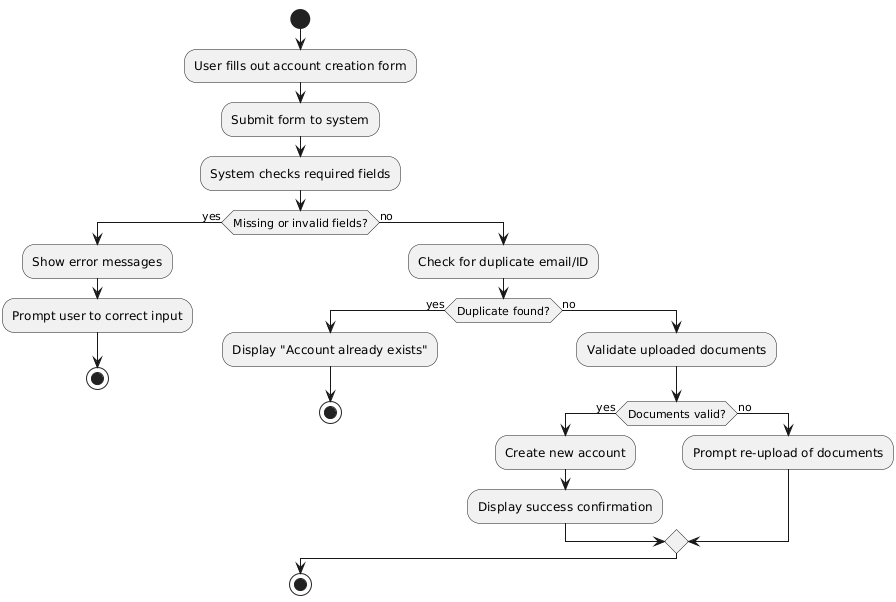
Requirement number 14: “Administrator shall be able to freeze or close accounts.”

****

Requirement number 15: “Administrator shall be able to generate reports on transactions.”

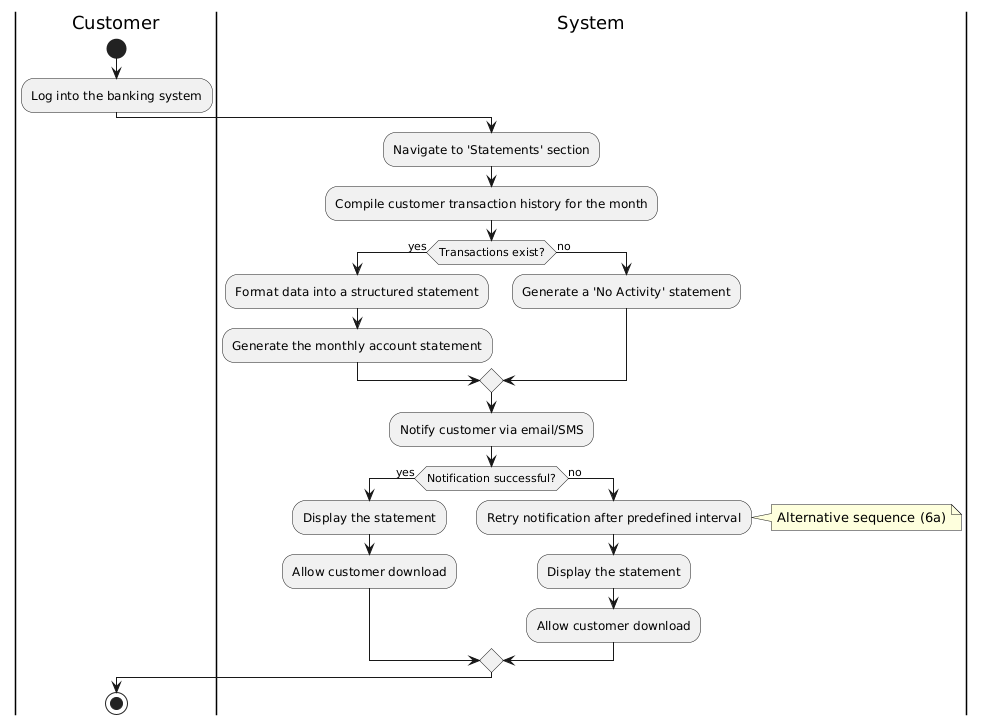
****

Requirement number 16: “System shall be able to validate account details during creation.”

****

(David Kapxhiu 17-20)

Use Case 17



Use Case 18

A diagram of a system

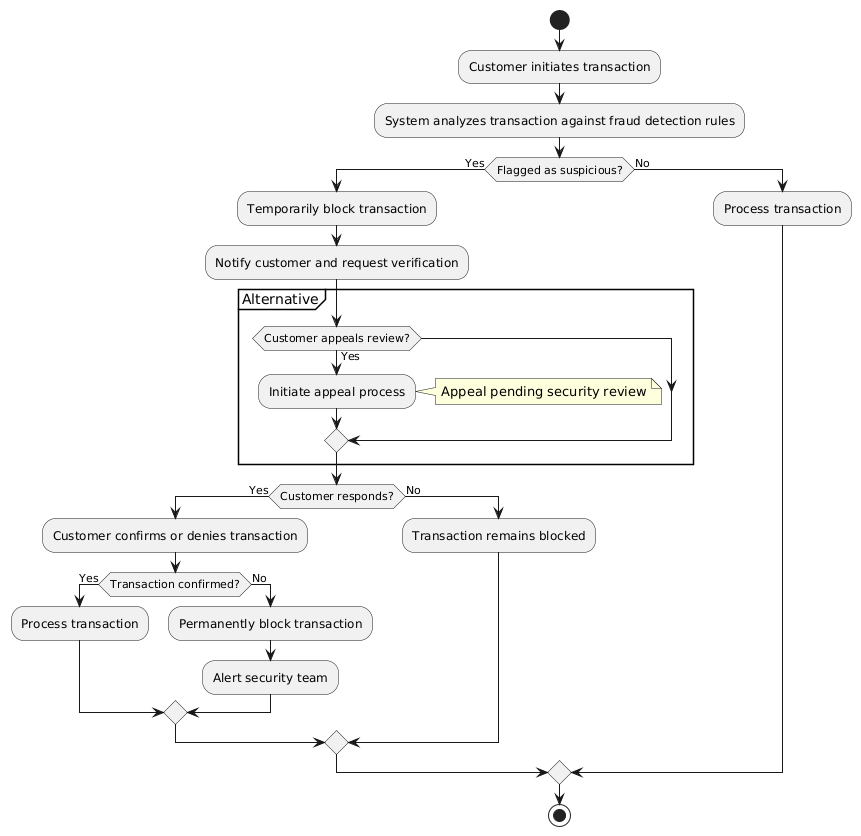
AI-generated content may be incorrect.

Use Case 19

A diagram of a system

AI-generated content may be incorrect.

