Project Description: DTUPay

Anshul Chauhan, 192164
Casper Bøgeskov Hansen, 190344
Danial Virk, 193167
Dumitru Racicovschii, 193166
Ionela Marinuta, 190388
Ismael Garrido Mansoa, 192675

January 2020

1 Introduction

This project is about building a *Mobilepay*-like payment system as a monolith and subsequently splitting it into a more desirable microservices architecture. The payment system, *DTUPay*, services customers and merchants.
DTUPay provides customers with *tokens* which they can use to authorize bank transactions on their behalf to merchants. Obtaining and using tokens are arbitrated by DTUPay and must adhere to certain rules, e.g.each customer may have up to 6 tokens in their account at one time, a token may only be used once, and every token usage is recorded in a transaction.

2 Team workflow

During this project period we spent most of our time working in the same room, but we also created a Slack channel so we could communicate at all times. A *GitHub* repository was setup for our code, where we worked in different branches and tried to make and review pull requests to keep up version control and maintain the integrity of the system on the master branch. We used *GitHub Projects* as a Scrum board to divide the tasks among group members. However, we also used our Slack channel for this. Group meetings were held especially at the start and end of the day to get status updates and discuss objectives, but we also frequently discussed what we are working on and what issues we are having.

Regarding the actual tasks, we have divided the project so that each of team members was able to work with each of the technologies learned through the course. Each of us has done at least one Cucumber Test Scenario, one Junit test class, a REST endpoint, and one microservice with its corresponding Docker image. We first started the course doing the tasks individually or in pairs until getting to task 4, where we had to think of the design and the approach to the DTUPay application.

Here, after brainstorming on our meetings after lecture, we decided on a specific design we agreed on, and divided the work so some people worked on figuring out the in-memory database to use, some others worked on Token management, and some made Cucumber and Junit testing.
Task 4 took us a longer time to be done, as some tasks we divided were dependent on the others. However, some of the DTUPay functionality was added in parallel with subsequent tasks, such as the transaction reporter. Task 5 was rather simple, so one person handled importing the SOAP Bank service.
We started task 6 with one person creating the first REST interface for the customer. The group members who had not yet done Cucumber and Junit tests did them for the REST services. This task was definitely the most time-consuming, as we decided to change from an in-memory database to a MySQL database. There was a person that worked on setting it up for the application, another while the others worked on

creating the other REST endpoints (merchant and token) and defining Cucumber and Junit tests for them. It also took a long time to get all these endpoints to work and connecting them to the new database. Task 7 was simpler as a few team members were experienced with Docker and were able to deploy the system quickly.

Regarding splitting the monolith application into a microservice-based application, we started with splitting the Customer service first and going through that together. We subsequently divided splitting the rest of the services (Token, Merchant, Transactions and DTUPay) amongst a few other group members.

3 Architecture

3.1 General diagram of the system

Here is an overview of the final implemented system. As can be noticed from the diagram, we have split the monolith into four microservices: transactions, tokens, customers and merchants.

To allow for lowest possible coupling, we have a central service that knows all the other microservices, such that none of the microservices knows about eachother. This also extends to database access – each microservice has its own database.

