IE-Sprint-2-Report

Ana Almeida 102618, Daniel Carvalho 102556, Group 7

May 2025

Contents

1 Introduction **1**
1.1 Documentation . . . 2
1.2 Microservice Changes . . . 2
1.3 Terraform Changes . . . 2

2 Information Flows and Business Processes **2**
2.1 Customer Management Business Process . . . 3
2.2 Shop Management Business Process . . . 3
2.3 Loyalty Card Management Business Process . . . 4
2.4 Discount Coupon Emission Business Process . . . 4
2.5 Cross Selling Recommendation Business Process . . . 5
2.6 Selled Product Analytics . . . 6
2.7 Discount Coupon analysis using Artificial Intelligence Business Processes . . . 7

3 Implementation and Deployment **7**
3.1 Kong . . . 7
3.2 Camunda . . . 7

4 Test **8**
4.1 Kafka . . . 8
4.2 Microservices and Kong . . . 8
4.3 Camunda . . . 8

1 Introduction

In our project structure, the only differences we had from Sprint 1 was the `bpmn` directory, where our business processes are stored, and the Terraform modules for Camunda and Kong.

However, in terms of terraform, since we had the limit of 9 EC2 instances, we had to merge some microservices into one instance. This leads us to have the following instances deployed:

- 3 EC2 instances running Kafka Brokers;
- 1 EC2 instance running Ollama;
- 1 EC2 instance running Camunda;
- 1 EC2 instance running Kong;
- 1 EC2 instance running both Discount Coupon and Cross Selling microservices;
- 1 EC2 instance running both Purchases and Customers microservices;
- 1 EC2 instance running Shop, Loyalty Card and Selled Product microservices;

1.1 Documentation

In the previous sprint, one area we received lower marks in was documentation. Most of it was included only in the final report. To address this, we restructured our documentation to be more accessible and segmented.

At the root of the project, we now have a `README.md` that outlines the overall structure and provides links to more specific documentation. A `running.md` file explains how to set up and run the entire project. Additionally, within the `microservices`, `tests`, and `bpmn` directories, we added a docs subdirectory containing Markdown files that explain the purpose and usage of the components in each section.

1.2 Microservice Changes

Based on the feedback we received from the first sprint, we made several changes to our microservices. As a result, all corresponding test scripts were updated accordingly to reflect these modifications.

- **Purchases**: Previously, our database was storing the shops' names and not the shop IDs. In this delivery, we have now changed it to store the IDs.
- **Cross Selling Recommendation**: This microservice originally stored recommendations in a database, which we realized was unnecessary. Since the recommendations are already published to Kafka and are only relevant to consumers (i.e., shops), we removed the database persistence entirely.
- **Selled Products**: This service underwent the most significant changes. Similar to the Cross Selling service, we removed the database layer. We also refactored the logic based on insights gained during BPMN modeling. Initially, we had a one-dimensional analysis model (e.g., purchases per product, per shop, etc.). We revised this to a more flexible approach where the analysis can be filtered by dimensions, such as location (postal code), shop, discount coupon, or loyalty card (we excluded customer because it yields the same results to loyalty card) and subsequently filtered by product. The output is now an array of products, each with its purchase count and total price, along with a timestamp of its occurrence.

In addition to functional changes within the microservices, we also introduced a `/health` endpoint to each one. This endpoint reports whether the microservice can successfully connect to its dependencies, specifically the database and Kafka, when applicable. These health checks are now integrated into our Terraform deployment process using `null_resource` blocks. After deploying each microservice's Docker container, Terraform queries the corresponding `/health` endpoint to ensure the service is up and fully operational before proceeding with the rest of the deployment.

1.3 Terraform Changes

Regarding the changes to our Terraform configuration, although previously mentioned, we'll summarize them again here for clarity.

First, due to the hard limit of 9 EC2 instances, we consolidated multiple microservices onto single instances, following the deployment layout described earlier.

Additionally, with the introduction of the `/health` endpoint in each microservice, we incorporated `null_resource` blocks into our Terraform modules. These resources continuously check for a successful response from each microservice after its container is deployed, retrying until the service is confirmed to be running. This ensures that the deployment only proceeds once all services are fully operational.

