02267 - Software Development of Web Services

Ugne Adamonyte s194705 ,
Hussein Dirani s223518,
Mahdi El Dirani s233031,
Ionut Andrici s242956,
Raihanullah Mehran s233837,
Mihai Munteanu s242996,
Luís Miguel Ferreira Freire s233483,

January 2025

Contents

1 Introduction

This report presents the development and implementation of a microservices-based system as part of the DTU Pay project. The goal of this project is to design, build, and deliver a scalable, maintainable, and testable solution that facilitates efficient payment processing and account management. The system has been developed using modern software engineering practices, including domain-driven design, event storming, and test-driven development.

The project is structured into independent microservices, each responsible for specific business functionalities, such as customer and merchant registration, payment processing, and token management. These services communicate through message queues to ensure a loosely coupled architecture that supports scalability and fault tolerance.

This report provides a comprehensive overview of the project, including a detailed description of the architecture, implementation details, event storming outcomes, and the integration of testing methodologies like Cucumber. Additionally, the document explains the design choices made for RESTful APIs and message-driven communication.

2 Brain storming process

2.1 Event storming diagram

The features and scenarios implemented in our system were designed using event storming, a methodology to develop a clear understanding of the system behaviour. Using event storming, we mapped out all the scenarios to identify how the system should trigger and handle specific events. This also allowed us to define policies for each event, which needed to be enforced during the implementation phase of the code. The event storming diagram for a successful payment scenario can be found below.