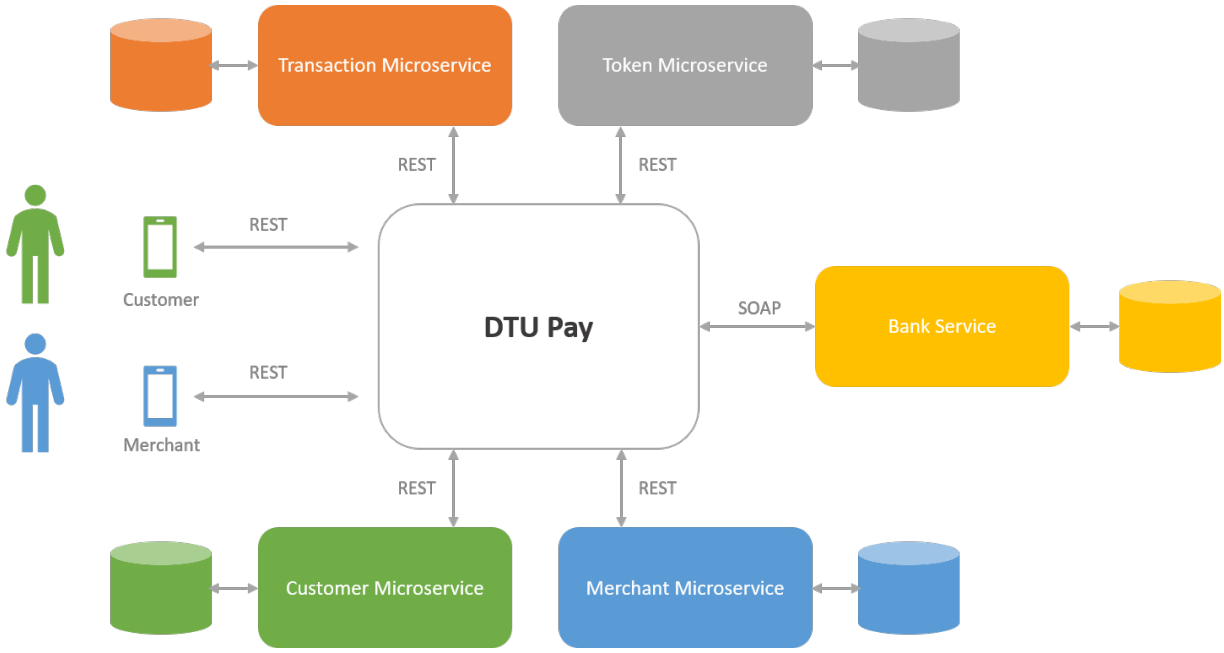

Figure 1: Diagram of DTU Pay

3.2 Customer Microservice

The customer is the end user of our application. It is composed by its name, CPR and a unique id. In the DTUPay app system, a created customer will be able to have tokens, which can be used for transactions. This microservice includes a REST service where you can request a list of all customers, a specific one, create a new one, update an existing one or delete a registered one. When using this REST service, it will use the CustomerManager interface that will act as an intermediary to connect to the database, in order to register and apply the changes requested.

URI Path = Resource	Endpoint	Meaning
/customer	GET	List of registered customers
/customer/{id}	GET	One specific customer
/customer	POST	Register a customer
/customer	PUT	Update a customer
/customer/{id}	DELETE	Delete a registered customer

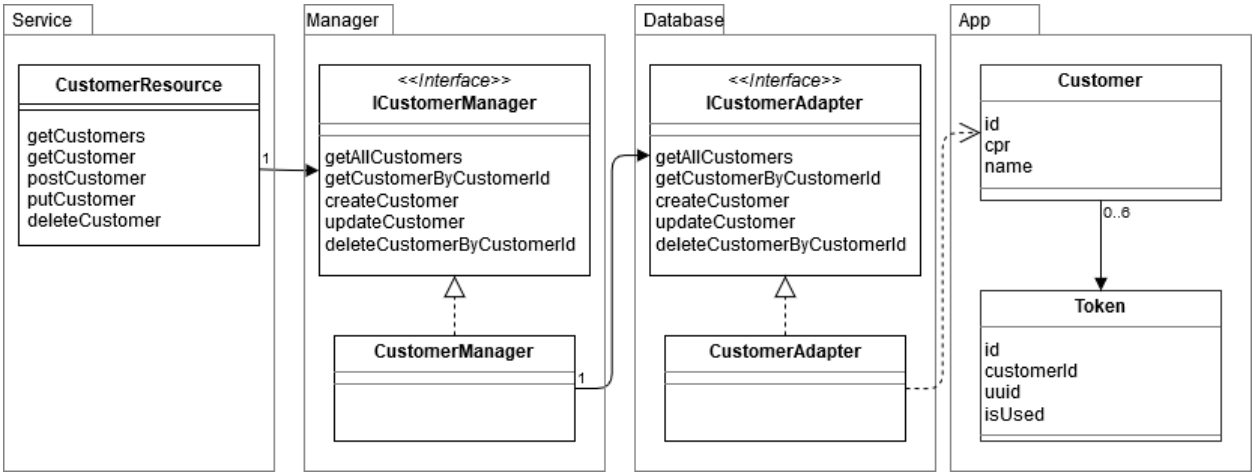

Figure 2: Diagram of Customer microservice

3.3 Merchant Microservice

A merchant (or shop owner) is the other end user of our system. It is composed of a name, a CVR, and a unique identifier. In a transaction, a merchant will be the entity that will receive the money transfer after asking DTUPay app to validate the token given by the customer.
The merchant microservice includes a REST service which is able to request a list of merchants in a call, get a specific merchant, create a new one, update the attribute information of a merchant, and delete a registered one. As done in the previous microservice, the REST service uses a manager interface to communicate with the database as a middleman. We also have a manager intermediary here that communicates the REST calls methods to the database.