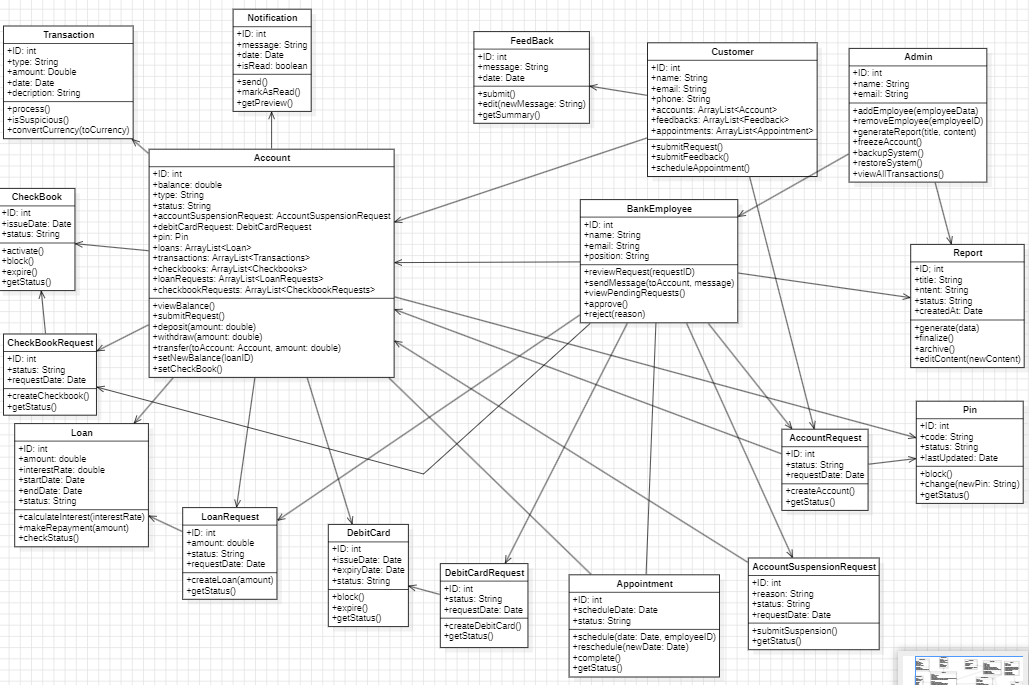
Use Case 20



(Erdi Meci Use Case 21-24)

(Fabio Sulo Use Case 25-28)

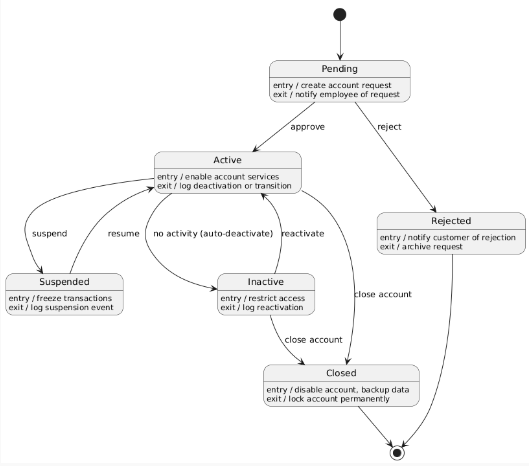
5.4. Class diagram.



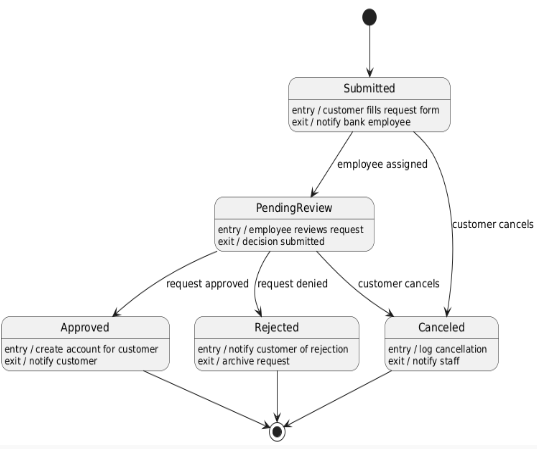
5.5 State diagram

(Florjon Allkaj)