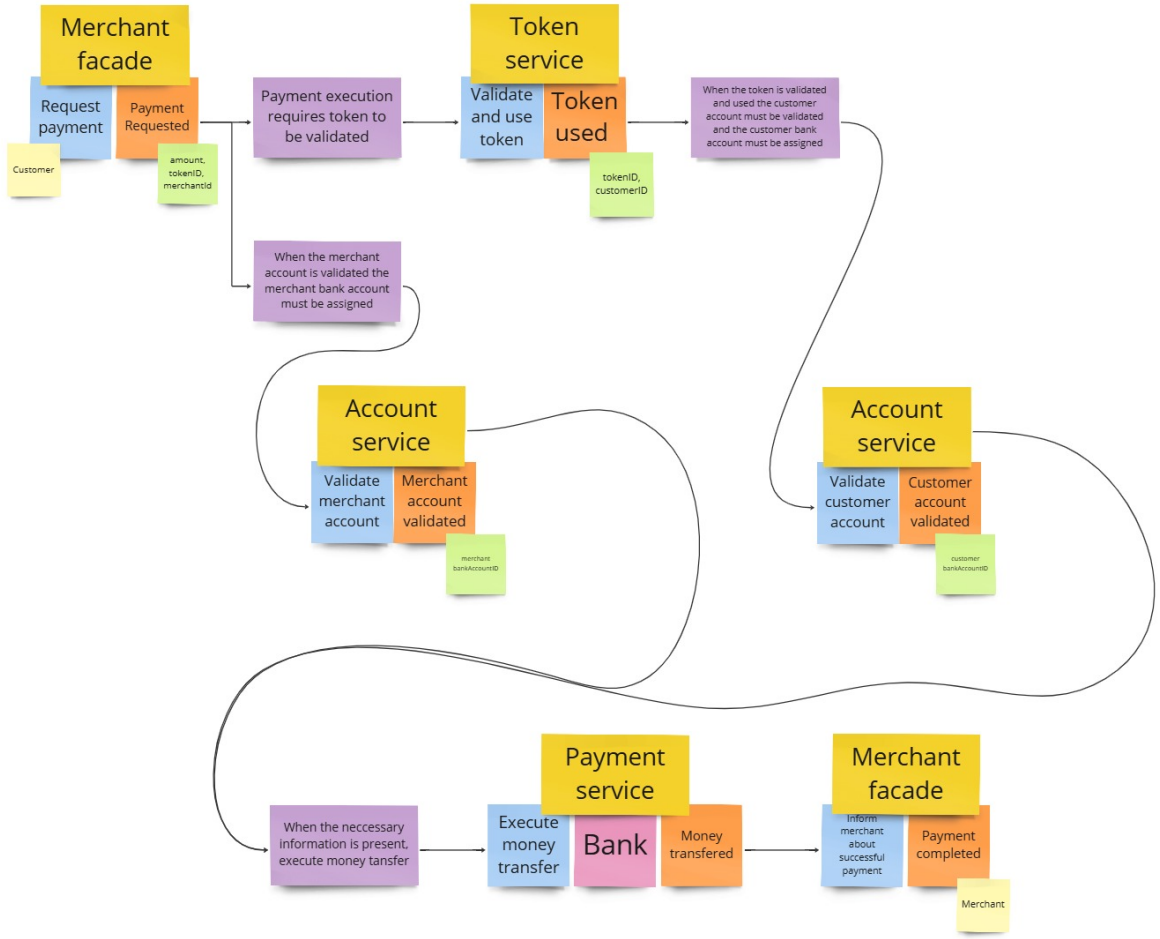

Figure 1: Successful Payment

The Successful Payment scenario is the main event flow of the system, with multiple different services communicating to validate and exchange data. The complete list of the primary event flow diagrams can be found in the appendix.

2.2 UML diagram and business logic

The DTU Pay application consists of 4 distinct microservices: Account Service, Payment Service, Reporting Service and Token Service. Excluding the Reporting Service, each of these microservices have a singleton repository that manages the storage and retrieval of service specific information. All microservices utilizes an event message object to exchange data with other components for both processing request events and returning results. As shown in the UML diagram, the Payment Service communicates with the REST Server using message passing. The message includes a correlation ID which is used to associate response event messages with their corresponding request event messages and a PaymentEventMessage object carrying the relevant request data. This design pattern is consistently implemented across all other microservices in the system.

