The Bank Application Documentation (final version)

Borkowski Maciej
Górski Szymon
Harbatsenka Pavel
Komlichenko Illia
Mankirov Erdni

Software Engineering 1,
1 February 2022

Contents

1 Introductory description

The goal of the project is to design a banking application which will focus on providing solutions for both customers, their workers and escorts. The documentation provides description of classes of users that can interact with the application as well their functionalities and dependencies.

The application provides distinct modules for different Actors performing actions within the bank. People using the solution are divided into Clients, Escorts, Employees and the Bank CEO classes. They are provided functionalities via an Internet website with by Graphical User Interface (GUI) adjusted to their needs and privileges. Clients use services which allow them to run accounts as well create deposits and take loans. Escorts are contracted by the bank to convoy money from one place to another, being provided with the interface to report status of their tasks. Employees, who work in bank agencies, have privileges to verify and approve Clients' actions, as a part of their job. Finally, the CEO of the bank can take most crucial decisions about functioning of the system, including managing his or her Employees. The Server class is responsible for handling all actions taken by other Actors, controlling cash flow, as well as running some maintenance events.

2 Components

The application will have three main components of a different size, each of them interacting with Actors via application programming interface (API). The unification of the structure allows for an easy communication between components. Description of components:

2.1 API

API will be introduced to unify the process of retrieving requested data. Its main goal will be to create a channel of communication between the Server and other Actors. It will make the calls used within the system easily translatable to supporting elements of the application (like frameworks).

2.2 Server Application

The Server is an Actor providing back-end functionality of the application. It will process requests sent via other two parts of the application and validate them against data stored in its local database(s). The Server will apply necessary locks and validate transactions. The results of computation will be then sent back to the proper components. The Server will work upon an operating system (OS) with a connection to the Internet to maintain the service for other parts of the application as well as store data about other Actors and their Objects.

2.3 Administrative Panel

The Panel will be an interface provided via the framework used to build the application which will bring possibilities to check the settings and contents of the Server and modify it if necessary. It will also allow to change the inner parameters of the work of the server if the Bank CEO wishes to do so. This component will not be covered with an additional GUI elements.

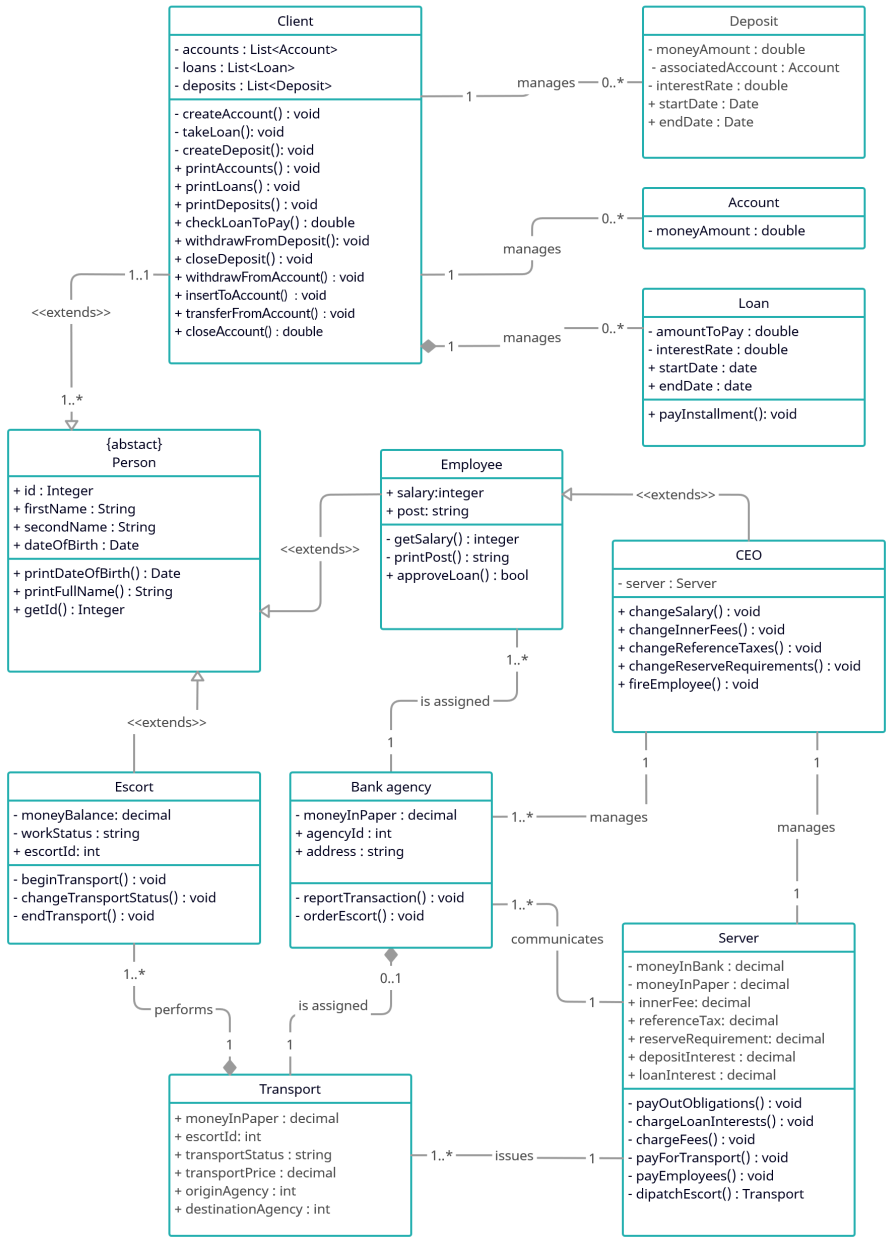
2.4 Web Application

The web application will provide a graphical user interface (GUI) with an access to a user's account upon verification with credentials via an internet browser. The GUI will allow to manage the account in an easy and intuitive way. It will offer a different set of activities for the Client, the Escort and the Bank Worker, dedicated for the privileges, possibilities and duties of each Actor.