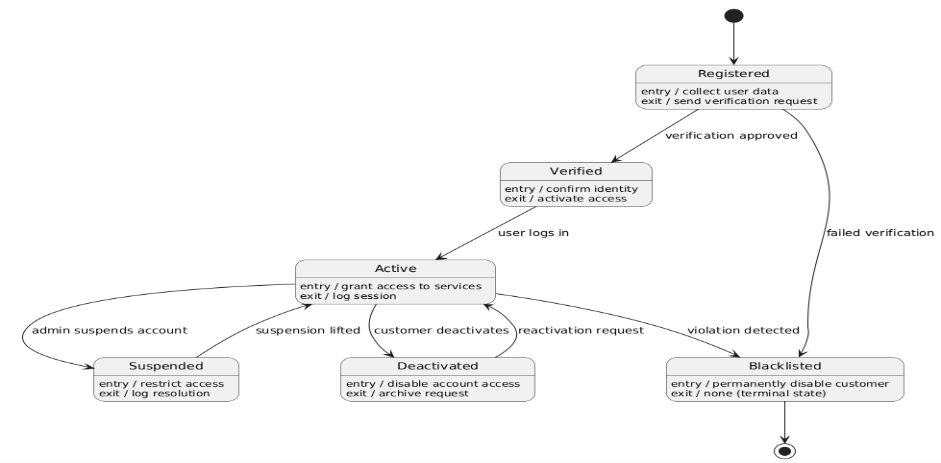
Account



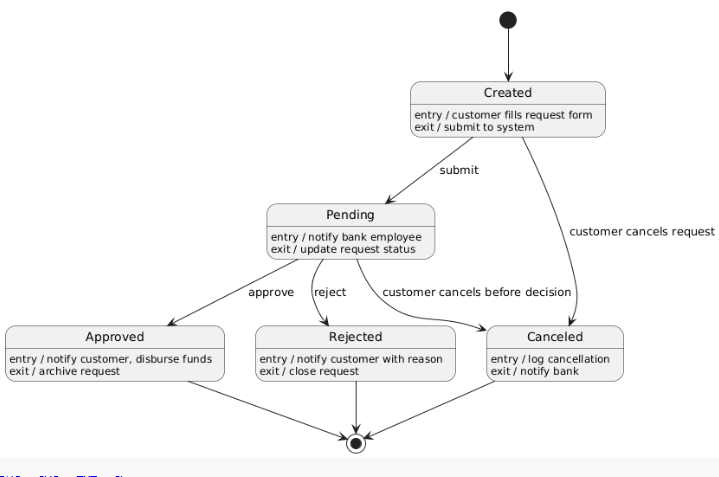
AccountRequest



Customer



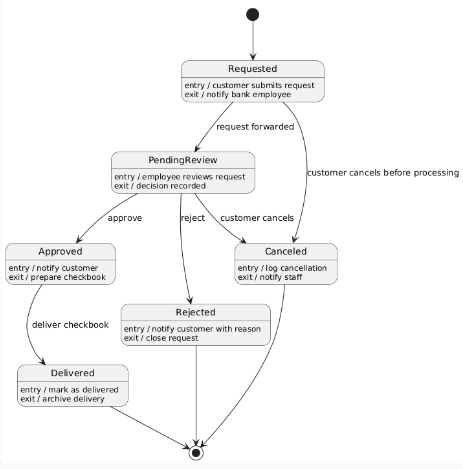
LoanRequest



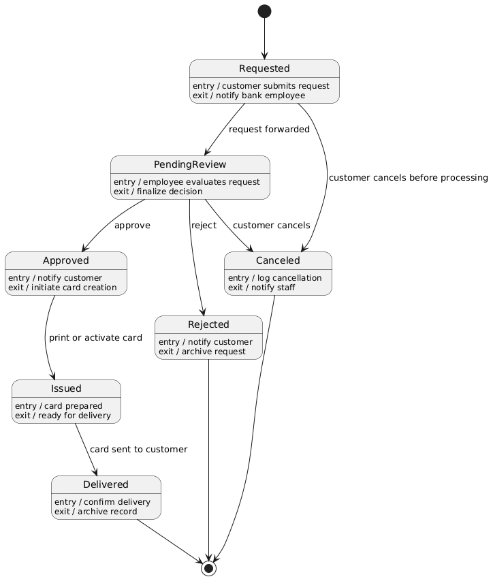
Appointment



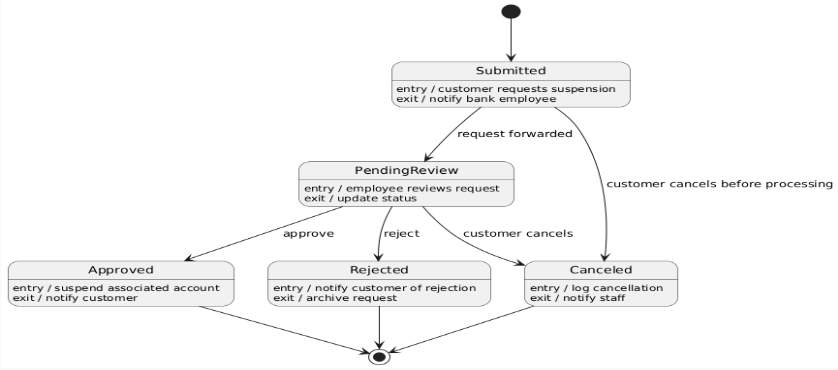
CheckBookRequest



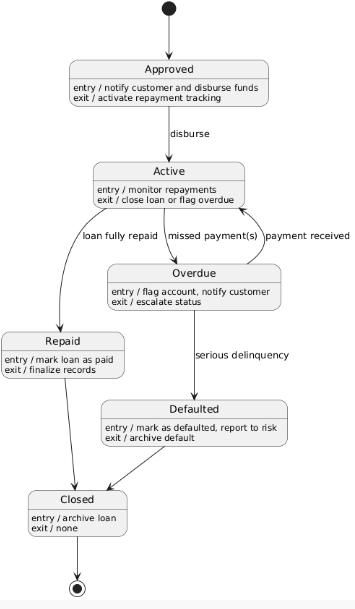
DebitCardRequest



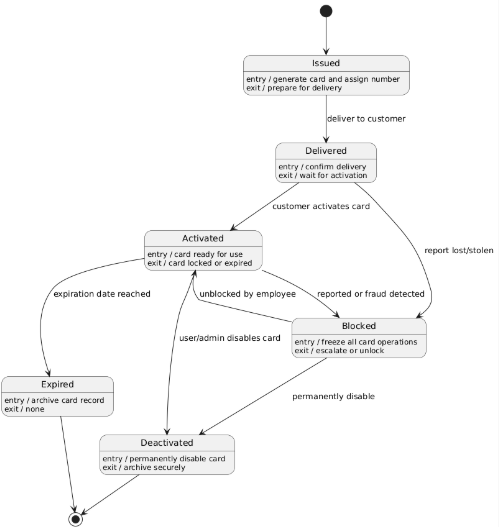