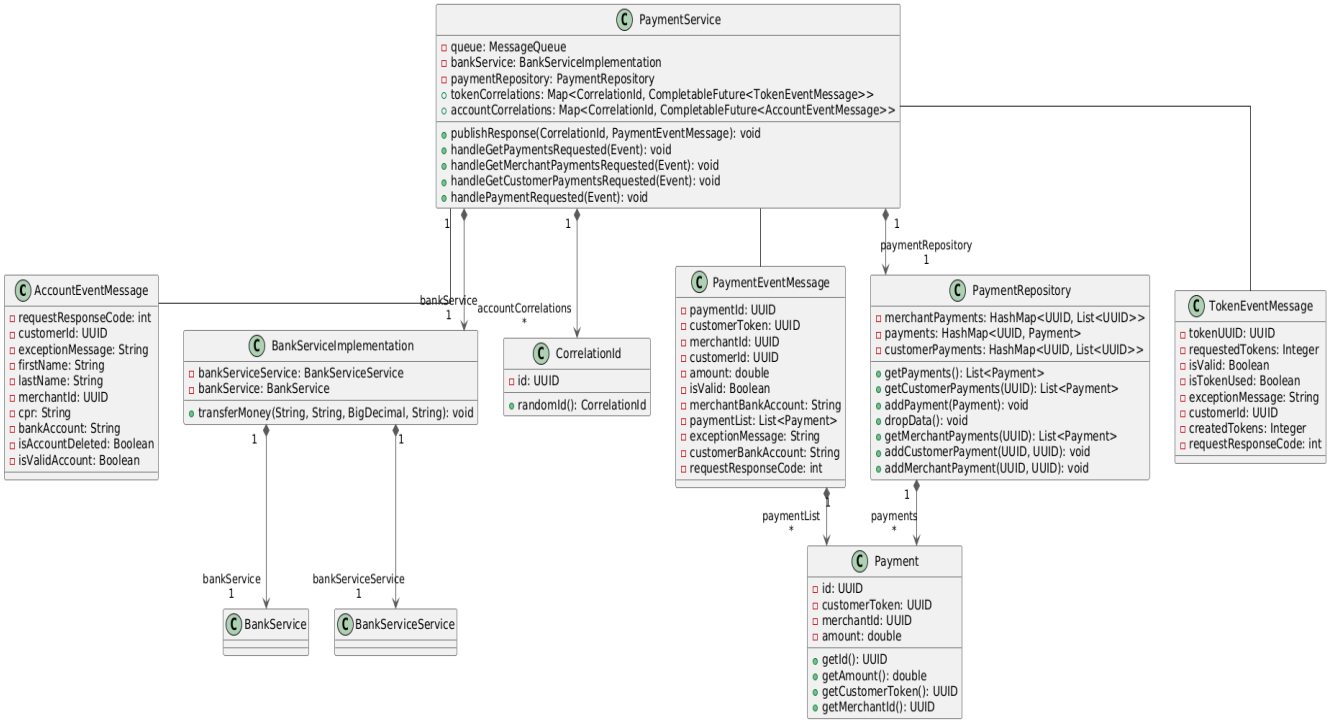

Figure 2: Payment service class diagram

3 Architecture

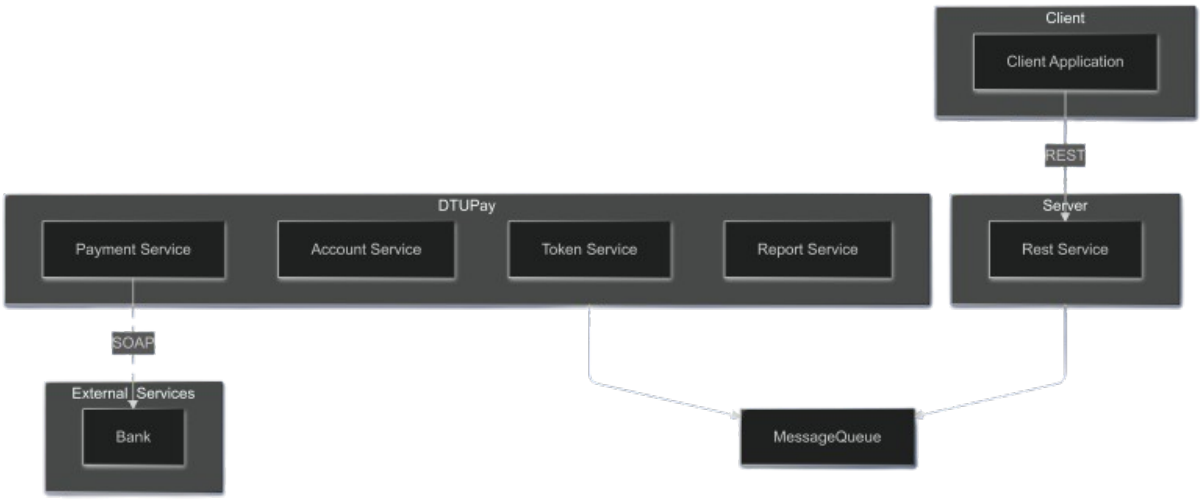

Figure 3: High-level Architecture Diagram

3.1 Microservices

In addition to the Client and DTU Payment server, the project implements the following services:

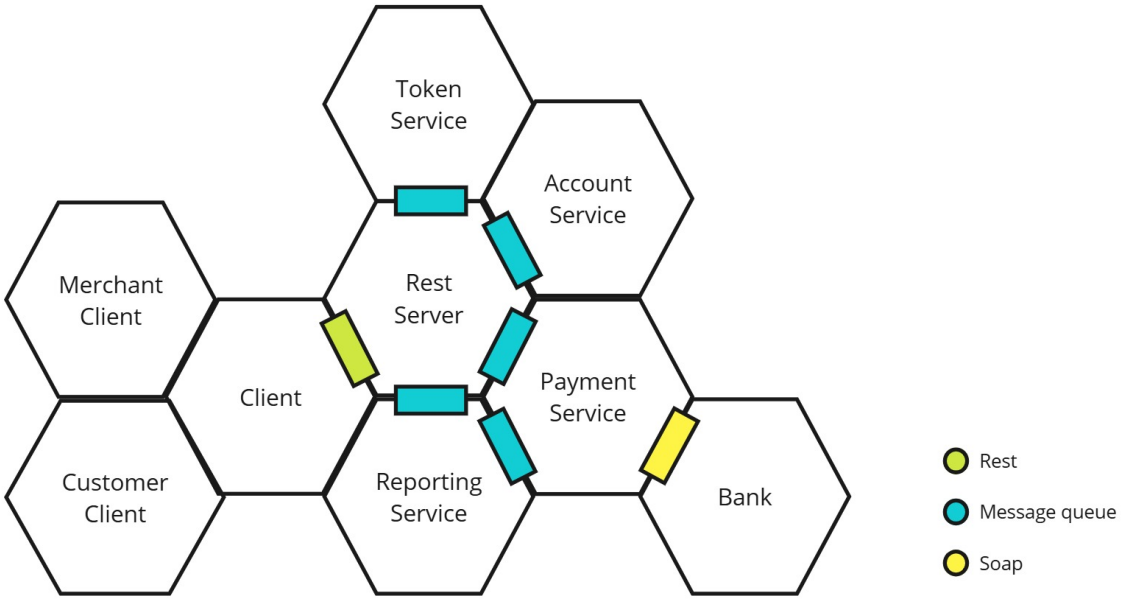

Figure 4: Hexagonal Architecture

3.1.1 Account Management Service

The Account Management Service handles both customer and merchant account operations through event-driven communication. It listens for registration, deregistration, and validation requests and processes them accordingly. The Service leverages the AccountEventMessage object to communicate account-related information through the message queue. This object carries essential details such as merchantId, customerId, personal and account data, validation status, response codes, and error messages. When a CustomerRegistrationRequested or MerchantRegistrationRequested event is received from the facade (in the DTU Pay Server), the service creates a new customer or merchant record in its repository and publishes either a CustomerCreated or MerchantCreated event with a success status. For deregistration, the service handles CustomerDeregistrationRequested or MerchantDeregistrationRequested events, removing the respective account and emitting a CustomerDeregistered or MerchantDeregistered event with the result of the action. Additionally, the service validates accounts through ValidateCustomerAccountRequested or ValidateMerchantAccountRequested events, responding with a status that includes whether the account is valid, along with relevant details like the bank account or any error messages.

3.1.2 Payment Management Service

The Payment Service is a central part of the DTU Pay application, designed to handle payments and communicate between microservices through an event-driven architecture supported by message queues. Its primary responsibilities include processing payment requests and retrieving payment histories for customers, merchants, or managers. The service uses the external bank using SOAP to execute money transfers. The message queue received from an event contains a CorrelationId object and a PaymentEventMessage object. The CorrelationId serves as a unique identifier for the event enabling the service to correlate responses with their corresponding requests. The PaymentEventMessage object contains all the essential details for processing the payment, including the customer and merchant bank accounts, the customer token, the transaction amount, and additional metadata such as the customer and merchant IDs but it can also contain error messages and response status codes. Once the transfer is completed, the payment details are saved in the system for both the customer and the merchant.

A PaymentCompleted event is then published back to the message queue, including the updated PaymentEventMessage with the transaction status, a response code, and the payment ID. In addition to processing payments, the service handles requests for retrieving payment histories. When events like GetCustomerPaymentsRequested, GetMerchantPaymentsRequested, or GetPaymentsRequested are received, the service fetches the relevant payment records from its repository. These records are included in a PaymentEventMessage

along with a success status code and are published back as CustomerPaymentsFetched, MerchantPaymentsFetched, or PaymentsFetched events.

3.1.3 Token Management service

The Token Management microservice handles token related requests, including a request to use tokens, get tokens and create more valid tokens for a particular client entity. It receives requests from Merchant and Client facades. When the token Management microservice receives a CreateTokensRequested event from a customer facade, the message queue received from an event contains a CorrelationId object and a TokenEventMessage object, similarly to other cases in our overall program structure. The TokenEventMessage object contains a customer ID, and the token service evaluates whether the associated customer has no more than 5 tokens, then creates a requested amount of valid tokens, placing them in the token repository. The repository holds a list of tokens with associated customer IDs. ResponseTokensCreated event is subsequently sent out to the customer facade, the token service message will now include an amount of newly granted tokens as confirmation and a success status.

When the microservice receives a GetTokensRequested event from the customer facade, the token service first evaluates if the associated customer entity has any active tokens via their customer ID from the TokenEventMessage object. If the customer has at least one active token, ResponseGetTokensReturned event is returned and contains a list with an ID of the returned token.

Finally, when the microservice receives a UseTokenRequested event from the merchant interface, the service validates that a supplied token ID belongs to an active token of an associated customer ID, the token is removed from a customers active token list in the token repository, and a ResponseTokenUsed event is returned containing an associated customer ID, as well as a now used token ID.