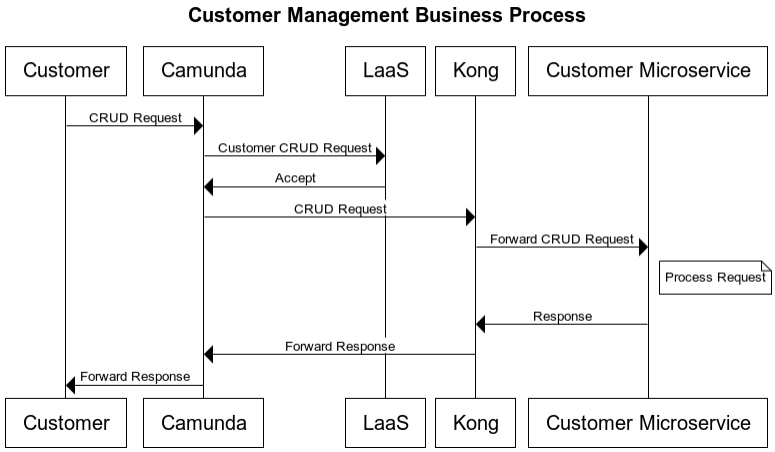
2 Information Flows and Business Processes

This section provides a detailed overview of the information flows associated with each business process and their behavior. For a deeper understanding of how each business process operates, **please refer to the respective documentation**.

The information flows are illustrated using Sequence Diagrams, created with this tool. Please note that some of the larger diagrams may appear slightly blurry due to their size.

In all diagrams, we included at least the users who interact with Camunda as part of the business process. Whenever information is exchanged between a business process actor and Camunda, one or more User Tasks are used to define and collect the necessary data.

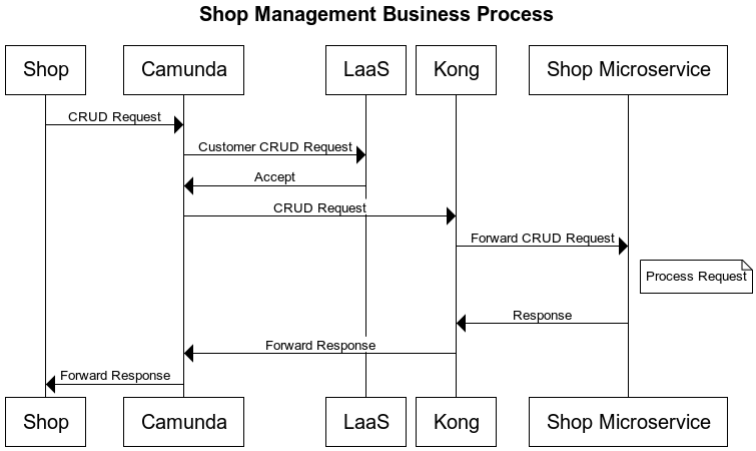
2.1 Customer Management Business Process



This business process involves two main actors: the Customer, who wants to perform a CRUD operation on their information, and the LaaS system, which is responsible for executing the request.

To facilitate modeling, we use a BPMN file named `CustomerManagement`. This process begins by asking the customer which type of CRUD operation they wish to perform and then redirects the flow to the corresponding sub-process, such as `CustomerCreation`, `CustomerDeletion`, `CustomerRead`, or `CustomerUpdate`, to handle the specific request.