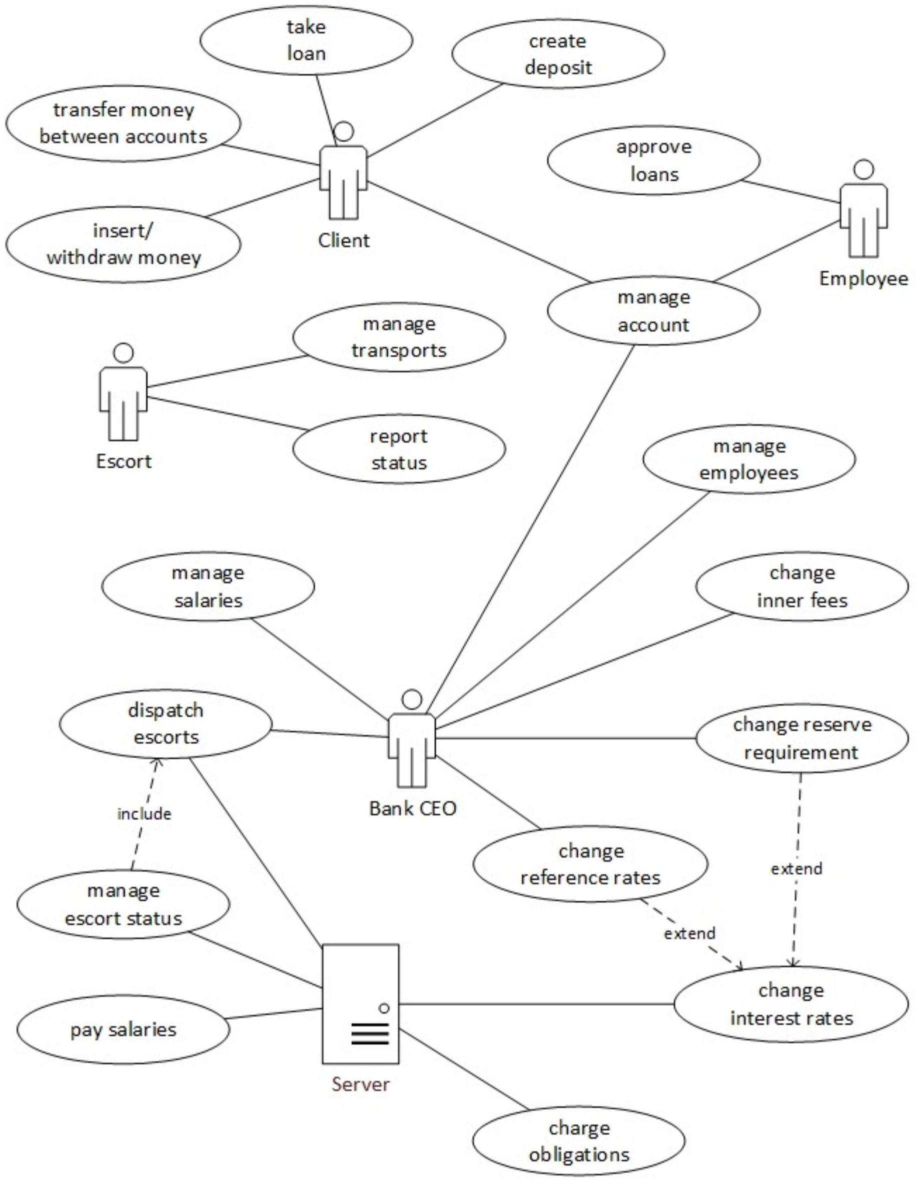
3 Design

In this section we will show our class diagram and system structure.

3.1 Class diagram



3.2 System Structure



4 Actors

In this section we will describe Actors of our application with their functionalities and user stories.

4.1 Server

Description

Server is a central unit managing flow of information and requests in the whole system. It is responsible for dispatching transports between bank facilities.

Functionalities

- Pay out obligations.
- Charge loan interest.
- Charge fees.
- Pay for transport.
- Pay employees.
- Dispatch escorts.

User Stories

- As a server I want to manage escort status so that I know where to expect a delivery.
- As a server I want to pay salaries in order to automate the process of payments for services.
- As a server I want to change interest rates in order to adjust the amount of paper money kept in the bank.
- As a server I want to charge obligations so that I can collect loan payments.
- As a server I want to dispatch escorts so that bank agencies will get money they need.

4.2 Client

Description

Client represents physical bank client, that can create and manage accounts, take loans, create deposits and make transfers between accounts.

Functionalities

- Creating an account.
- Managing account.
- Creating deposits.
- Taking loans.
- Making transfers between accounts.
- Closing account.
- Closing deposit.
- Check loans to pay.
- Withdraw from deposit.
- Withdraw from account.
- Insert money to account.

User Stories

- As a client I want to transfer money between accounts so that I have control over my finances.
- As a client I want to insert/withdraw money because I want to use money in both paper and electronic form.
- As a client I want to take a loan because I want to find a possibility to try new things in my life.
- As a client I want to create a deposit so that I can multiply my money.
- As a client I want to manage my account so that I can close it when I no longer need it.

4.3 Escort

Description

Escorts are responsible for transferring physical money between bank facilities, they regularly report their status.

Functionalities

- Managing transport.

User Stories

- As an escort I want to report my status in order to inform the bank about the progress of job.
- As an escort I want to manage transports so that everyone knows what has already been done.

4.4 Employee

Description

Staff working in bank facilities with different rank and possibility to take decisions about the clients loans.

Functionalities

- Ability to approve loan for client.

User Stories

- As an employee I want to manage clients' accounts in order to grant them privileges.
- As an employee I want to approve clients' loans so that they can get additional money.

4.5 Bank CEO

Description

The top manager of the bank, who controls economics of the bank, manages salaries of all employees and has ability to fire an employee.

Functionalities

- Managing employees.
- Decides about economics of the bank (changing inner fees, changing reference taxes).

User Stories

- As the bank CEO I want to change inner fees in order to use the resources in most effective way.
- As the bank CEO I want to manage salaries in order to pay just enough for things being done.
- As the bank CEO I want to dispatch escorts so that each facility has enough money to operate.
- As the bank CEO I want to change reference rates in order to adjust to economic situation.
- As the bank CEO I want to change reserve requirement in order to meet requirements of the government.
- As the bank CEO I want to manage clients' accounts in order to manages their privileges.
- As the bank CEO I want to manage employees in order to let people work in bank agencies.

5 Objects

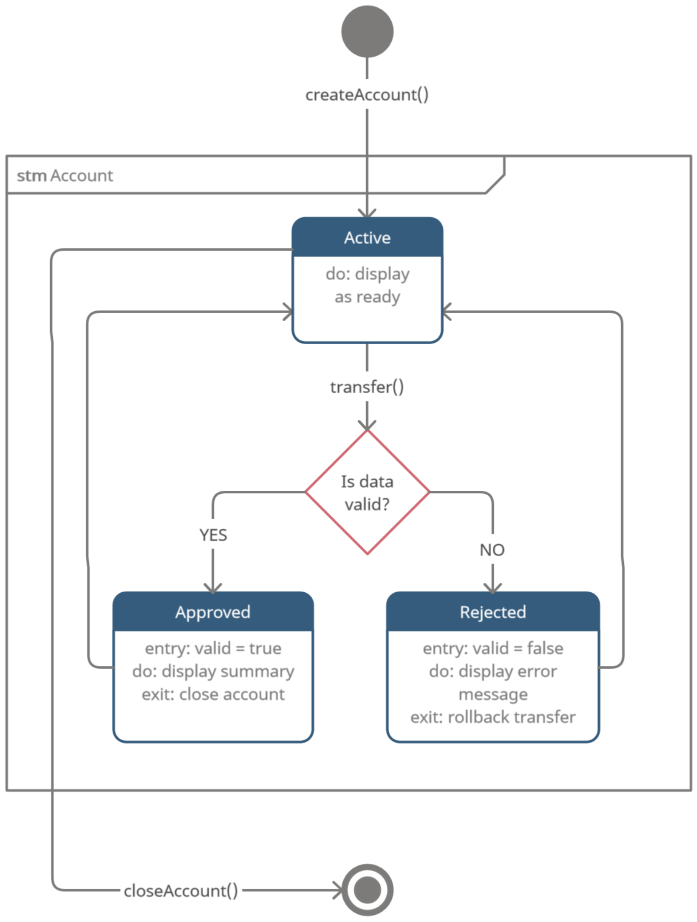
This section is dedicated to class description and state diagrams.

