Table of contents

The document purpose

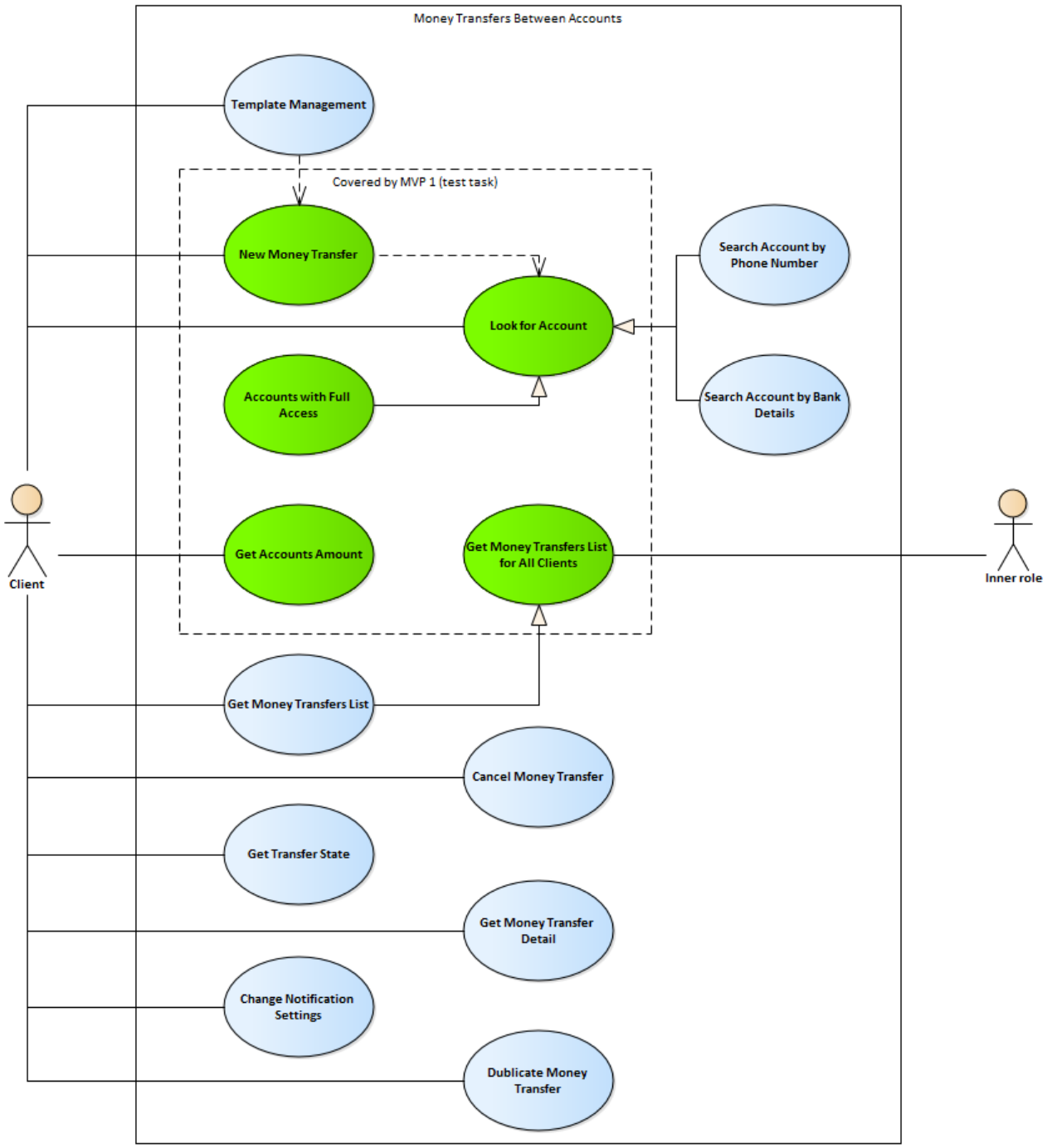
This document provides reasons for the solution architecture implemented for backend test. The purpose of this document is the explanation of the decisions in the solution.