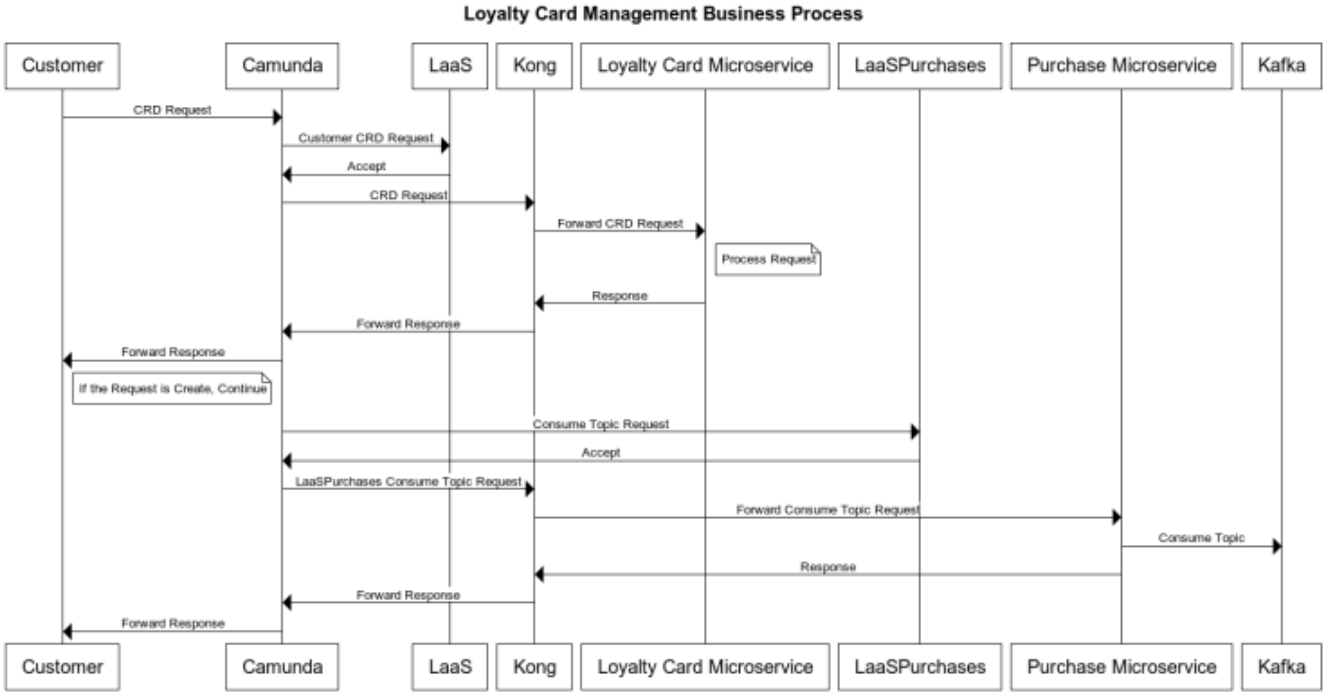
2.2 Shop Management Business Process



This business process involves two main actors: the Shop, which wants to perform a CRUD operation on its information, and the LaaS system, which is responsible for executing the request.

To facilitate modeling, we use a BPMN file named `ShopManagement`. This process begins by asking the shop which type of CRUD operation they wish to perform and then redirects the flow to the corresponding sub-process, such as `ShopCreation`, `ShopDeletion`, `ShopRead`, or `ShopUpdate`, to handle the specific request.

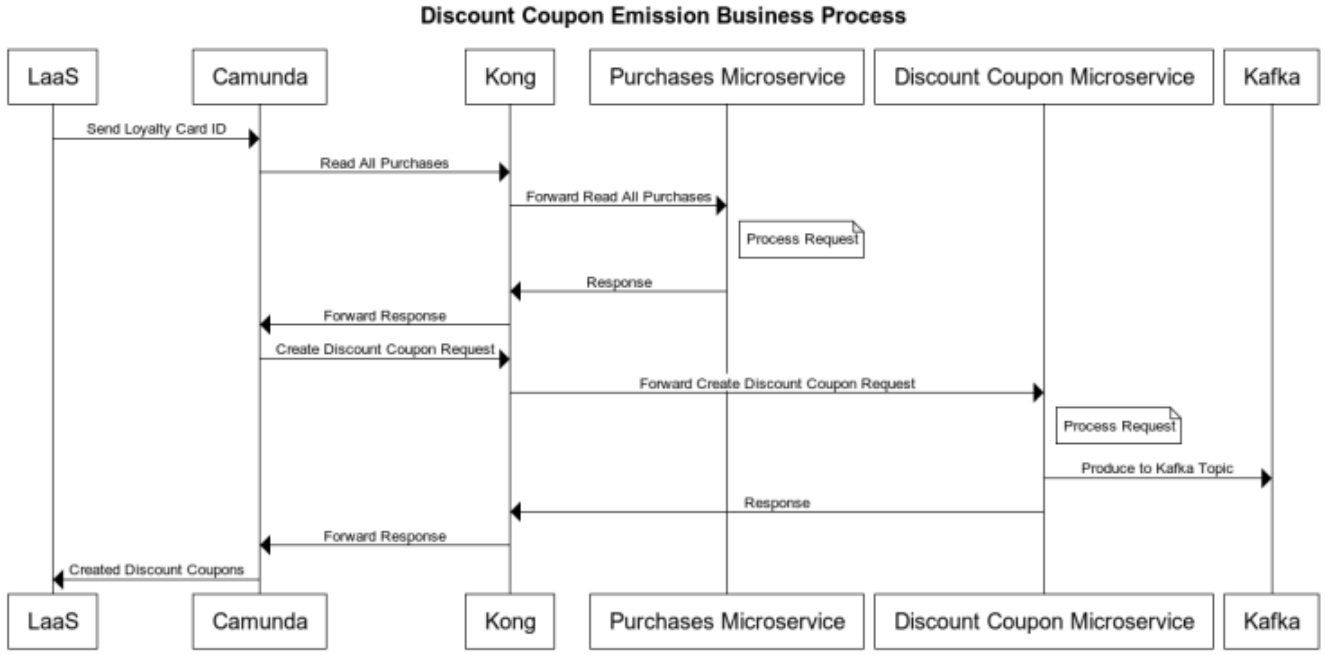
2.3 Loyalty Card Management Business Process