AccountSuspensionRequest



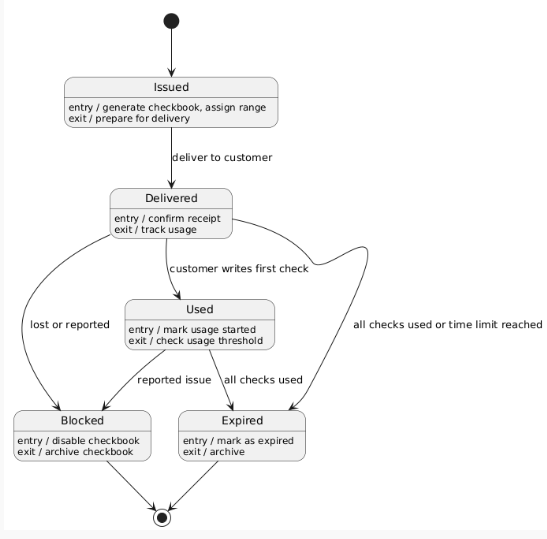
Loan



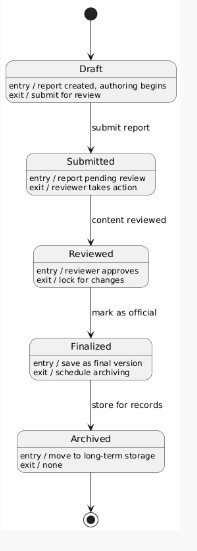
DebitCard



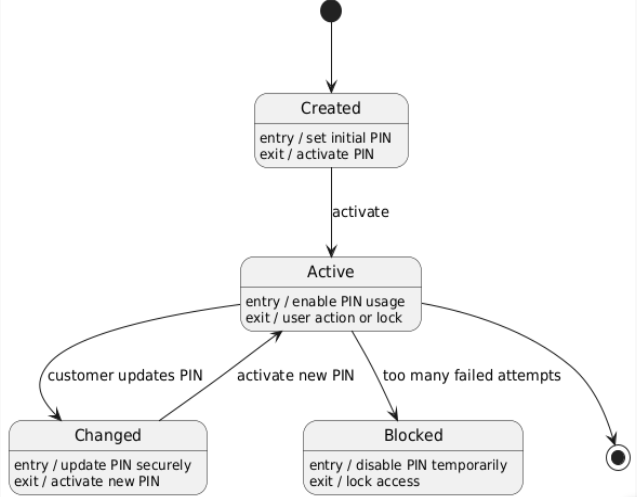
CheckBook



Report



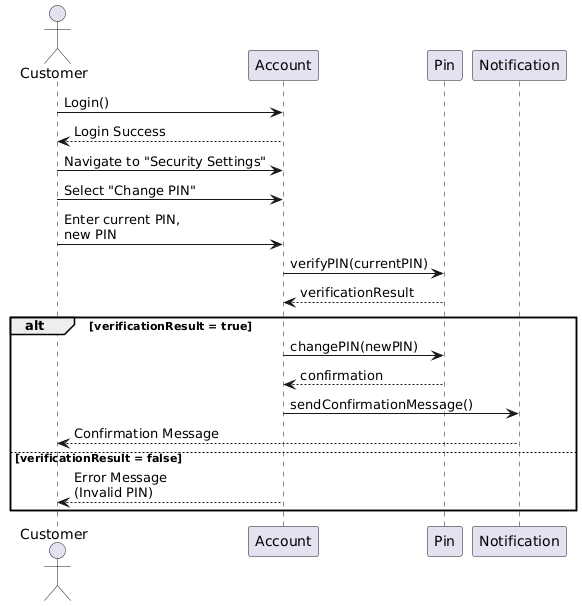
Pin

****

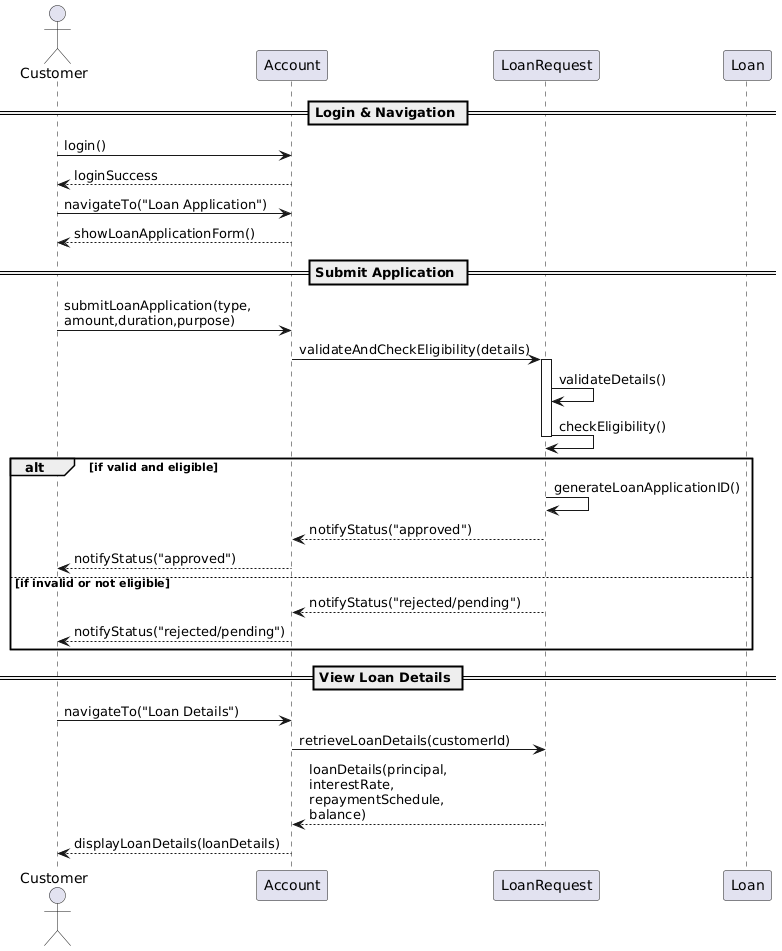
5.6 Sequence diagram.

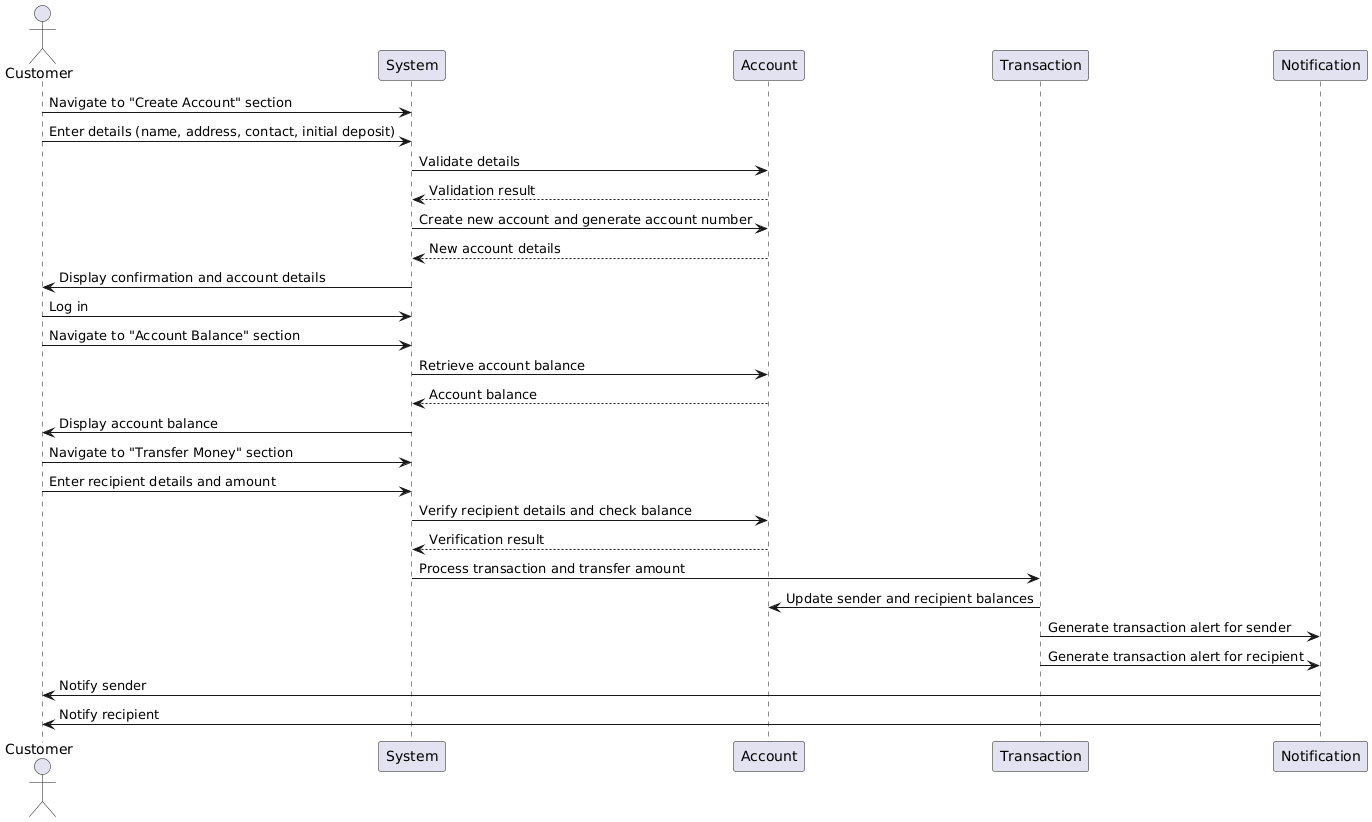
Enklajd Hodo (UC 1-4)