5.1 Account

Class Description

Account can be managed by a client in order to make transfers and control his funds.

5.1.1 State Diagram 1: createAccount()



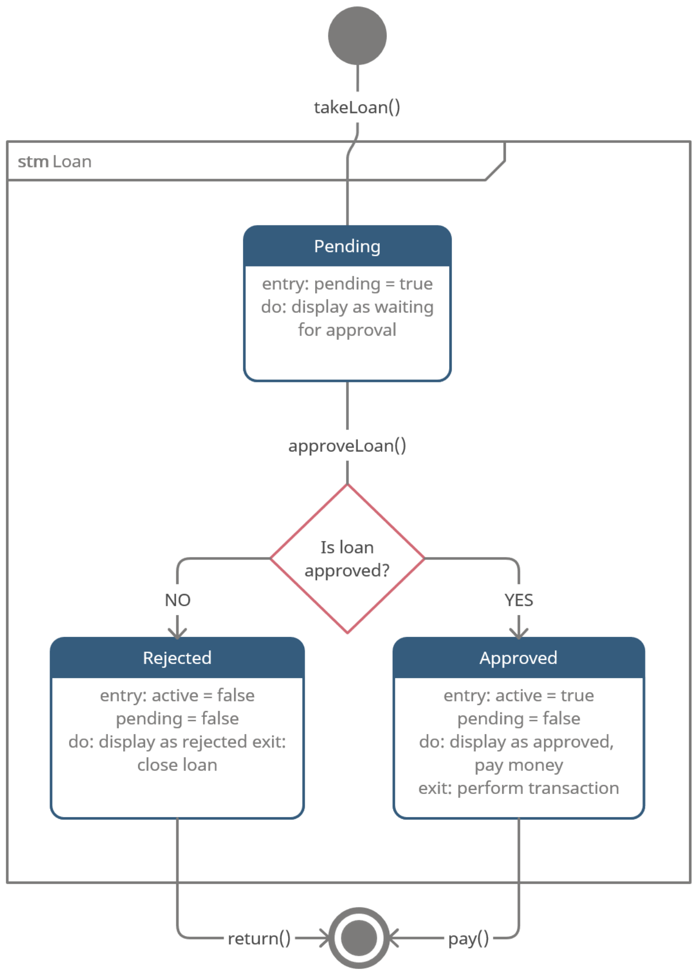
After creation state of an account is active and can be displayed to the client. From here it can be closed, what ends this state diagram, or transfer can be made. It checks if date for transfer is valid, if it is not then state is changed to rejected, error message is showed, and it goes back to active state where client can correct his mistakes or cancel the transfer. If data for transfer is valid, then state is changed to approved, client has showed summary info and states goes back into active mode.

5.2 Loan

Class Description

Loan can be taken by a client after approval from employee. Loan fees are automatically taken by server.

5.2.1 State Diagram 2: takeLoan()



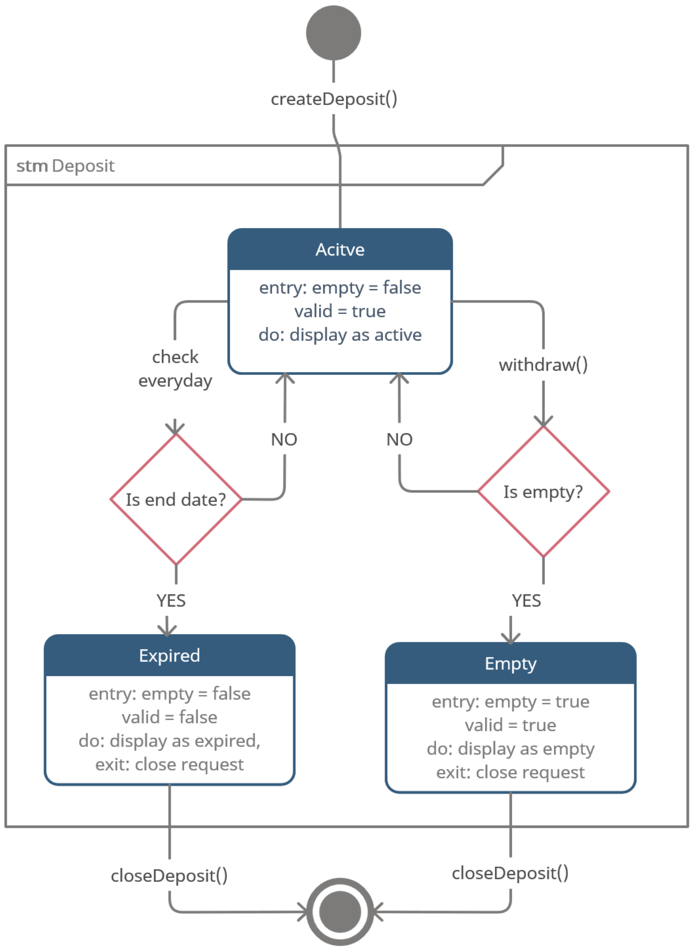
After calling the client's request for a loan the employee will have a choice based on the client's credit rating and other formal economical properties of the client to approve the loan or to not. If loan is not approved client just receives nothing on his/her account and the balance will be the same. If loan is approved client receives some sum on his/her balance.

5.3 Deposit

Class Description

Deposit can be created by a client for some period of time. Client can withdraw money. Empty or out of date deposit is being closed.

5.3.1 State Diagram 3: createDeposit()



After client pressed ”Create Deposit” / ”Withdraw”, a request sends to the server. If client wants to withdraw money, then the program checks whether balance is empty, and if not, then lets client to withdraw money. If clients wants to open deposit, then the open and close date are set, and server checks the expiry date every day, and if it expired, then the deposit is closed.