3.1.4 Reporting Management service

The Reporting Management Service is responsible for retrieving payment information for managers, customers, or merchants. It uses an event-based communication pattern with the Payment Management Service coordinating requests and responses via a MessageQueue and CorrelationIds. In the workflow, the Reporting Management Service serves as a bridge between the facades and the Payment Management Service. For manager requests, it processes payment history by communicating with the Payment Service through events such as GetPaymentsRequested and PaymentsFetched.

Similarly, for customer and merchant-specific requests, it manages interactions via events like GetCustomerPaymentsRequested and CustomerPaymentsFetched or GetMerchantPaymentsRequested and MerchantPaymentsFetched. The service uses specific mes-

sage queue objects to communicate, the ReportingEventMessage, which encapsulates essential data for requests and responses, including the customerId, merchantId, and a list of Payment objects representing the payment details. It also includes a requestResponseCode to indicate the status of the operation and an exceptionMessage for error reporting.

3.2 REST interfaces

The project implements three REST api facades - merchant, customer and manager.

3.3 Merchant interface

Resource URI Path	HTTP method	Function
merchants/register	POST	registers a merchant
merchants/deregister/{merchantId}	DELETE	removes a merchant record
merchants/reports/{merchantId}	GET	returns merchant payments
merchants/payment	POST	initiates payment

3.4 Customer interface

Resource URI Path	HTTP method	Function
customers/register	POST	registers a customer
customers/deregister/{customerId}	DELETE	removes a customer record
customers/reports/{customerId}	GET	returns client payments
customers/tokens/create	POST	cerates tokens
customers/tokens/{customerId}	GET	retreives an active token

3.5 Manager interface

Resource URI Path	HTTP method	Function
reports	GET	returns all payments

4 Features

Each microservice has been built on the premise of behaviour driven development - concluding in the features attributed to each service being covered by Cucumber testing. Service-specific tests focus on testing a part of the overall process relevant to that service, mocking the calls and responses coming from other services via a mocked MessageQueue.

In this section we describe a few of the main features of the system and the scenarios represented by them.

4.1 Register/Deregister merchants/customers

In our system, the registration and deregistration of merchants and customers are handled by the Account Management Service, which is one of the core microservices. When a merchant or customer requests registration or deregistration, the Facade Service (part of the DTU Pay server) sends an appropriate request to the Account Management Service.

4.1.1 Implementation

This section should go into technical details about how it works:
The Facade Service sends one of the following events to the Account Management Service depending on the operation:

- CustomerRegistrationRequested
- MerchantRegistrationRequested
- CustomerDeregistrationRequested
- MerchantDeregistrationRequested

The Account Management Service processes these requests by interacting with the relevant repositories. For registration, it creates a new account in the repository and responds with a confirmation event, such as CustomerCreated or MerchantCreated, which includes details about the registration status. Similarly, for deregistration, the service removes the account from the repository and sends an event such as CustomerDeregistered or MerchantDeregistered, indicating whether the operation was successful.
This event-driven communication ensures that both registration and deregistration operations are processed asynchronously, allowing the system to remain scalable and maintainable. The response events include all relevant information, such as the success or failure of the operation, and are sent back to the Facade Service, which provides the results to the user.

4.2 Create and get customer tokens

The creation and retrieval of tokens are separated between two different endpoints accessible to the client, as we believed this was reasonable during the event storming, as can be seen in figures 10 and 9.

4.2.1 Implementation

The customer is allowed to request up to 5 new tokens if they currently have at most 1 token, as highlighted in the project description. A REST request is sent to the API endpoint by the client entity to issue more tokens - an amount of new requested tokens is specified in the request along with the customerID, to name key properties.

Upon a successful request to create more tokens, a requested amount of valid tokens is created and stored in the token repository, and the client receives a response message from the token service that includes the number of new tokens granted to the customer, as well as a success status. In an event of an unsuccessful request, the customer receives a token service message containing an exception message.

A customer can furthermore request to get a token for payment. A REST request is sent to the API endpoint by the client to get a token - the customerID will be included in the request. This operation is successful if the customer has at least one valid token, which is validated by the Token Service Management microservice. In an event of a successful request from the customer, the service retreives a token from the repository. At the end of the process, the customer receives a response which includes a list with a retrieved tokenID. In an event of an unsuccessful request, the customer receives a response containing an exception message.