URI Path = Resource	Endpoint	Meaning
/merchant	GET	List of registered merchants
/merchant/{id}	GET	One specific merchant
/merchant	POST	Register a merchant
/merchant	PUT	Update a merchant
/merchant/{id}	DELETE	Delete a registered merchant

3.4 Token Microservice

A token is an object used to verify transactions in our app. A customer will request a token and send it to the merchant, who later will ask DTUPay to validate it to perform a transaction. Our token is composed by a UUID (universally unique identifier), a customer id that links it to a customer, a used/not used variable, and an id we use as primary key on our database.
Here, we also have a REST service that includes the following features: Get all existing tokens in the database, get a specific one, ask for a customer's unused token, register a new one, generate unused tokens for a customer, change the status of a token, and delete tokens.

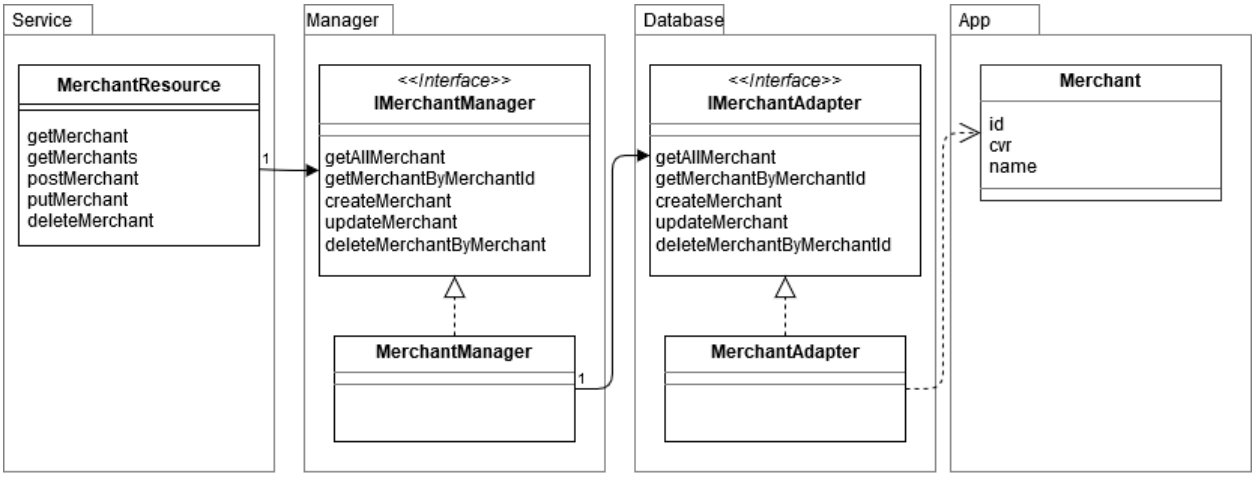

Figure 3: Diagram of Merchant microservice

URI Path = Resource	Endpoint	Meaning
/token	GET	List of registered tokens
/token/{id}	GET	One specific token
/token/unused/{customerId}	GET	A customer unused token
/token/	POST	Register a token
/token/newTokens	POST	Generate new tokens for a customer
/token/{tokenId}	PUT	Update token to used
/token/{id}	DELETE	Delete a registered token