5.4 Bank Agency

Class Description

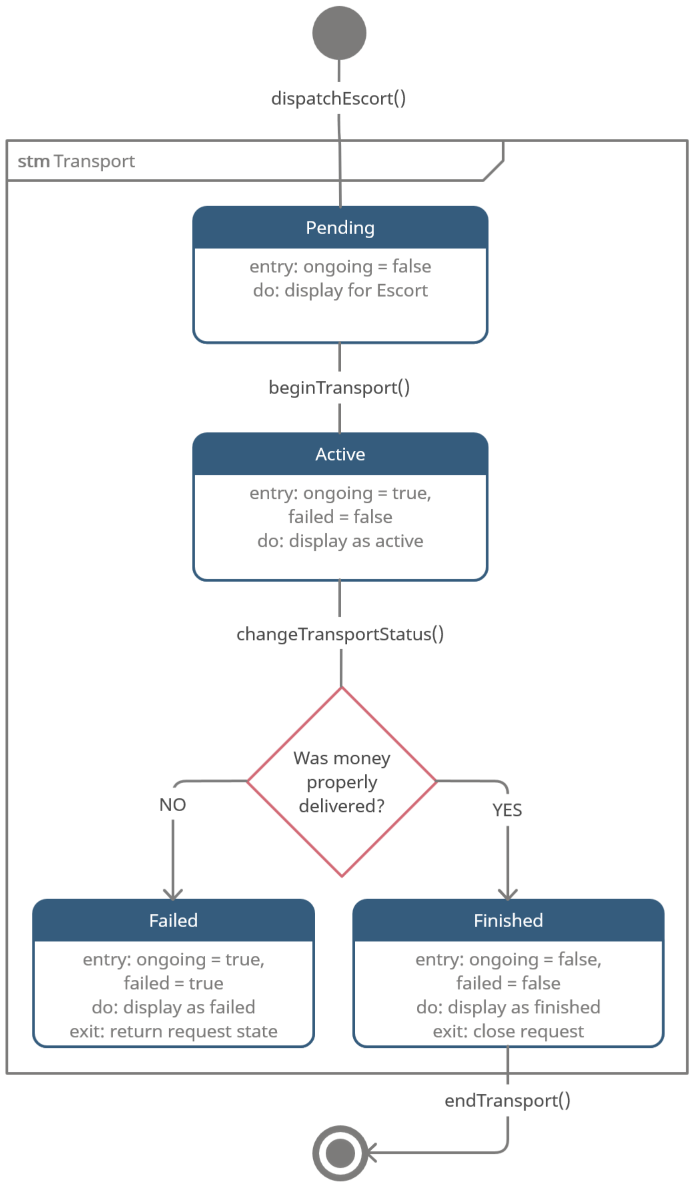
Bank agency is physical facility where clients can receive the service from the bank.

5.5 Transport

Class Description

Transport is responsible for transporting money between bank facilities performed by escorts.

5.5.1 State Diagram 4: dispatchEscort()

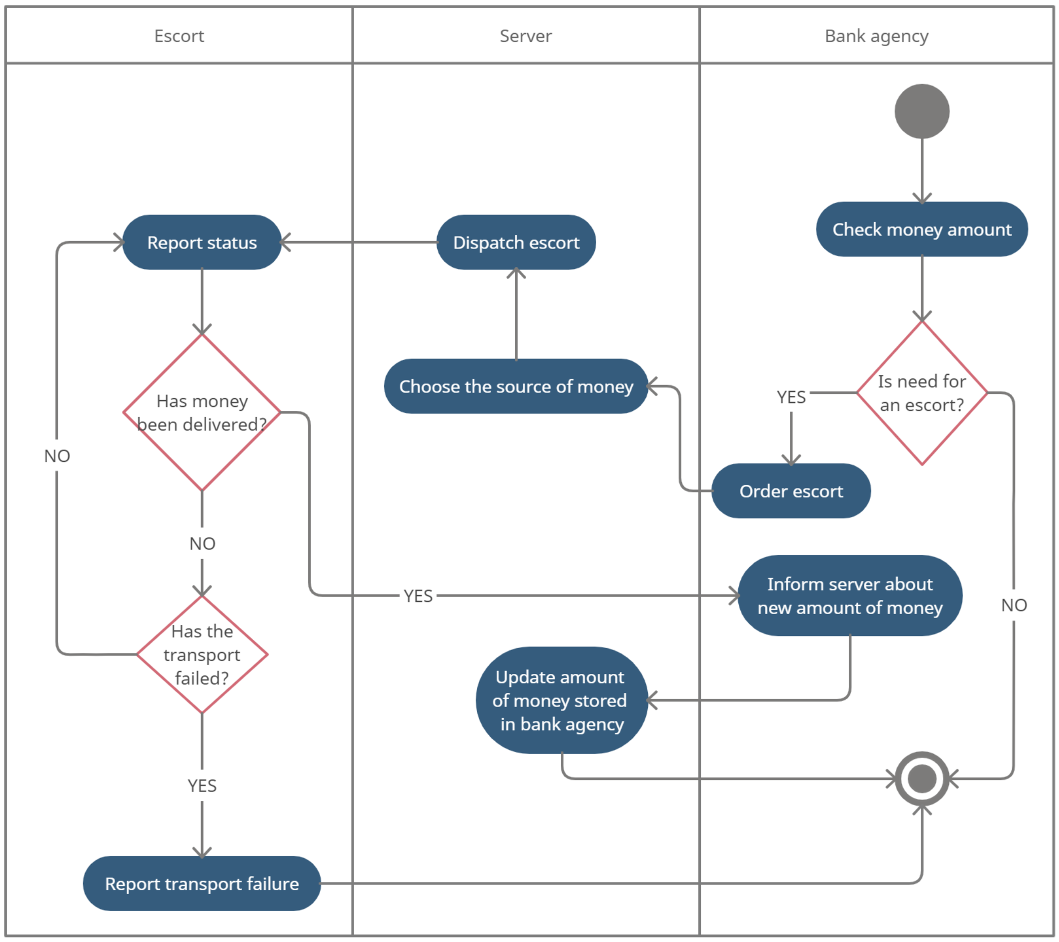


After calling the client's request for a transaction it goes to starts the transaction and checks whether the money were delivered properly or not. If not, the transaction returns to the request state. If yes, then it displays the confirmation and closes the request and the transaction.

6 Activity Diagrams

An Activity Diagram shows the execution of the process as well as the flow of messages between different actors.