This business process can involve two or three main actors: the Customer, which wants to perform a CRD operation (there is no update) on its information, the LaaS system, which is responsible for executing the request and the LaaSPurchase, which only acts in the creation of a Loyalty Card by interacting with the Purchases Microservice.

To facilitate modeling, we use a BPMN file named `LoyaltyCardManagement`. This process begins by asking the customer which type of CRD operation they wish to perform, and then redirects the flow to the corresponding sub-process, such as `LoyaltyCardCreation`, `LoyaltyCardDeletion`, or `LoyaltyCardRead`, to handle the specific request.

2.4 Discount Coupon Emission Business Process



This business process involves a single main actor: the LaaS system, which is responsible for handling the request.

The process is modeled using a single BPMN file named `DiscountCouponEmission`. Since the process is customer-specific, it begins by requesting the customer's Loyalty Card ID, each loyalty card is uniquely linked to a customer, and purchases are associated with this ID.

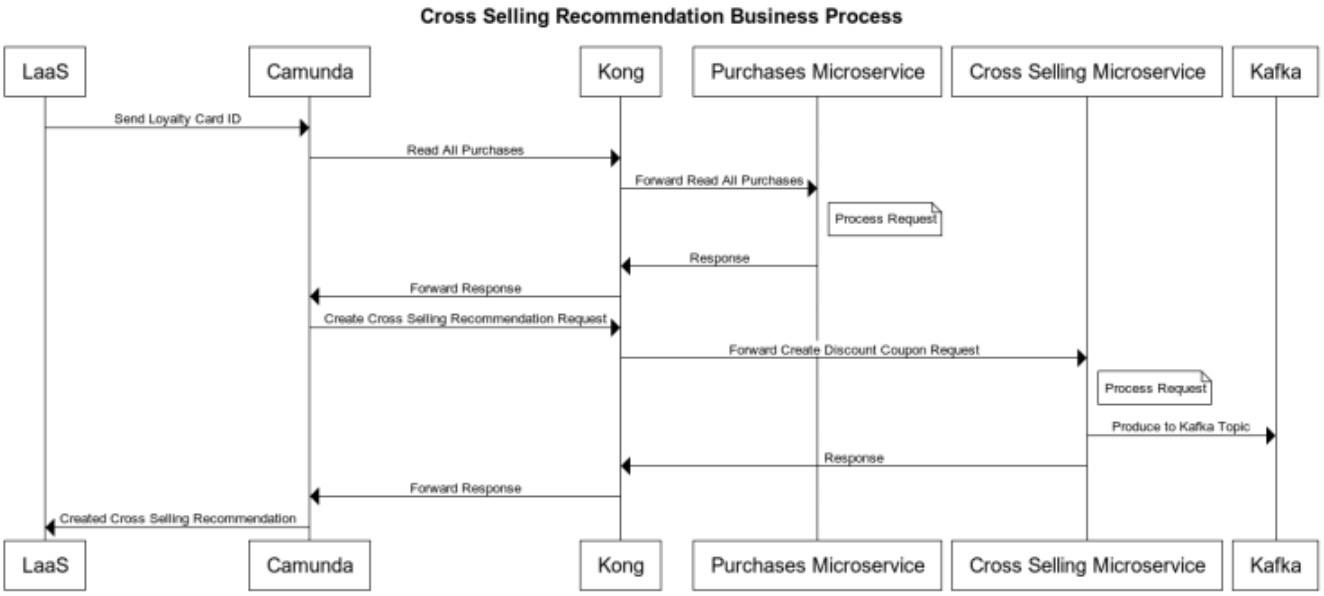
Next, the system retrieves all purchases made by the customer in the past month, under the assumption that this process runs on a monthly basis. Based on these purchases, the system generates discount coupons according to the following rules, which are applied per shop:

- **3 to 5 purchases**: a 15% discount coupon is issued to encourage more visits.
- **6 to 9 purchases**: a 10% discount coupon is issued.
- **More than 10 purchases**: a 5% discount coupon is issued, which does not decrease further, ensuring loyal customers continue to benefit.

These rules are designed to support a multi-tenant system with various shops, which may include competitors. The goal is to foster customer loyalty to individual shops through discount incentives.

Note that in this information flow, microservices related to customers and shops are not explicitly included, as they do not play a direct role in the application of these business rules.

2.5 Cross Selling Recommendation Business Process



This business process involves a single main actor: the LaaS system, which is responsible for handling the request.

The process is modeled using a single BPMN file named `crossSellingRecomendationsCreate`. Since the process is customer-specific, it begins by requesting the customer's Loyalty Card ID, each loyalty card is uniquely linked to a customer, and purchases are associated with this ID.

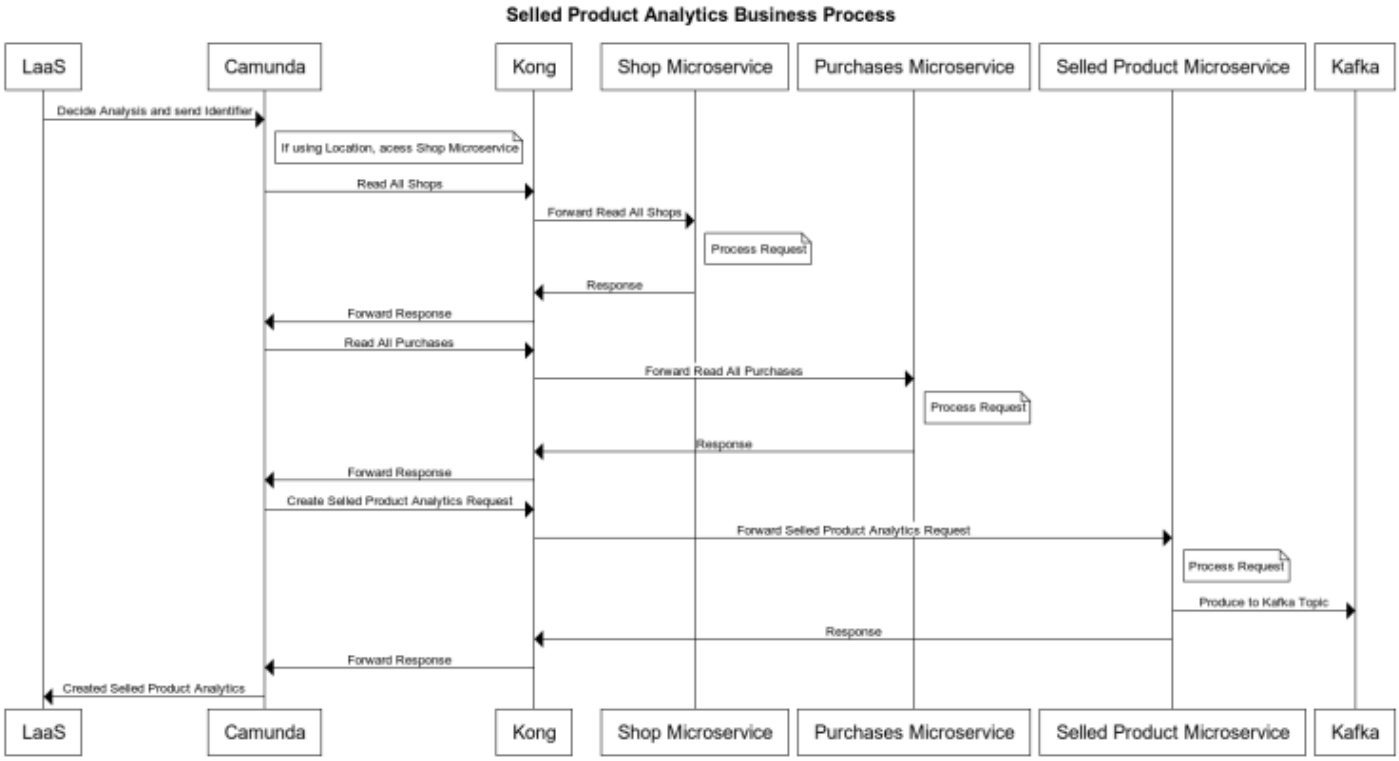
Next, the system retrieves all purchases made by the customer in the past month, under the assumption that this process runs on a monthly basis. Based on these purchases, the system generates a cross selling recommendation according to the following rules, which are applied per shop:

- We analyze the number of purchases made by a customer at each shop over the past month.
- A cross-selling recommendation can include up to six shops.
- To generate these recommendations, we select shops where the customer has made the **most** purchases and those with the **fewest** purchases.
- In the case of ties (if more than three shops share the highest or lowest number of purchases), we randomly select three shops from each group.

These rules are designed to encourage customers to visit shops they frequent less often. The aim is to promote loyalty across a broader range of shops by fostering collaboration between those with higher and lower customer engagement.

Note that in this information flow, microservices related to customers and shops are not explicitly included, as they do not play a direct role in the application of these business rules.

2.6 Selled Product Analytics



This business process involves a single main actor: the LaaS system, which is responsible for processing the request.

The process is modeled using a single BPMN file named `SelledProductAnalytics`. Since the behavior of the process depends on the type of analysis requested, it begins by asking for both the analysis type and its corresponding identifier, which will be used to filter the relevant purchases.

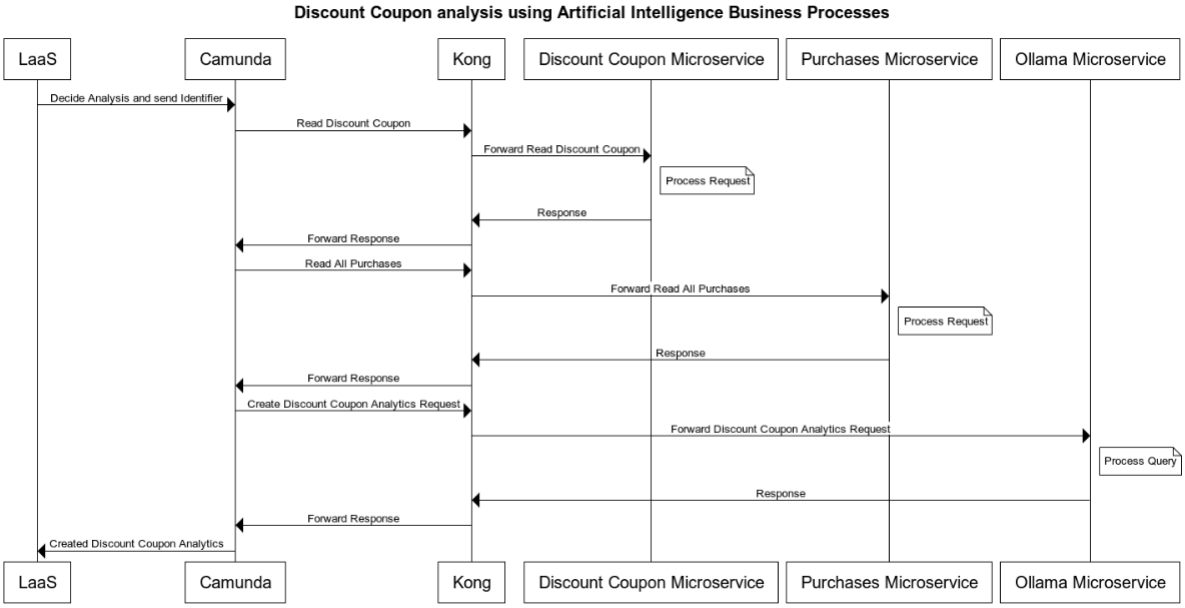
The system then retrieves all purchases and applies the necessary filters based on the selected analysis. If the analysis is based on location (Postal Code), it first identifies all shops within the specified Postal Code and then filters the purchases accordingly.