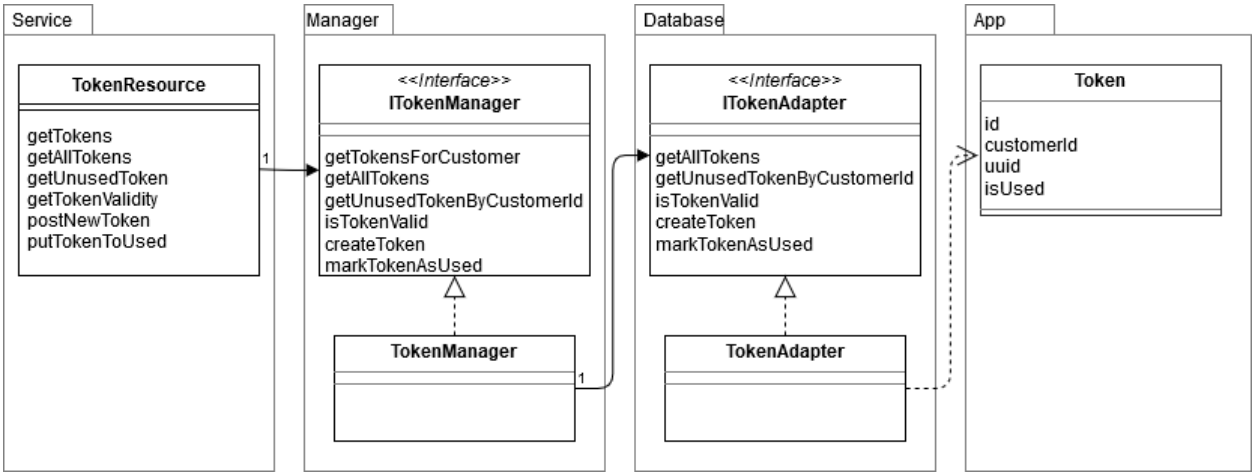

Figure 4: Diagram of Token microservice

3.5 Transactions Microservice

A transaction is the main operation and functionality of the bank application, where customers request to pay merchants a specific amount of money. Therefore, our transaction object is composed by a unique id, a timestamp, a money amount, a token id associated to verify it, and refund status Boolean variable to identify which of the ways the transactions are going.
Our REST service here can create new transactions, and provide both the customers and the merchants with monthly reports.

URI Path = Resource	Endpoint	Meaning
/transaction/{customer}	POST	Create a monthly report for a customer
/transaction/{merchant}	POST	Create a monthly report for a merchant
/transaction	POST	Create a new transaction

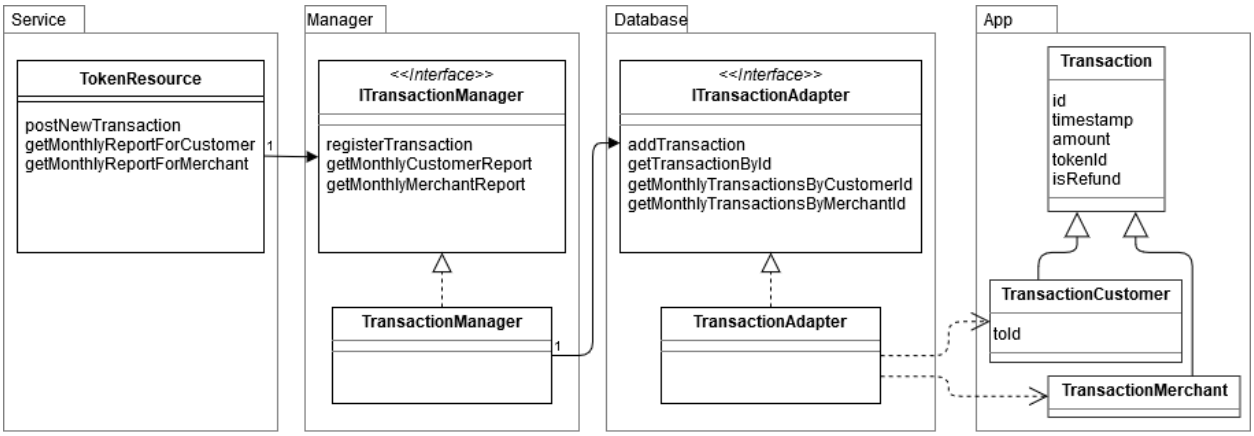

Figure 5: Diagram of Transaction microservice

3.6 DTUPay Microservice

DTUPay is our biggest microservice and the one that connects the other 4. Its REST service can create a transaction, generate new unused tokens for a customer, create and update both customers and merchants, or generate reports for them.
This microservice works differently than the others, as instead of using a rest manager interface to access the database, it actually access a DTUPay interface that communicates with the other 4 microservices by using their adapters that will then call the corresponding REST endpoints on the microservices already up and running.

URI Path = Resource	Endpoint	Meaning
/customer	POST	Create a customer
/merchant	POST	Create a merchant
/customer	PUT	Update a customer
/merchant	PUT	Update a merchant
/newTokens	POST	Generate new tokens for a customer
/customer/transactions	POST	Create a monthly report for a customer
/merchant/transactions	POST	Create a monthly report for a merchant
/transaction	POST	Create a transaction