UC 1

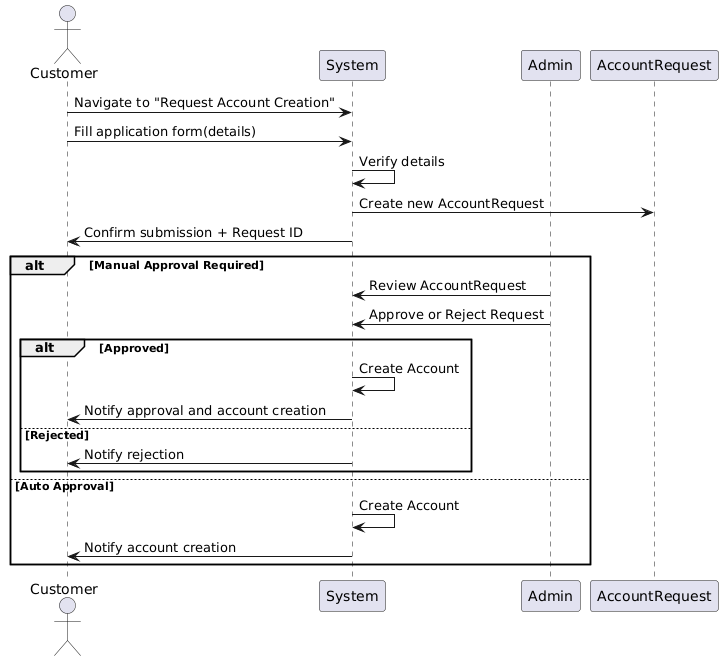


UC 2



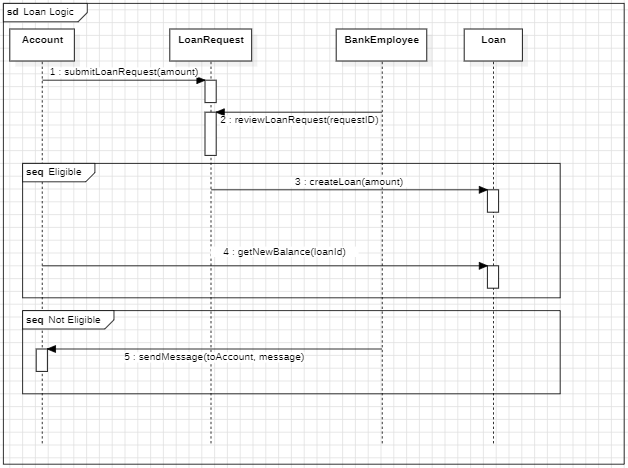
UC 3

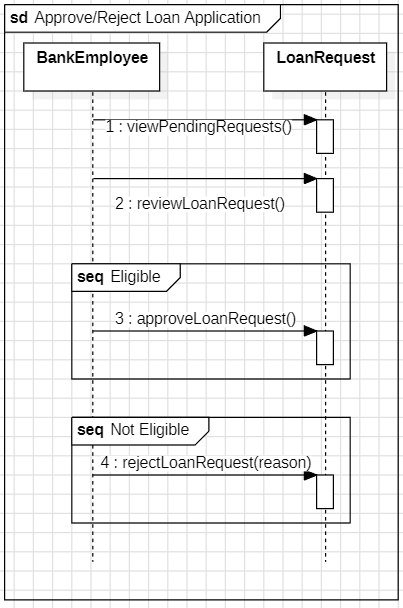
UC 4



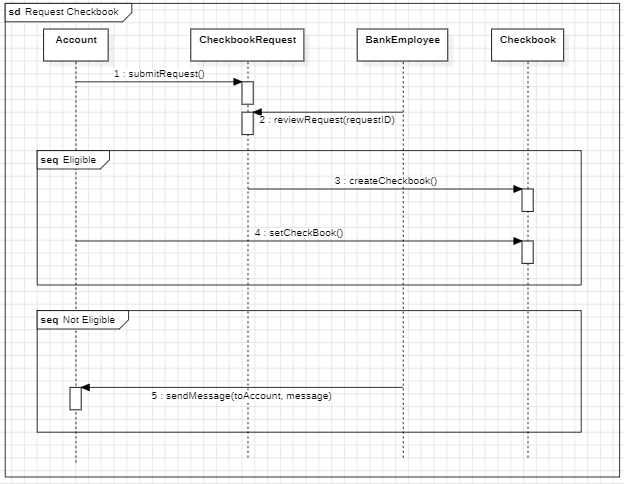
Florjon Allkaj(UC 5-8)