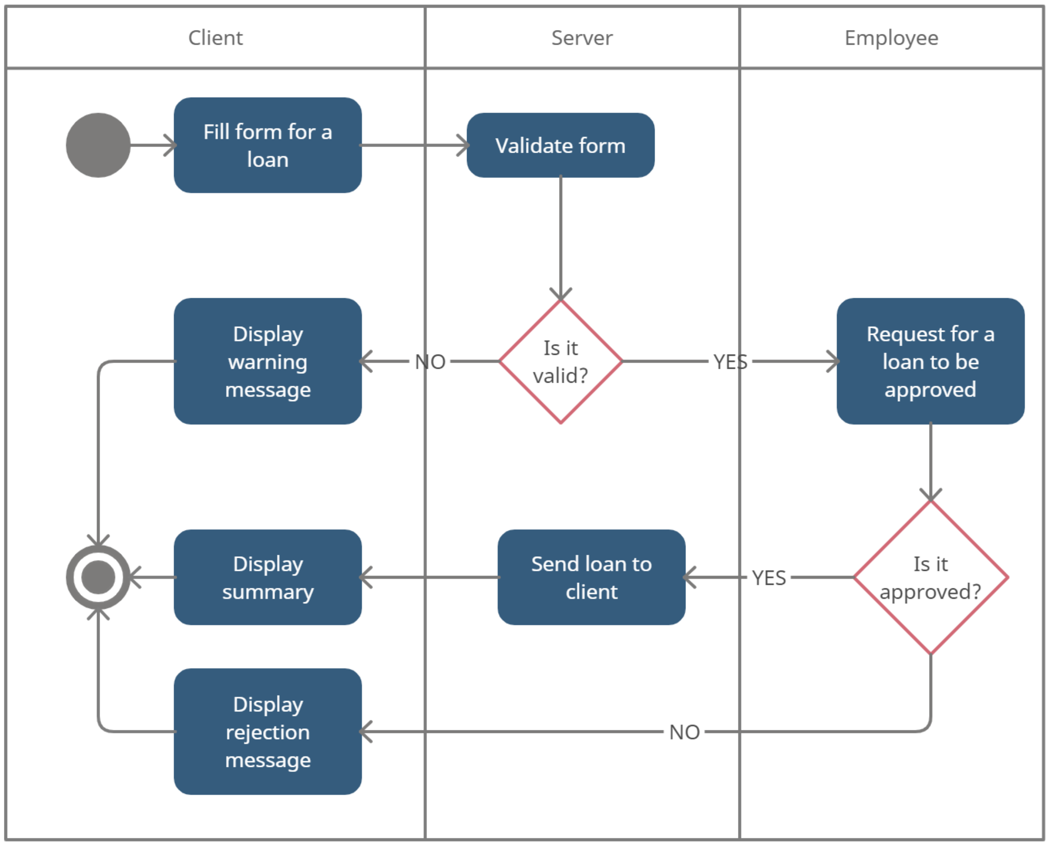
6.1 Activity Diagram 1 - transporting money



Escort
Server
Bank agency
Check money amount
Is need for an escort?
YES
NO
Order escort
Choose the source of money
Dispatch escort
Report status
Has money been delivered?
NO
YES
Has the transport failed?
NO
YES
Report transport failure
Inform server about new amount of money
Update amount of money stored in bank agency

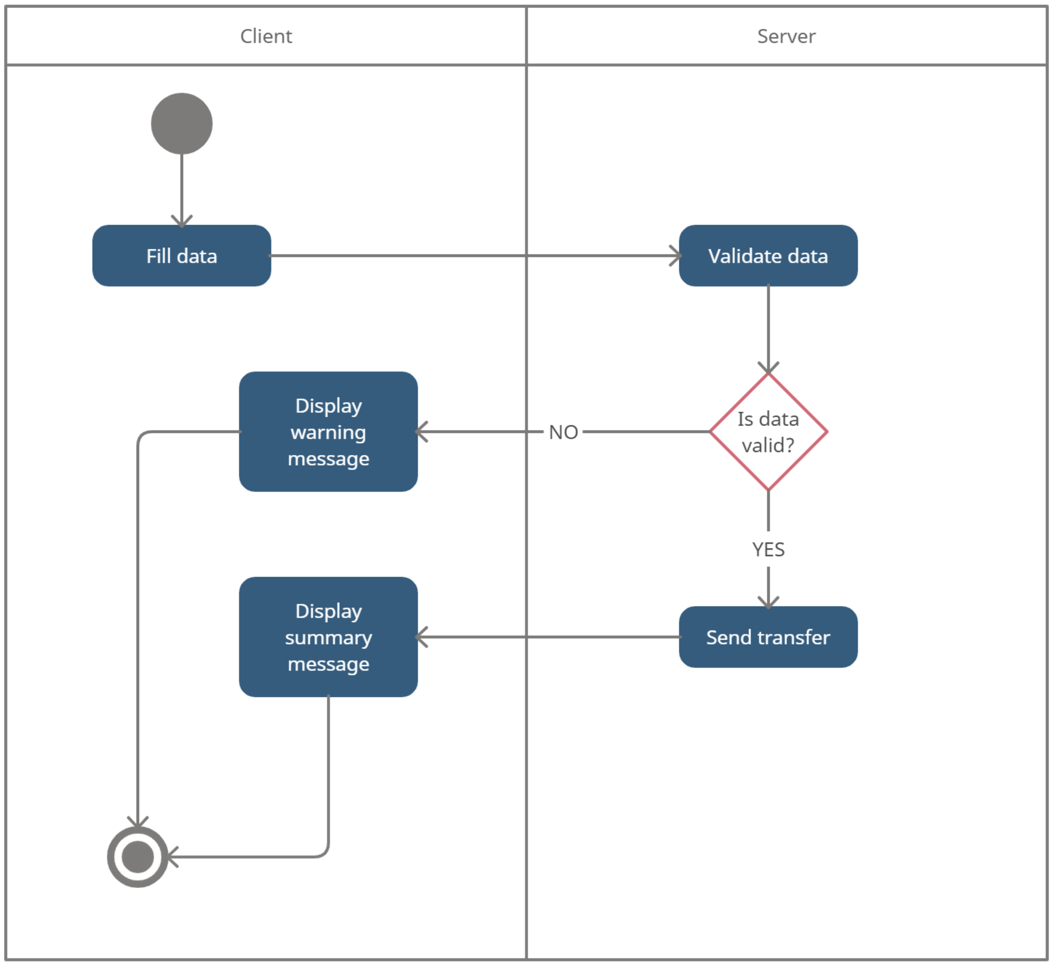
A bank agency checks periodically the amount of money it stores. If this value is sufficient to operate, the activity ends. If no, it decides to order an escort which will transport money to facility. The request is sent then to the Server, which chooses the amount of money that can be sent to the agency. A transport is dispatched and from then an Escort responsible for the delivery reports its status. If money was delivered, the agency gets information about the money obtained and updates its money balance to the Server. If money was not delivered, the escort updates the statue of delivery or reports a failure of some kind.