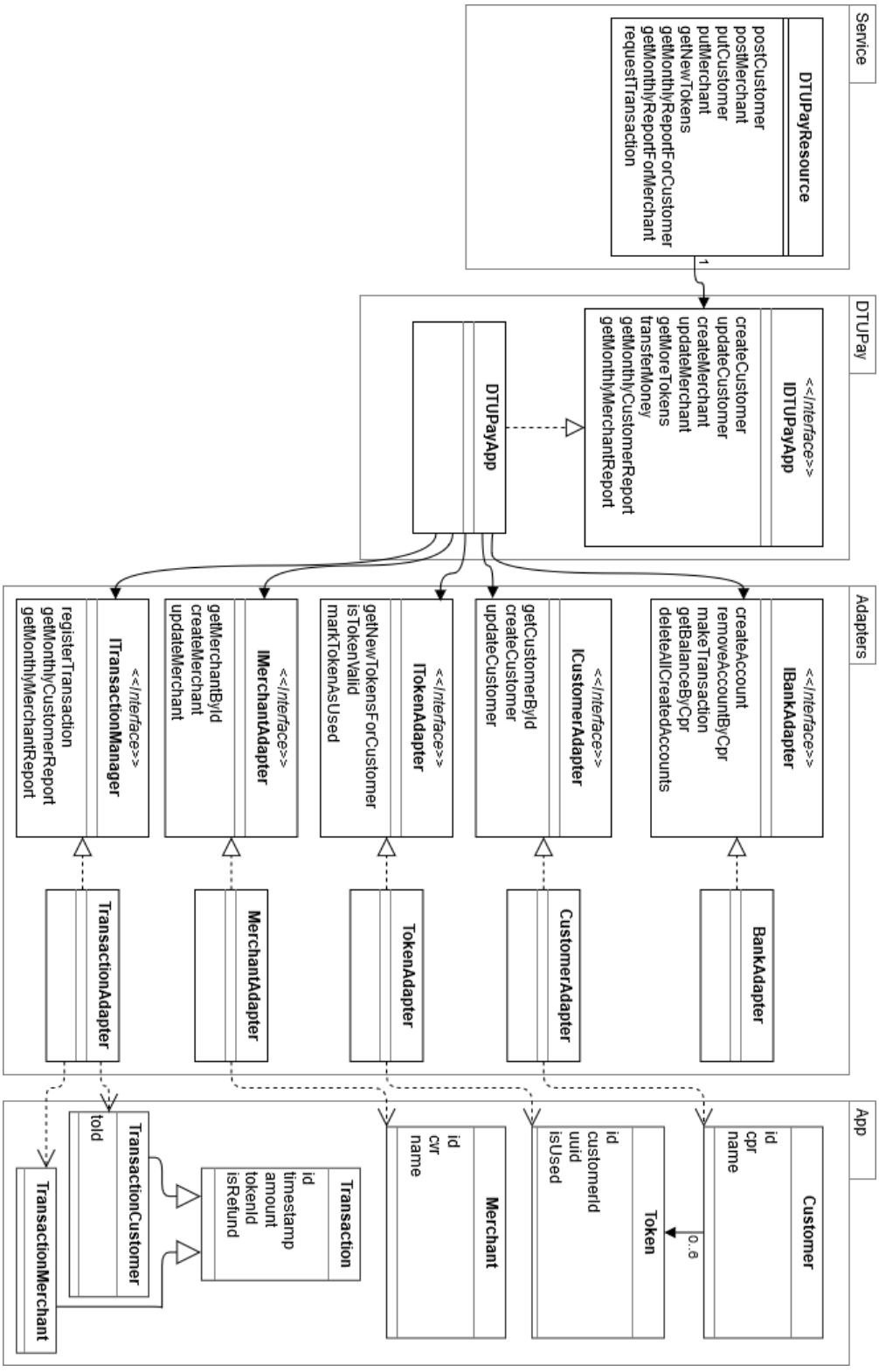

Figure 6: Diagram of DTUPay service

4 Conclusion

Microservices is a software development architectural approach that defends the structure of an application into a collection of small services in order to improve maintenance, Independence, reliability, and deployment. In this course, we have experienced how to build a solid monolith application from scratch to then learn how to turn it and separate it into a microservice structure. We have also learned how to make sure we were building a successful and working product by making use of a Continuous Integration tool as Jenkins, that would make sure each of our build was passing all the tests with no errors.

Moreover, the encouragement to use TDD and BDD testing methods as Junit and Cucumber to make sure every functionality of the system was working at every moment, ended up being very beneficial for our development, as we have been able to spot specific errors and have a better understanding on why they were failing. For example, when splitting our monolithic database to the 4 different databases for the microservices, we could make sure none of the tests were failing and we successfully did the migration.
At the end, we are all confident we now know how to build solid web services and we can use tools like Jenkins, Git, Junit, Cucumber, SOAP, REST, and Docker to improve the development of these, as also having a grasp of the benefits of the microservice architecture.