4.3 Payment

The payment feature allows a DTU Pay registered customer with a sufficient amount of money in their bank account to be deducted by a payment they wish to perform with a certain merchant. The merchant initiates the payment and the system validates the token, the merchant DTU Pay account, and the customer DTU Pay account. Once all is validated, then the money is transfered from the customer bank account to the merchant and the payment is complete.

4.3.1 Implementation

In our implementation, the process is managed by the REST server. Specifically, the Merchant Resource serves as the REST API entry point to trigger the event. It then invokes the payment logic within the REST Server Payment Service, which communicates with the microservices and the bank service to finalize the payment. When a merchant initiates a payment, a REST request is sent to the API endpoint. The server begins by sending a request with the customer token to the Token Management microservice for validation. Given that the token is valid, the Token Management microservice invalidates the token and responds with the customer ID. The REST server then sends another request to the Account Management microservice to validate the DTU Pay customer account and retrieve the associated customer bank account. Simultaneously, the REST server validates the merchant DTU Pay account with the Account Management microservice and retrieves the associated merchant bank account. Given all these are verified, the server then proceeds send a request to the Payment Management microservice to execute the transfer passing all the information, which then sends a SOAP request to the external Bank Service and given it is successful it returns a confirmation to the REST server of the payment, which then returns to the merchant a confirmation of successful payment.

4.4 Get payments (Manager/Merchant/Customer)

This feature enables different system actors to retrieve payment records tailored to their roles. The Manager can request a complete list of all payments in the system, while Merchants and Customers may request only their own. The request is routed through the ReportingService in ReportingManagement, which is responsible for sending events to and receiving responses from the PaymentService in PaymentManagement.

4.4.1 Implementation

When the Manager facade issues a request (e.g., via the /reports endpoint), the ReportingService publishes a message to the PaymentService, which queries the PaymentRepository for all recorded payments and returns the results. Similarly, a Customer or a Merchant facade can invoke /customers/reports/{customerId} or /merchants/reports/{merchantId} respectively, causing the ReportingService to forward a request event containing the specific identifier. The PaymentService then retrieves the relevant payment records and responds with an event that the ReportingService translates into a final response for the requesting facade. This event-based approach keeps the services loosely coupled, as the ReportingService only needs to publish events and handle asynchronous responses, with all payment data management concentrated in the managed by the PaymentService.

5 Group work

We have worked in a mob programming style throughout the project, sometimes in smaller teams, each tackling different scenarios, or when these were more complex, tackling parts of a scenario/feature and then convening to finish the whole scenario.

Feature or Task	**Contributions**
Token Service	Luis and Ugne
Account Service	Mahdi and Hussein
Payment Service	Ionut and Mihai
Reporting Service	Raihan and Ionut
Server/End-to-end	Ionut, Mahdi, Hussein, Mihai, Luis and Ugne
Queue and CI/CD	Ionut, Mihai, Luis and Mahdi
Project Report	Ionut, Luis, Ugne, Mihai, Raihan, Hussein and Mahdi
Event-Storming	Luis, Ionut, Mihai, Mahdi, Hussein, Ugne, Raihan
Installation Guide	Ionut

Table 1: Table of Features and Contributions

6 Repository and CI/CD

6.1 GitHub Repository

The project source code is hosted on GitHub and can be accessed using the following URL:

- **Repository URL:** https://github.com/WebServices-G09/dtu-pay-final
- **Access:** Repository has been made public

6.2 Jenkins CI/CD Pipeline

The Continuous Integration and Continuous Deployment (CI/CD) pipeline is configured in Jenkins and accessible at:

- **Jenkins URL:** URL: http://fm-09.compute.dtu.dk:8282/
- **Access Credentials:**
  - **Username:** huba
  - **Password:** SwDoWS09