Link to the source code: https://github.com/Yuriy-Tikhonov/FinanceApp

The approach to the task

To find the scope of this solution I started with main Use Cases and selected which of them are crucial (green color below).

Use Cases



Use Cases for MVP (test solution)

1. New Money Transfer. Client posts data to create a new order to transfer money.
2. Look for Account. The client needs account list to find the account for the transfer operation.
3. Get Accounts Amount. The client wants to check amounts on all accounts where he has access. I can use it to test the operation processing results.
4. Get Money Transfers List. Some inner system role needs the list of all transfers in the system to check amounts, statuses, etc. I need it in this task to test order creation operations.

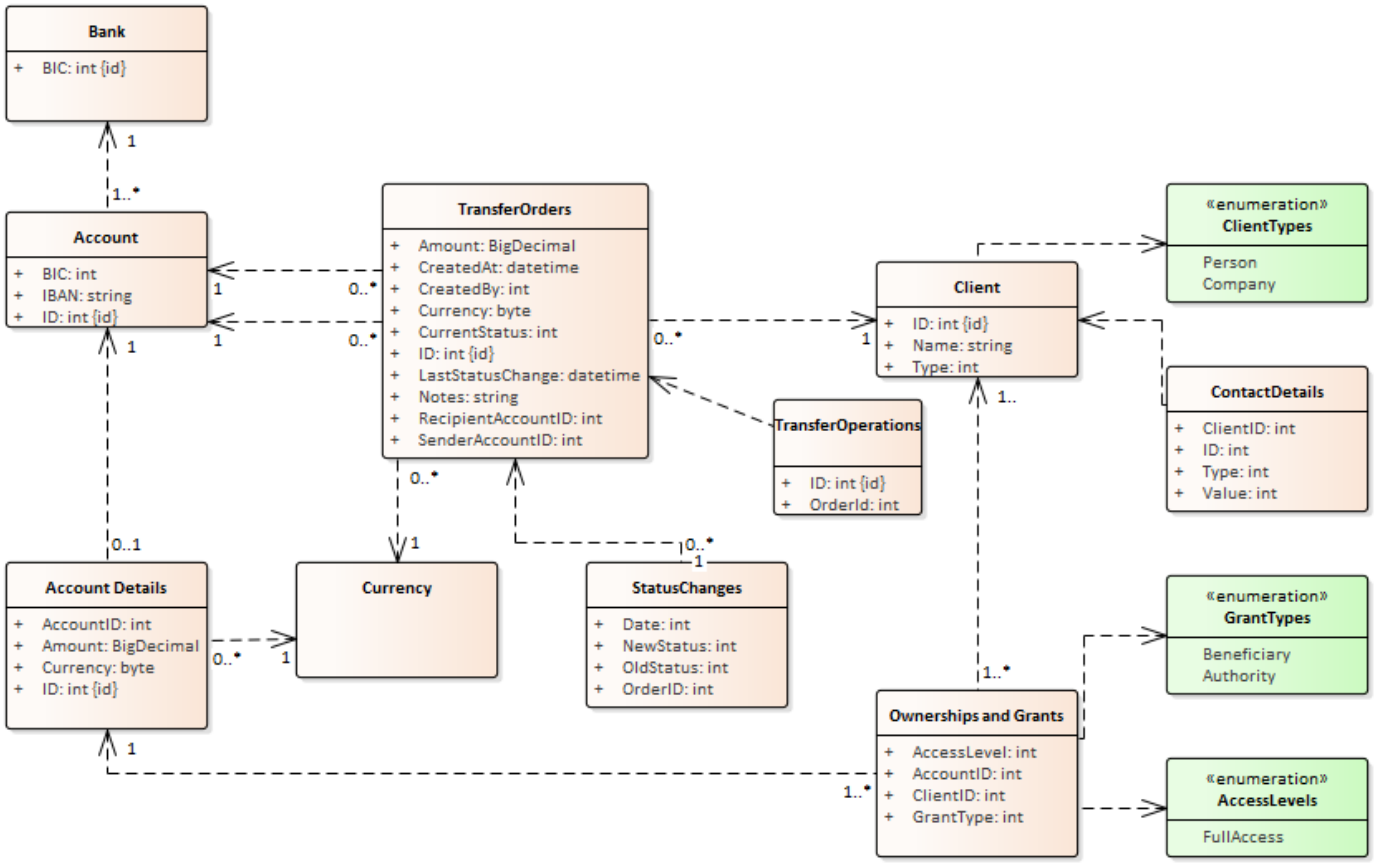
System architecture

Architectural decisions and reasons for it

1. The actual time for inner procedures and data transfer might take a long time in terms of REST communications.

2. Money transfer will need some additional procedures like risk management, control of rests on accounts, fees, taxes, etc.
3. Therefore, it is crucial to implement independence REST to create orders from orders processing. I think quite a good way for communications between services might be message broker. Because a standalone application must be created for this test, I added the direct call of processing. You can find comments in source code.
4. Transfer operations will be created during processing orders. It allows us to organize separated flow for client orders and money operations.
5. Operation processor should have several points to extend it functional by external modules or applications.
6. The client of the company might have several accounts with different access levels. In this solution, only full access mode will be implemented.
7. A client application might need to implement some additional logic in terms of account list. Special account management service will be created because of it.
8. The client might need to transfers money between accounts inside the financial company or he can create a transfer from his account to an account in any other bank. The processing of the operations will be different.
9. Transfers from another bank to client account not supported in this task because these operations should initialize in a different way.
10. Operations from one unknown bank to another unknown bank not supported in this task because the test system cannot control the amounts on these accounts.
11. Control of rests on accounts ignored in this solution because it should be covered by special procedures. I can suppose that the amount must be zero or positive but in practice, I think that it might be negative in some cases.