6.2 Activity Diagram 2 - giving a loan



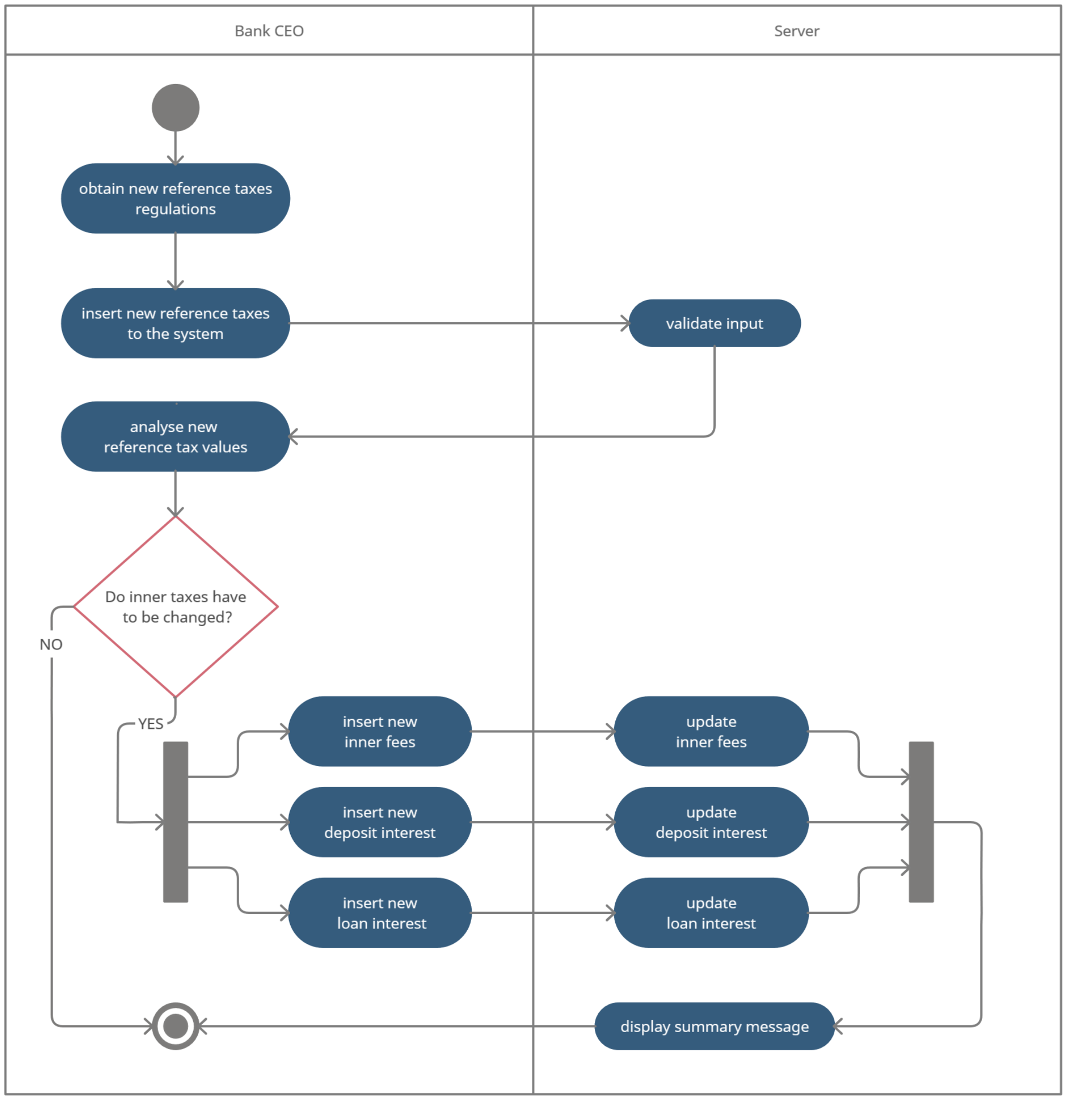
The activity starts from the Client side. Its asked to fill a form for a loan. After that the Server validates it. If no, it sends warning message to the Client and ends the activity. If its validated Server sends a request to the Employee to approve a loan. If the loan is not approved, the special rejection message is sent to the Client and ends the activity. If the Employee approves it, then he or she sends the loan to the Client though the Server, displays summary to the Client and the activity ends.

6.3 Activity Diagram 3 - making a transfer



The activity starts with asking a Client to fill the data to make a transfer. After the Server validates it. If the data was incorrect the Server displays warning message to the Client and ends the activity. If data validated successful then the Server sends transfer and displays summary message to the Client. After this step the activity ends.

6.4 Activity Diagram 4 - changing reference taxes

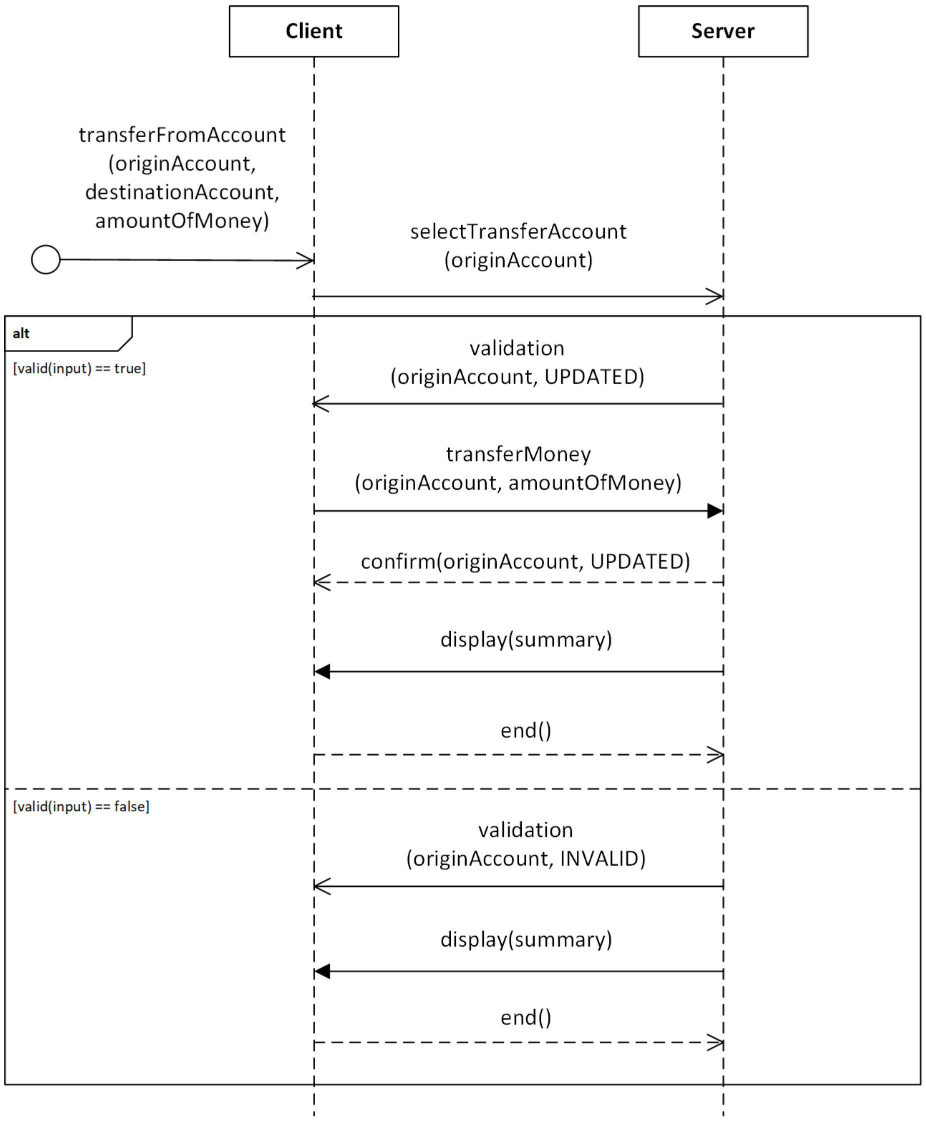

Bank CEO
Server
obtain new reference taxes regulations
insert new reference taxes to the system
validate input
analyse new reference tax values
Do inner taxes have to be changed?
NO
YES
insert new inner fees
insert new deposit interest
insert new loan interest
update inner fees
update deposit interest
update loan interest
display summary message

The activity begins when the Bank CEO is given information with new reference taxes. He or she inserts them into the system and then their correctness is validated by the Server. The CEO makes the decision whether to change fees in the bank. If no, the activity ends. If yes, he inserts the desired values of inner fees, deposit interest and loan interest. These are validated by the server, which then returns a summary message.