7 Appendix

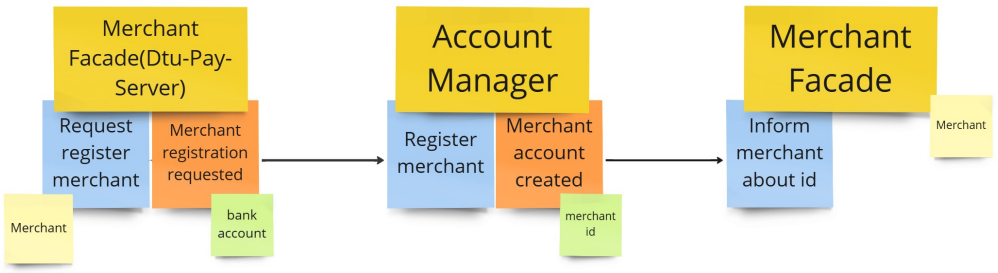

Figure 5: Merchant Registration Scenario

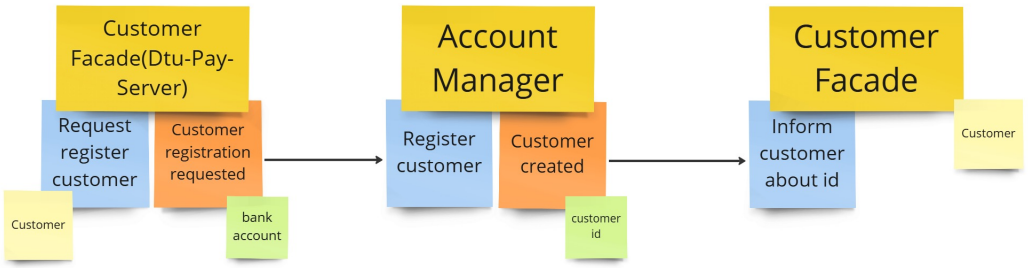

Figure 6: Customer Registration Scenario

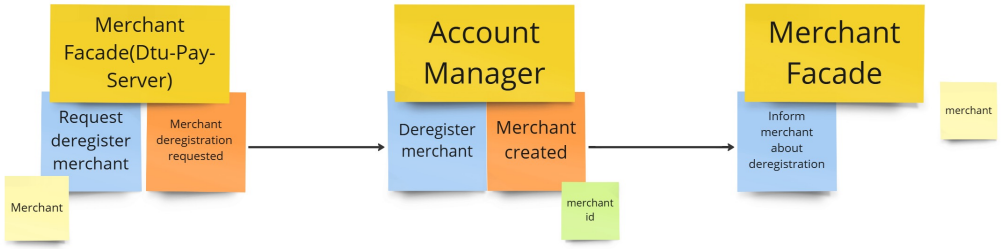

Figure 7: Merchant Deregistration

Figure 8: Customer Deregistration

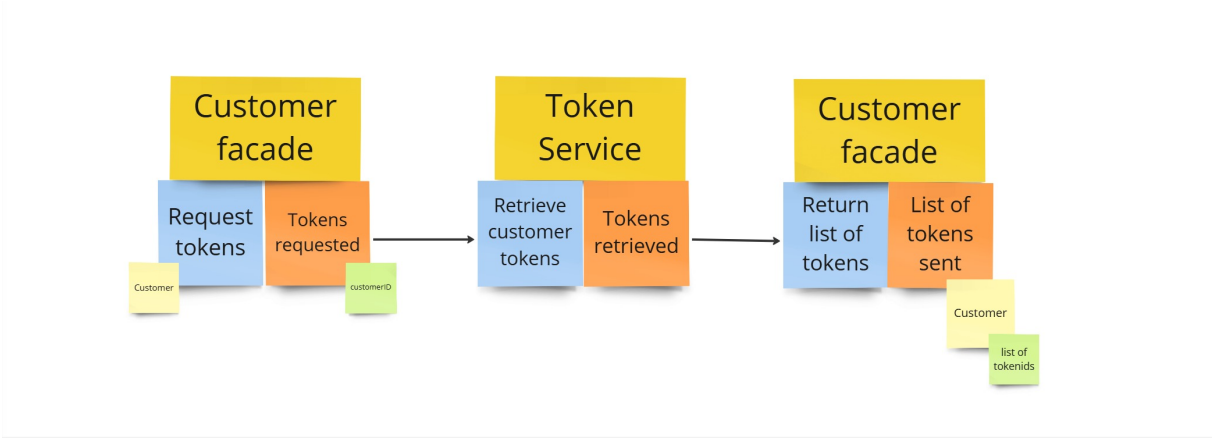

Figure 9: Customer Get Token Scenario