Data structure

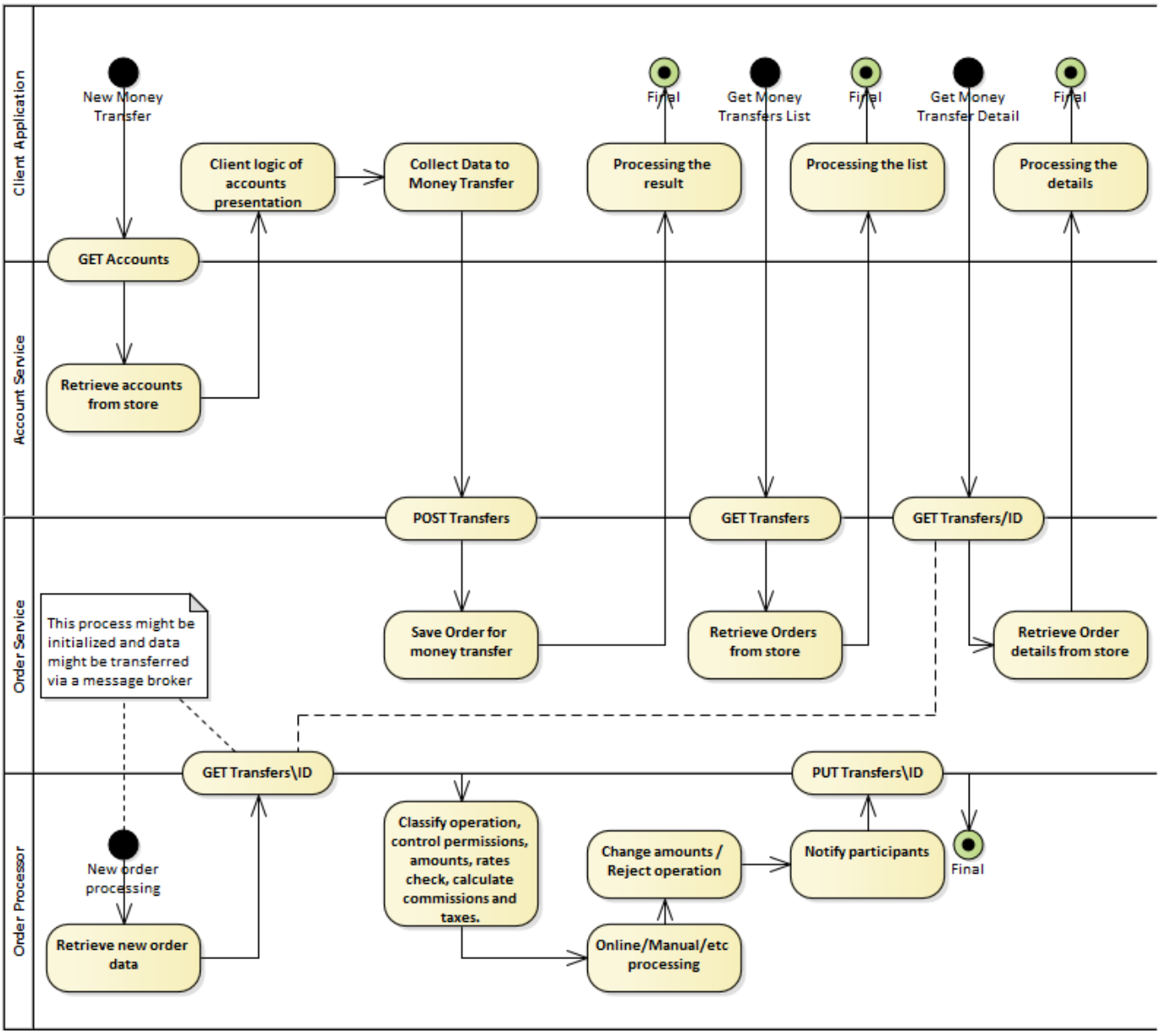


Some comments to the diagram

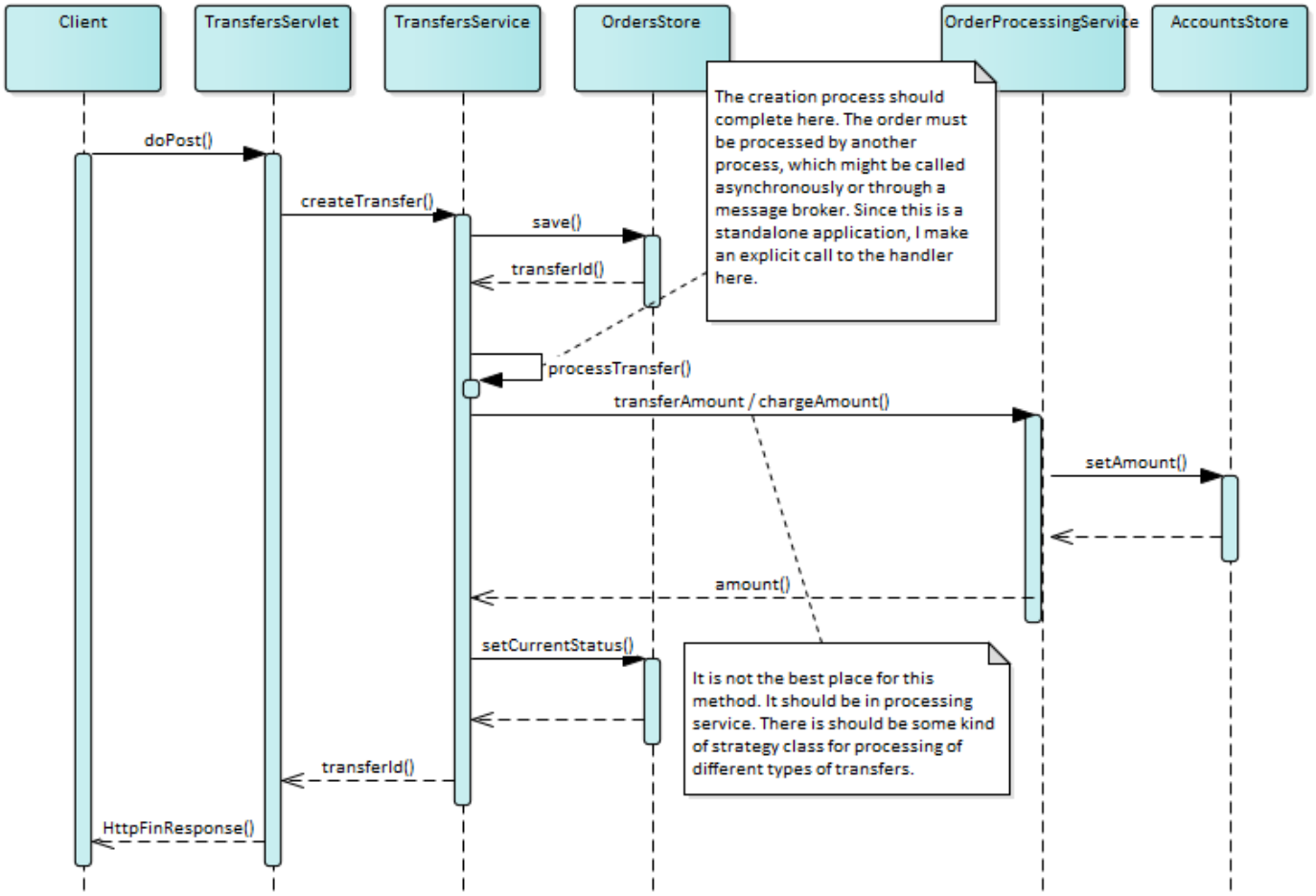
1. Transfer Orders class is the main class to store data posted by a client application.
2. Transfer Operations class is not implemented in the test application because I decided not to implement money flow documents because of some lack of details of this process.