UC-5 Bank Employee shall handle Loan Logic

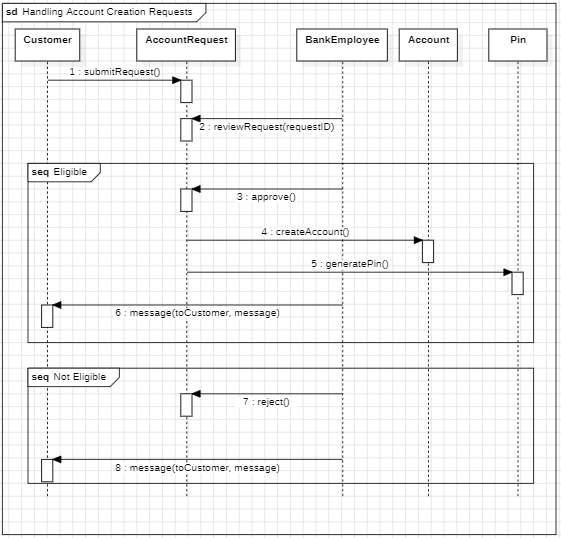


UC-6 Approve/Reject Loan request  


UC-7 Request a Checkbook



UC-8 Bank Employee Handling Account Requests



5.7. Collaboration diagram

Collaboration Diagrams

| Florjon Allkaj |

1. CD\_CreateBankAccount

2. CD\_TransferMoneyWithAlert

3. CD\_SubmitFeedback |

| Enklajd |

4. CD\_ApplyForLoan

5. CD\_RequestDebitCard

6. CD\_ManageCashAndPIN |

| Elvis Hamati |

7. CD\_RequestCheckbook

8. CD\_RequestAccountSuspension

9. CD\_ScheduleAppointment |

| David Kapxhiu |

10. CD\_ApproveAccountRequest

11. CD\_UpdateAccountInfo

12. CD\_ResetCredentials |

| Erdi Meci |

13. CD\_ReviewLoanRequest

14. CD\_GenerateReport |

| Alonso Tiko |

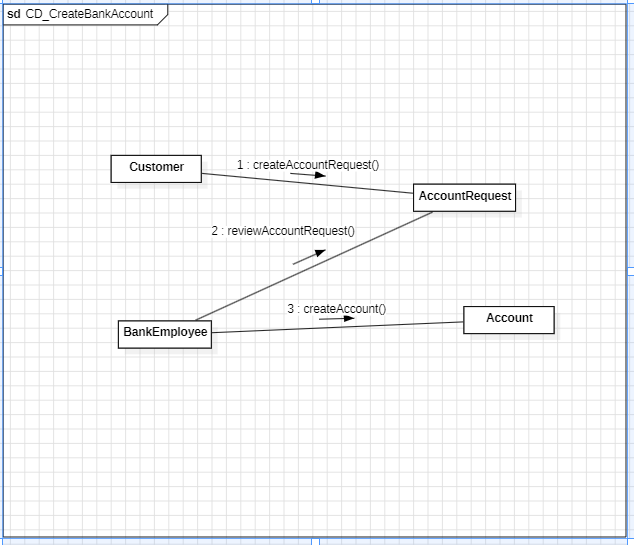
15. CD\_FreezeOrCloseAccount

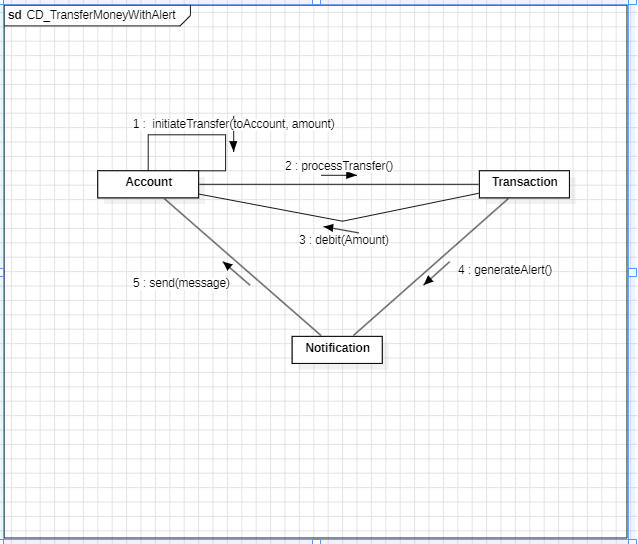
16. CD\_ManageEmployees |

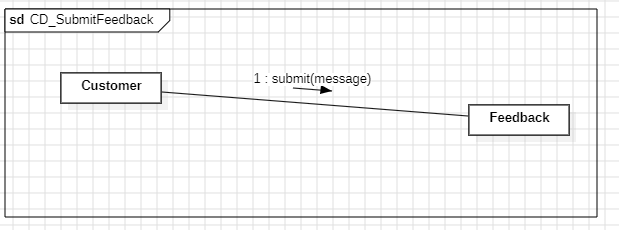
| Fabian Sulo |

17. CD\_BlockSuspiciousTransaction |

Create Bank Account, Transfer money with alert, Submit Feedback (Florion Allkaj)







Apply For Loan, Request Debit Card, Manage Cash And Pin (Enklajd Hodo)