7 Sequence Diagrams

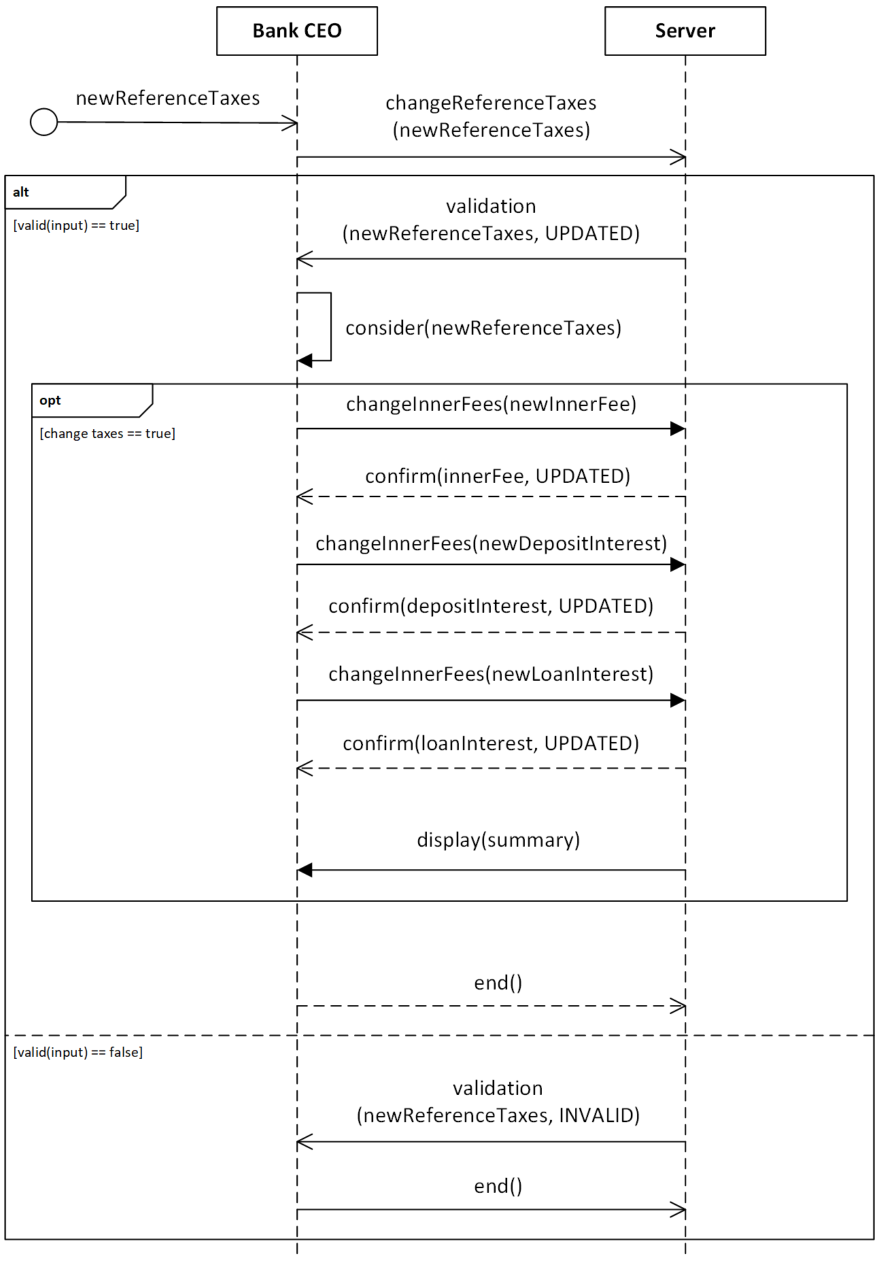
A Sequence Diagram shows the message flow from actor to another, which is used to represent the ordered information flow within a process.

7.1 Sequence Diagram 1 - making a transfer


Client
Server
transferFromAccount (originAccount, destinationAccount, amountOfMoney)
selectTransferAccount (originAccount)
alt
[valid(input) == true]
validation (originAccount, UPDATED)
transferMoney (originAccount, amountOfMoney)
confirm(originAccount, UPDATED)
display(summary)
end()
[valid(input) == false]
validation (originAccount, INVALID)
display(summary)
end()

The diagram represents the sequence of calls between the Client and the Server which is initialized by the Client who wants to transfer money from his or her account.

7.2 Sequence Diagram 2 - changing reference taxes


Bank CEO
Server
newReferenceTaxes
changeReferenceTaxes (newReferenceTaxes)
alt
[valid(input) == true]
validation (newReferenceTaxes, UPDATED)
consider(newReferenceTaxes)
opt
[change taxes == true]
changeInnerFees(newInnerFee)
confirm(innerFee, UPDATED)
changeInnerFees(newDepositInterest)
confirm(depositInterest, UPDATED)
changeInnerFees(newLoanInterest)
confirm(loanInterest, UPDATED)
display(summary)
end()
[valid(input) == false]
validation (newReferenceTaxes, INVALID)
end()

The diagram represents the sequence of calls between the Bank CEO and the Server after the CEO obtains information about new reference taxes.

8 API Endpoints

An API (Application Programming Interface) Endpoints are end of communication channels. They present here possible requests and responses in the system.

8.1 Client Endpoints

Endpoints for client CRUD requests.

8.1.1 GET

- URL: **/api/clients**
- Returns: **200 + Clients list** (OK), **403** (Forbidden)

- URL: **/api/clients/{client_id}**
- Parameters: **Client ID**
- Returns: **200 + Client details** (OK), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

- URL: **/api/clients/{client_id}/loans**
- Parameters: **Client ID**
- Returns: **200 + Loans list** (OK), **400** (ID is invalid), **403** (Forbidden)

- URL: **/api/clients/{client_id}/loans/{loan_id}**
- Parameters: **Client ID, Loan ID**
- Returns: **200 + Loan details** (OK), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

- URL: **/api/clients/{client_id}/deposits**
- Parameters: **Client ID**
- Returns: **200 + Deposits list** (OK), **400** (ID is invalid), **403** (Forbidden)

- URL: **/api/clients/{client_id}/deposits/{deposit_id}**
- Parameters: **Client ID, Deposit ID**
- Returns: **200 + Deposit details** (OK), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

8.1.2 POST

- URL: **/api/clients**
- Content-type: application/json
- Body: **Client (object)**
- Returns: **201** (Created), **403** (Forbidden), **422** (Client is not valid)