Once the relevant data is collected, the system performs aggregations over the total number of units sold and the total price for each product.

We chose to include all purchases in the system to allow Kafka Topic consumers to perform analysis at any desired granularity (e.g., monthly, weekly, daily) by comparing different instances of the analysis.

Note that in this information flow, the customer microservice is not explicitly included, as it does not have a direct role in applying these business rules.

2.7 Discount Coupon analysis using Artificial Intelligence Business Processes



This business process involves a single main actor: the LaaS system, which is responsible for processing the request.

The process is modeled using a single BPMN file named `DiscountCouponAnalysis`. We considered this process to be discount-coupon-specific in order to provide better context to Ollama, possibly leading to a more accurate analysis, since LLMs are prone to errors. Since the process is discount coupon-specific, it begins by requesting the Discount Coupon ID.

The system then retrieves relevant coupon details, such as its expiration date, the associated loyalty card, and the shops where it is valid, and uses this information to filter the set of purchases that were eligible for that coupon.

Based on the filtered data, a prompt is constructed and sent to the Ollama microservice to perform a quantitative analysis. The resulting insights are then made available to the user who initiated the business process.

3 Implementation and Deployment

In this section, we will talk more in detail about our implementation and the decisions taken in each portion of this sprint.

3.1 Kong

The most important aspect of Kong in our setup is its deployment process. Kong is deployed through a dedicated Terraform module that executes the `deploy.sh` script. This script runs the Kong Docker container and configures a Kong Service for each microservice, along with the corresponding Routes.

For each Route, we use regular expressions to match all requests that start with a specific path (e.g., /Customer, /Purchase, etc.), ensuring they are correctly routed to the appropriate microservice.

After deployment, a `null_resource` is used to run the `setup.sh` script. This script continuously checks whether Kong is accessible by verifying that port 8000 is responsive, and it only exits once Kong is fully up and running.

3.2 Camunda

Camunda is deployed using a strategy similar to that of Kong. It is managed through a dedicated Terraform module that runs the `deploy.sh` script, which pulls the Camunda Docker image and starts the container.

Following the deployment, a `null_resource` executes the `setup.sh` script. This script first checks whether port 8080 is responsive, ensuring that Camunda is up and running. Once confirmed, it iterates through all `.bpmn` files located in the `bpmn` directory and its subdirectories, deploying each one via the Camunda API. The script only completes after all BPMN files have been successfully deployed.

4 Test

In this section we will talk about how to test our project.

4.1 Kafka

The approach for testing Kafka remains the same as in Sprint 1 and will not be detailed in this report. However, the documentation includes clear instructions on how to perform these tests.

4.2 Microservices and Kong

The test scripts used in Sprint 1 for validating the microservices remain largely the same, with minor adjustments to reflect changes made to the microservices themselves.

In this sprint, we also needed to test Kong. The most effective way to do so is by sending requests through Kong to each microservice. To support this, we developed three additional scripts:

- `update-dns.sh` - Updates the EC2 instance DNS names in the test scripts to match the current deployment.
- `use_kong.sh` - Updates the test scripts with the current DNS name of the Kong instance, enabling routing through the API Gateway.
- `run_all_tests.sh` - Runs all test scripts using both the direct EC2 instance URLs and the Kong gateway.

With this setup, we maintain a single test script per microservice while still verifying that both the EC2 instances and Kong routing behave as expected.

4.3 Camunda

Currently, we do not have automated tests for validating the business processes. However, we provide a script named `populate-databases.sh` that populates the RDS database, via our microservices, with the necessary data to support the execution of the following business processes: Cross-Selling Recommendation, Discount Coupon Emission, Discount Coupon Analysis, and Selled Products Analysis.

After running the script, you will also need to populate the RDS with purchase data. This requires following the same steps outlined for testing Kafka.

For more detailed instructions, refer to the `bpmn.md` file located in the `tests/docs` directory.

The remaining business processes, which are focused on management operations (create, read, update, delete), can be tested independently as they do not depend on additional data beyond what they handle directly.