3. Account class used to store information about external accounts.
4. Account Details class collect information about accounts on which we can control the amounts.
5. Using of Currency enumerator instead if java.util.Currency implemented because the Currency class should be customizable and will be connected with the company's storage data format.
6. Classes Status Changes, Client, Contact Details and Ownership and Grants classes implemented in the minimal volume to provide information about Client ID and to select list of accounts by Client ID. The full access mode implemented only.

Interactions of modules



Simplified sequence diagram



Additional assumptions

1. Access rights to accounts can has different types. We consider only "no access" and "full access".
2. DebitAccount should not immediately debit money. It might block money when an order created and remove or commit blocking after actual operation.
3. processTransfer – Some additional procedures need to be added (calculate taxes and fees, convert Currency by using some rates). It implements additional process depends on the risk management of the company.

List of implemented REST interfaces

Protocol	Version	URL	Sample of BODY	Description
GET	V1	/accounts?clientId=1		Get a list of accounts and amounts by ClientId

POST	V1	/accounts	{
			"iban":" IBAN0001234500000000000009876",
			"bic":"BIC001",
			"amount":100.0,
			"currency":"EUR",
			"clientId":"1"
			}
				Create a new account

GET	V1	/transfers		Get all transfers list
POST	V1	/transfers	{ "senderIban":"IBAN0001234500000000000009854", "recipientIban":"IBAN0001234500000000000000000",	Create a new transfer

			"amount":100.0, "currency":"EUR", "notes":"Own funds", "createdBy":"1" }	

You can find com.finance.postman_collection.json in the project folder.

Top priority tasks to implement for the next version

1. To integrate Swagger as the best way to document this REST.
2. Create the index page.
3. Input parameters validation.
4. Error codes and messages.
5. Integrate logging.
6. Improve the Transfers REST.
   This service is useful for an inner purpose only now. It requires implementing an analysis of client id for client purposes.
   a. Implement user context
   b. Implement paging

Second priority tasks

1. Integrate real storage.
2. Create a load test.
3. Add versioning by using Accept header.
4. Use HTTP codes to inform clients about action results.
5. Implement WebHooks or any other strategy to notify the client about orders or accounts changes.
6. Create documentations for modules.

Estimation of time spent on the solution

1. Design of the solution – 2h
2. Some knowledge about Servlets, Jetty, Guice etc had required some refresh – 4h
3. Implementation – 8h
4. This document – 2h