CD\_RequestDebitCard

3:expire

**Customer**

3: blockDebitCard

**Notification**

1: getStatus

**BankEmployee**

4: activateDebitCard

**DebitCard**

CD\_ApplyForLoan

4:confirmRequest

5: confirmLoan

**BankEmployee**

1: createRequest

2:submitRequest

3:review

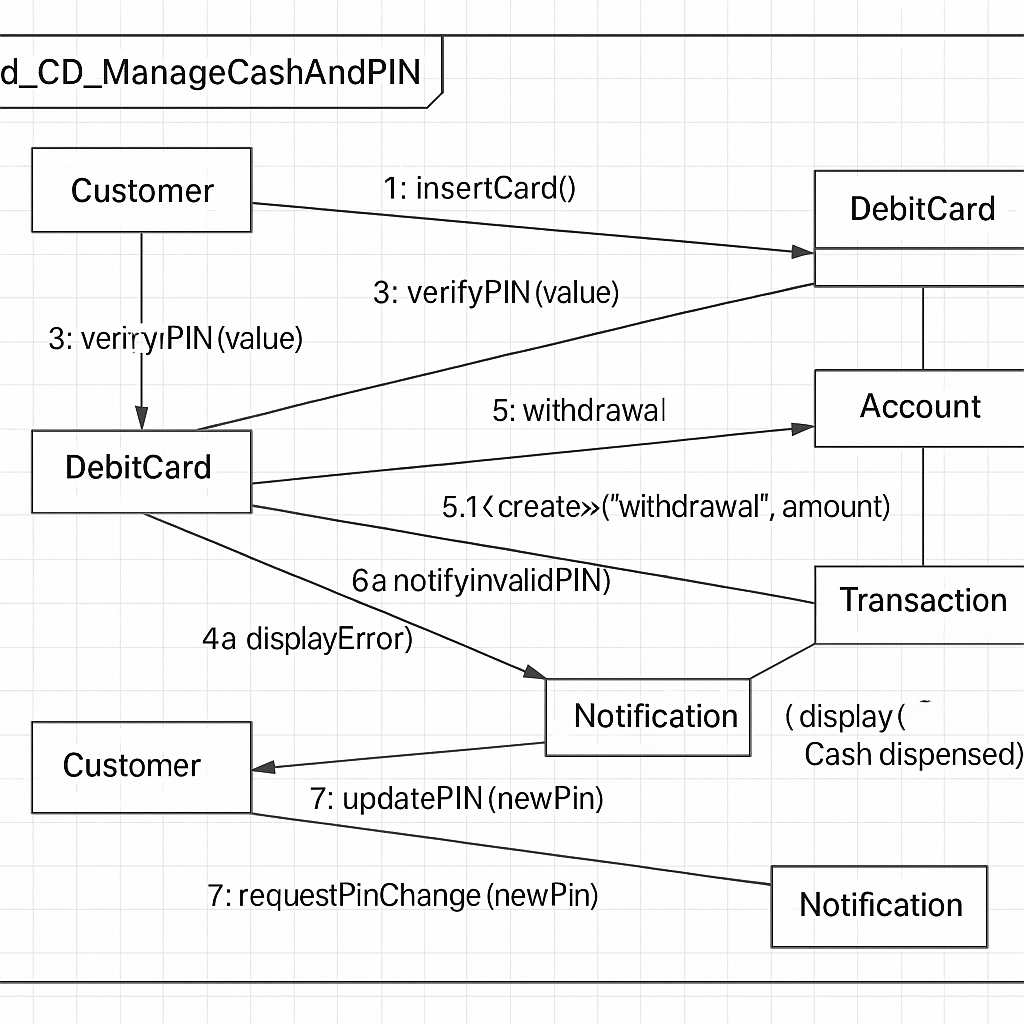
**Loan**

**Account**

**RequestApplyForLoan**

**Customer**

CD\_ManageCashAndPIN

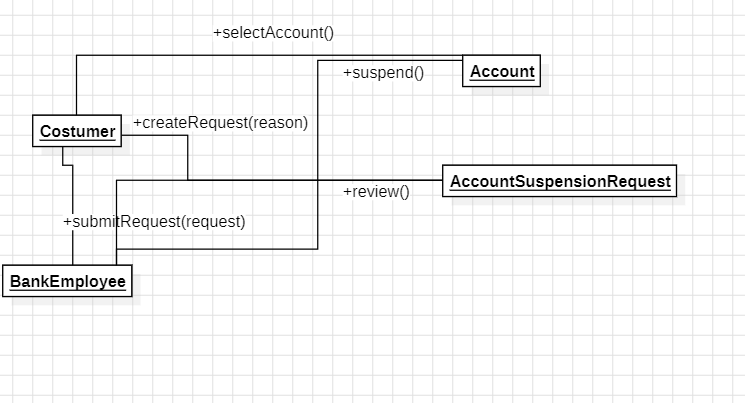


Elvis Hamati

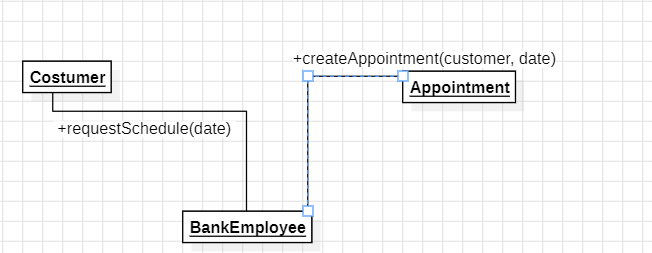
CD\_RequestCheckbook



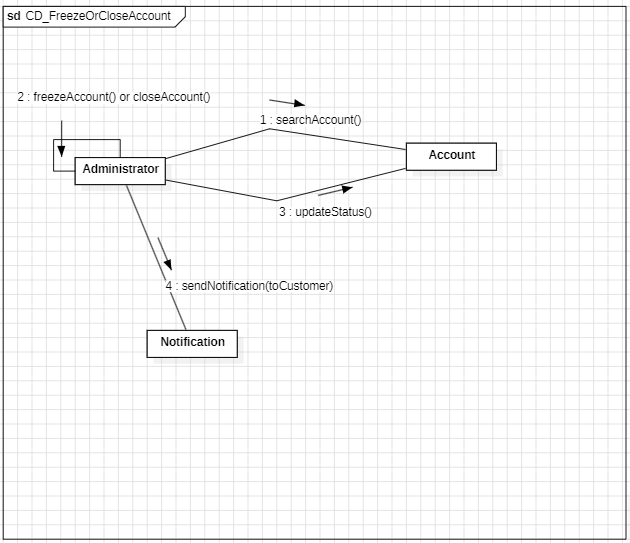
CD\_RequestAccountSuspension

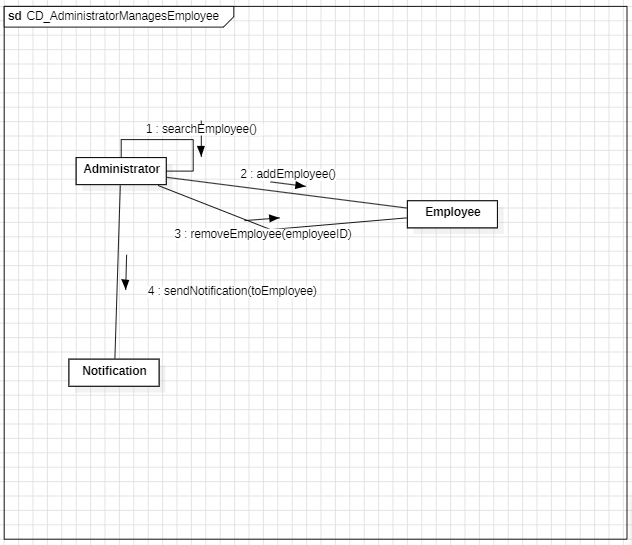


CD\_ScheduleAppointment



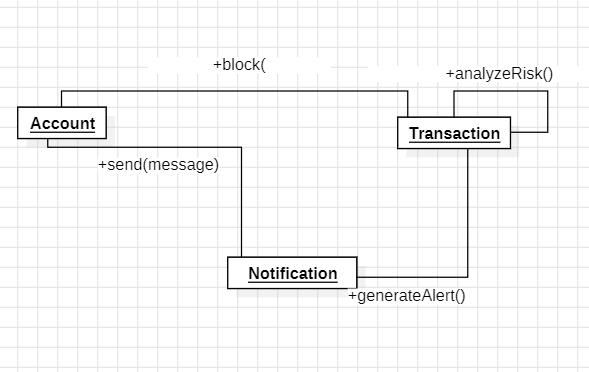
Freeze or close account, Block suspicios transaction (Alonso Tiko)





Fabio Sulo

CD\_BlockSuspiciousTransaction



6. Design Patterns

Alonso Tiko (All design patterns)

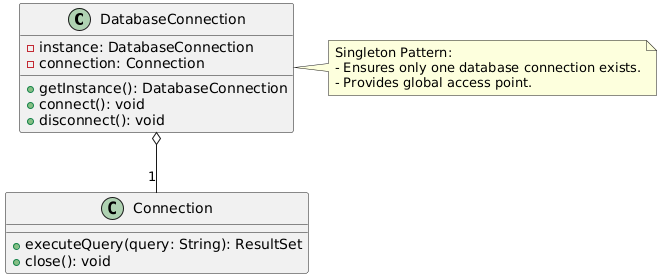
## Singleton Pattern

**Where It Fits in Our Real System:**

Based on our class diagram, a perfect fit for Singleton is the **DatabaseConnection** or **SystemController** — if either is responsible for central access/control.

**Justification:**

* Ensures a single, shared database connection for:
  + Customer
  + Transaction
  + Account
  + Loan
* Centralizes control and resource usage.



**Explanation:**

* DatabaseConnection has a **private instance.**
* getInstance() gives controlled access.
* Other system components call this to access or close the connection.

1. **Factory Method Pattern**

**Where It Fits in our System:**

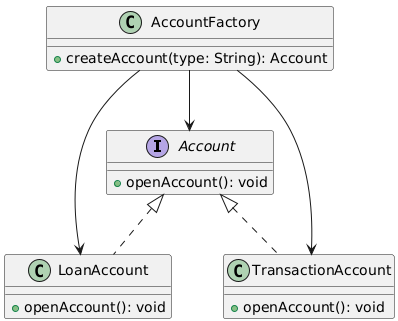
The best place to apply this pattern is in the **Account Creation Process**.