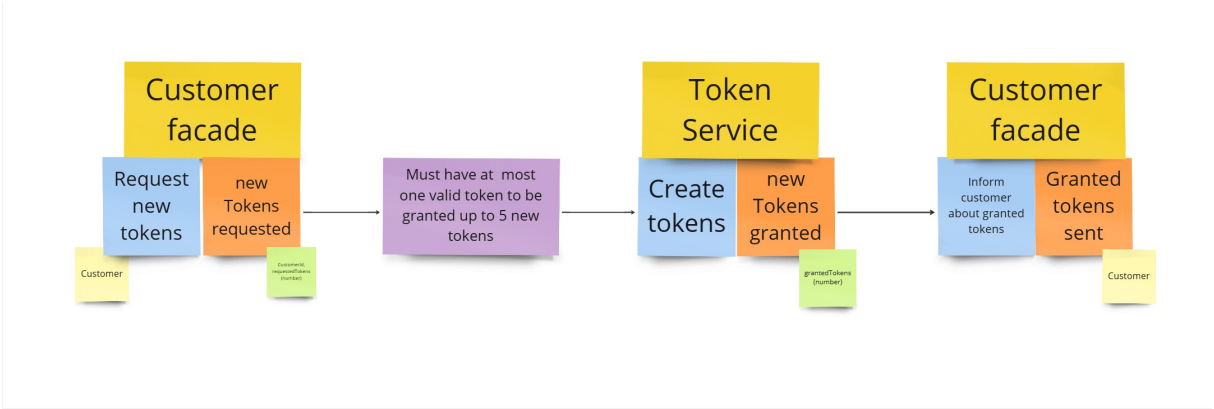

Figure 10: Customer Create Tokens Scenario

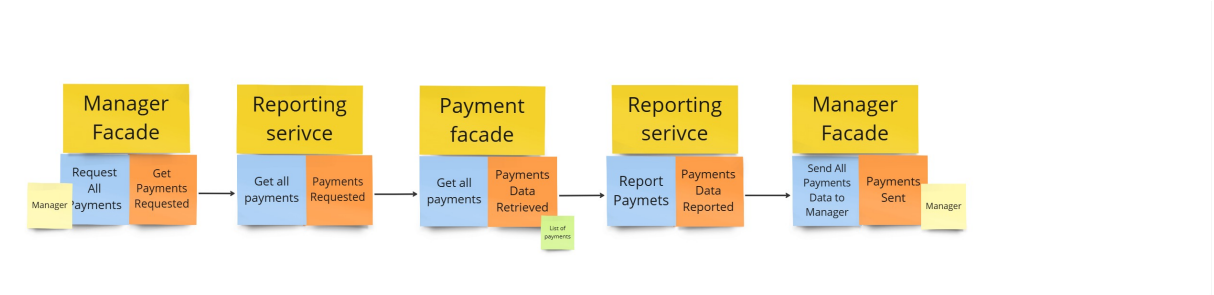

Figure 11: Get All Payments Scenario

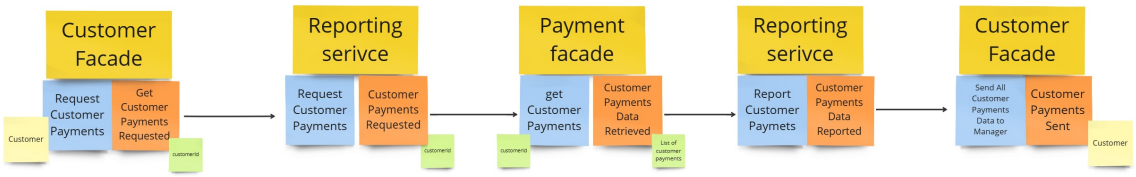

Figure 12: Get All Customer Payments Scenario

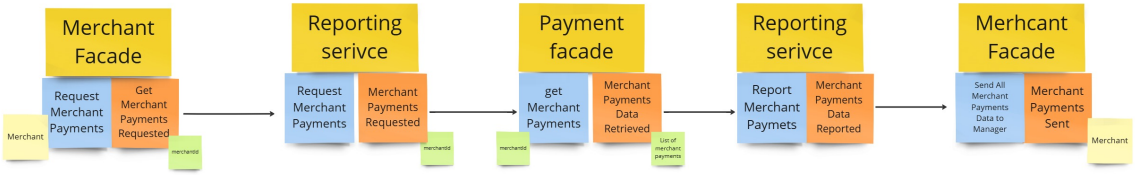

Figure 13: Get All Merchant Payments Scenario