- URL: **/api/clients/{client_id}/loans**
- Content-type: application/json
- Parameters: **Client ID**
- Body: **Loan (object)**
- Returns: **201** (Created), **400** (ID is invalid), **403** (Forbidden), **404** (Not found), **422** (Loan is not valid)

- URL: **/api/clients/{client_id}/deposits**
- Content-type: application/json
- Parameters: **Client ID**
- Body: **Deposit (object)**
- Returns: **201** (Created), **400** (ID is invalid), **403** (Forbidden), **404** (Not found), **422** (Deposit is not valid)

8.1.3 PUT

- URL: **/api/clients/{client_id}**
- Content-type: application/json
- Parameters: **Client ID**
- Body: **Client (object)**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found), **422** (Client is not valid)

- URL: **/api/clients/{client_id}/loans/{loan_id}**
- Content-type: application/json
- Parameters: **Loan ID**
- Body: **Loan (object)**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found), **422** (Loan is not valid)

- URL: **/api/clients/{client_id}/deposits/{deposit_id}**
- Content-type: application/json
- Parameters: **Deposit ID**
- Body: **Deposit (object)**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found), **422** (Deposit is not valid)

8.1.4 DELETE

- URL: **/api/clients{client_id}**
- Parameters: **Client ID**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

- URL: **/api/clients/{client_id}/loans/{loan_id}**
- Parameters: **Client Id, loan ID**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

- URL: **/api/clients/{client_id}/deposits/{deposit_id}**
- Parameters: **Client ID, deposit ID**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

8.2 Employee Endpoints

Endpoints for employee CRUD requests.

8.2.1 GET

- URL: **/api/employees**
- Returns: **200 + employees list** (OK), **403** (Forbidden)

- URL: **/api/employees/{employee_id}**
- Parameters: **Employee ID**
- Returns: **200 + Employee details** (OK), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

8.2.2 POST

- URL: **/api/employees**
- Content-type: application/json
- Body: **Employee (object)**
- Returns: **201** (Created), **403** (Forbidden), **422** (Employee is not valid)

8.2.3 PUT

- URL: **/api/employees/{employee_id}**
- Content-type: application/json
- Parameters: **Employee ID**
- Body: **Employee (object)**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found), **422** (Employee is not valid)

8.2.4 DELETE

- URL: **/api/employees/{employee_id}**
- Parameters: **Employee ID**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

8.3 Escort Endpoints

Endpoints for escort CRUD requests.

8.3.1 GET

- URL: **/api/escorts**
- Returns: **200 + Escorts list** (OK), **403** (Forbidden)

- URL: **/api/escorts/{escort_id}**
- Parameters: **Escort ID**
- Returns: **200 + Escort details** (OK), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

8.3.2 POST

- URL: **/api/escorts**
- Content-type: application/json
- Body: **Escort (object)**
- Returns: **201** (Created), **403** (Forbidden), **422** (Escort is not valid)

8.3.3 PUT

- URL: **/api/escorts/{escort_id}**
- Content-type: application/json
- Parameters: **Escort ID**
- Body: **Escort (object)**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found), **422** (Escort is not valid)

8.3.4 DELETE

- URL: **/api/escorts/{escort_id}**
- Parameters: **Escort ID**
- Returns: **204** (No content), **400** (ID is invalid), **403** (Forbidden), **404** (Not found)

9 Users simulation

This chapter presents a probable working experience for users of our Web application.

9.1 Welcome page

1. The user is able to choose two different options: to register a new account or to log in to their existing one. There will be two labels for login and password, 2 buttons under the password label on the right and left corners. On the left side, there would be a log-in button, on the other side - a sign-in button. By clicking on the sign-in button the user would be moved to the page for new users.

9.2 The Client

1. When the user goes to the registration page he or she would notice six labels, two buttons, option to choose his gender and date of birth. The labels would consist of name, surname, phone number, email, and two for passwords. Also, there will be „Continue" and „Add files" buttons. The first one would be inactive while all the fields would not be filled and the „Add files" button would be used to send the scan of the passport. If the user used an already registered email, then the application would inform him or her about it.

2. After successful registration or login into the account the user will be redirected to the main page of our application. On the good part of UI, he or she would see their balance and account. Under this feature there would be three different buttons to click:

   - history of payments,
   - transfer,
   - available cards and accounts.

   By clicking on them the user would be moved to the specified page. On the bottom, there also would be a log-out button to log out from the account.

3. By clicking the „History of payments" button the user would see all the previous transfers and payments. It would be simple enough and easy to check everything. Also, there would be an option to choose the number of payments or the date. The user may click on every bill and check all the details about it.

4. For the „Transfer” button it would be possible to choose the type of it. According to it, new labels would be displayed. Account name, account, the amount of transfer, option to save or not to save the account. If a „Top-up mobile” option, then only the phone number and the amount labels would appear. For the „BLIK” option a BLIK code would be displayed.

5. When the „Available cards and accounts” button is used the user would be redirected to the simple UI page with all the information about his or her accounts and cards. Images of cards would be displayed on the left side of the card. By clicking on any of it all the information about the state, account, or card numbers, date of expiration, and owner’s name and surname will be shown.

9.3 The Bank Worker

1. While opening our application with the „Worker” status of the user, one will be able to see the list of clients, which will be loaded from the database. The following info about each client will be printed:
   - Client ID,
   - deposits,
   - loans,
   - verification status,
   - Escort status (if needed).
2. The Worker will be allowed to perform the following operations:
   - change verification status (by clicking on a tick),
   - support clients via chat (by clicking on the chat icon, there will be a chat between client and worker opened),
   - assign the escort crew to perform a physical transfer.

9.4 The Escort

1. While opening our application with the „Escort” status of the user, one will be able to see the following:
   - the crew list (all escort workers are split into crews),
   - the list of the performed and upcoming transfers (date, time, the starting point, and the destination point).

10 Abnormal behavior

During the usage of our application, some unwanted program behavior may occur. The examples of such behaviors is listed below.

10.1 Incorrect data entry

If the user or employee enters incorrect data (for example, the user wants to make a transfer to a non-existent account) - the system will not perform this operation, will display an error message and refresh the page, allowing you to enter new data.

10.2 Lack of resources on the server

If the server capacity is not enough to serve all the requests of the bank - the server operation will be stopped immediately, all transactions will be suspended if possible and all attempts to send a request to the server will end with an error message. All operations will continue when the servers return to normal.

10.3 External impacts on application operation

In case of a DDOS attack or physical destruction of the servers, all possible operations are immediately suspended, all users and bank employees are notified about technical failures. At the first opportunity to use the application safely, everyone is notified accordingly.

10.4 Suspicious user behavior

If the user can't log in to his account at the 3rd attempt, the system will offer him to pass CAPTCHA. If the user solves the CAPTCHA, but still can't enter the correct login data, the user's ability to log in will be blocked for 10 minutes with the possibility of extending the blocking. Blocking interval can be from 10 minutes to 24 hours.

10.5 Disconnection of the server

If all other system components work normally, but the server still does not respond to requests, the user will get an error of no connection to the server. After some time, the user will be able to try to communicate with the server again.