From our class diagram, we have classes like:

* Account
* Loan.
* Transaction

**Justification:**

* Creating different types of accounts (e.g. Loan, Transaction) depending on user or system input.
* Factory method keeps the code **open for extension**, but **closed for modification.**
* Simplifies future maintenance if new account types are added.



**Explanation:**

* Account is the interface.
* LoanAccount and TransactionAccount implement it.
* AccountFactory creates instances of the correct account type at runtime.
* This logic would live inside a real class like AccountService in your system.

1. **Observer Pattern**

**Where It Fits in our System:**

A perfect spot for this is in **Transaction Monitoring**.

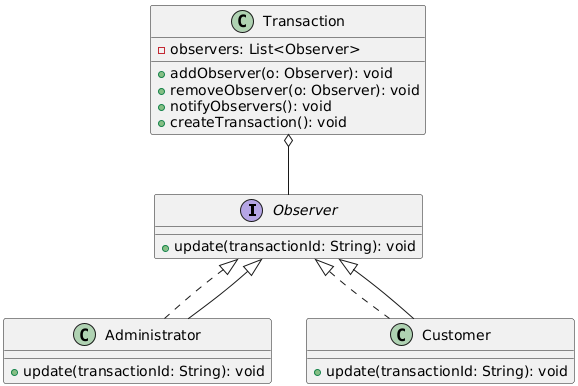
From your class diagram, you have:

* Transaction.
* Administrator.
* Notification (used logically).

**Justification:**

* When a new transaction occurs, the system should **notify all observers**, like:
  + The **Administrator** (to review flagged transactions).
  + Possibly a **Customer** (to view it in their account).

This supports real-time monitoring and decouples the logic.



### Explanation:

* Transaction holds a list of observers.
* Administrator and Customer are observers.
* When a transaction is created or updated, all observers are notified.

1. **Facade Pattern**

**Where It Fits in our System:**

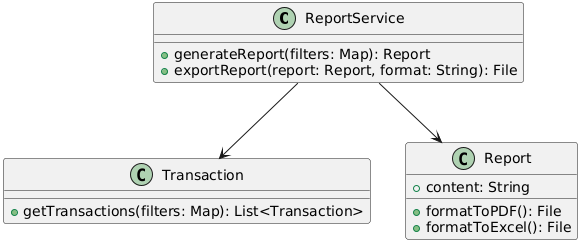
The best use of the **Facade Pattern** in your system is for **Report Generation**.

From your class diagram, we have:

* Report.
* Transaction.

**Justification:**

* Report generation involves multiple steps:
  + Fetching transactions.
  + Formatting data.
  + Creating the report.
  + Exporting it.
* A facade like ReportService hides all that complexity behind a simple interface.



### Explanation:

* ReportService acts as the **facade**
* It fetches transaction data, passes it to Report for formatting/exporting
* The system user (Admin) only interacts with ReportService

1. **Decorator Pattern**

**Where It Fits in Your System:**

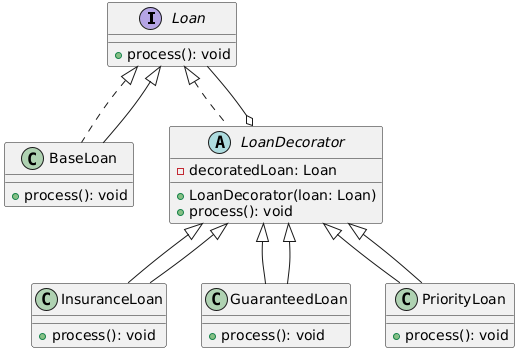
The **Decorator Pattern** fits perfectly in your **Loan Application** flow.

From your class diagram, we have:

* Loan.
* LoanRequest (or assumed based on context).
* Optional features like **insurance**, **guarantee**, **priority processing** can be modeled as decorators.

**Justification:**

* Instead of having tons of subclasses like LoanWithInsuranceAndGuarantee, we can:
  + Start with a base Loan.
  + Add functionality dynamically with decorators.
* This makes it **flexible**, **scalable**, and **easy to maintain.**

****

### Explanation:

* BaseLoan is the core loan functionality.
* LoanDecorator wraps any Loan instance.
* Decorators like InsuranceLoan, GuaranteedLoan, and PriorityLoan add functionality dynamically.

## **7. Appendix**

### **Organization of Requirements**

In this Bank Management System (BMS) project, the requirements were organized primarily **by user class** and **by feature**, allowing for clear separation of system functionalities based on the actors involved and the services provided by the system.

#### **By User Class**

The system is used by multiple user categories, each of whom interacts with specific sets of features:

* **Customers**: Access services like account creation, fund transfers, loan applications, security settings, account suspension, etc.
* **Bank Employees**: Handle account approvals, loan processing, checkbook issuance, customer support, and account updates.
* **Administrators**: Perform system-level operations like monitoring transactions, generating reports, managing user roles, enforcing security, and performing audits.

This user-based classification ensures that each stakeholder's requirements and permissions are well scoped and individually tested.

#### **By Feature**

Each major functionality of the system was treated as a **feature**, with a dedicated set of use cases:

* **Account Management**: Includes creation, viewing, updating, suspension, and closing of accounts.
* **Transaction Handling**: Covers internal/external fund transfers, real-time alerts, transaction history, and fraud detection.
* **Loan Services**: Includes applying for loans, loan eligibility checks, repayment schedule management, and employee loan approval workflows.
* **Customer Support Services**: Features chatbot FAQs, appointment scheduling, checkbook and debit card requests, feedback forms, etc.
* **Security and Access**: Includes PIN management, auto-logout, two-factor authentication, encryption, and credential resets.
* **Reporting and Audit Logs**: Admin features for monitoring user activity, generating system and financial reports, and ensuring compliance.

This organization allows for modular development and testing, ensuring that each functional area can evolve independently without affecting unrelated parts of the system.

### **Methodological Notes**

* The **Agile methodology** was followed to ensure continuous feedback, iterative development, and adaptive planning.
* Requirements were elicited, validated, and tracked using collaboration tools and stakeholder meetings.
* Diagrams such as **Use Case**, **Class**, **Activity**, **Sequence**, **State**, and **Collaboration** were used extensively to visualize the system logic.

### ****Design Pattern Justification****

Various **design patterns** were used to support maintainability, scalability, and code reuse:

* **Singleton Pattern**: Centralized access for components like database connections.
* **Factory Pattern**: For creating different account or loan types.
* **Observer Pattern**: To notify users of real-time events (e.g., transactions).
* **Facade Pattern**: For simplifying report generation workflows.
* **Decorator Pattern**: To flexibly extend loan features (e.g., insurance, guarantees).

### ****Compliance & Standards****

The system aligns with:

* **GDPR** for data privacy.
* **PCI DSS** for secure handling of financial information.
* National regulations for banking audits and reporting.
* Interoperability standards like **SWIFT/SEPA** for cross-bank transactions.