Networked Software for Distributed Systems – Project 5 Documentation

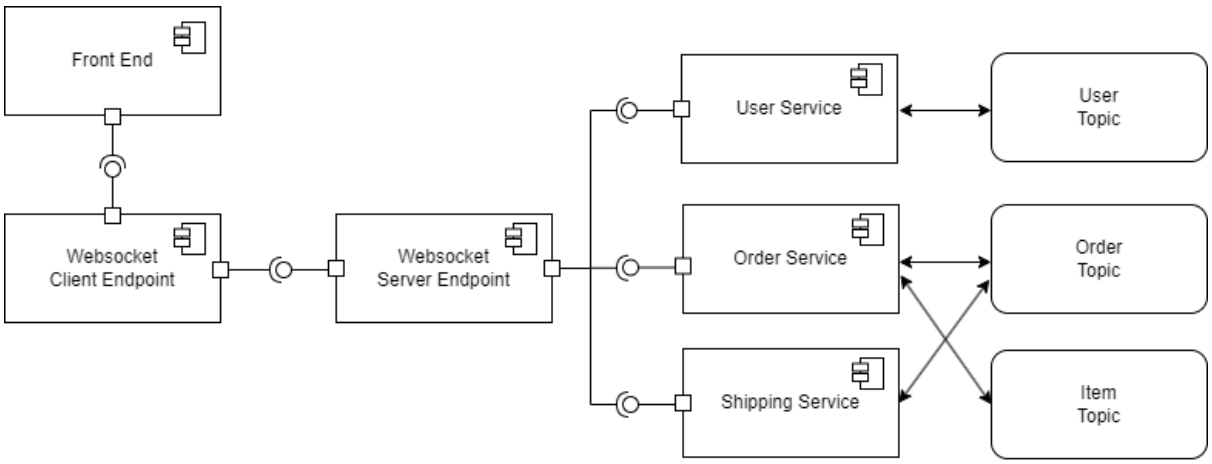
Group 7 – De Novellis Torralba Venkataraman

Introduction

This project simulates a food delivery application, where the users can be of type Customer, Admin or Delivery Employee. We have made use of the Kafka platform in order to concurrently pass messages, used to signal events and to store and retrieve relevant information such as users, orders, or food availability.

Architectural Design

Component View



- **Frontend:** The front end is where the three types of users interact with the system. This is a simple command line program that displays a different set of options depending on the user. At the beginning the user has to register or log in so that the front end will display the appropriate options. A customer has three options, placing an order, showing the order history and showing the available items. A delivery user can either update the order status or show the pending deliveries. Finally, an administrator can either update the list of available items or see the current available item list.
- **WebSocket Client End Points:** In order for the frontend to communicate with the backend and vice versa we used web sockets, the client end points were placed at the front end side with the respective socket address of the web socket servers, so that the commands typed by the users at the front end could be sent back to the backend.
- **WebSocket Server and End Points:** The server end points were the ones that receive the messages from the client end points and are also where each one of the Kafka services are initialized. This end points receive the messages and call the methods on the Kafka services that correspond to the given commands.
- **User Service:** The user service is in charge of handling all information pertaining to the user, here we make use of the Kafka tools in order to register and login users. When started, a Kafka consumer will be initialized and will subscribe to the topic “userCreation”, in order to retrieve

the existing users. Whenever a user registers, the service ensures that the user does not already exist, if confirmed, the user is created, and the service will act as a producer to send the user email as the key and the serialized user information as the value into a topic called “userCreation”. Whenever the user wants to log in, the service checks if there is any instance matching the credentials and eventually the login is allowed.

- **Orders Service:** The order service is in charge of handling the orders placed by the customers as well as checking the item availability. When started, two types of Kafka consumers will be initialized. One that will subscribe to the “orderCreation” topic and the other one will subscribe to the “itemsCheckpoint”, to recover the existing orders and the status of the items. Whenever a Customer user wants to place an order, the front end will send the information to the order service, the order service will first check the availability of the desired items, if confirmed, then it will act as a producer and write to the “itemsCheckpoint” topic the new availability of products, it will include a timestamp in order to always recover the latest list, as key a random number, and as value the serialized list of items available removing the items from the latest order. Afterwards, it will also write to the “orderCreation” topic with the customer email as key and the rest of the serialized order information as value to submit the requested order. If the customer asks for his or her order this service is also in charge of returning them to the front end. Whenever the administration user wants to update the available list of items he or she can also do so through this service.
- **Shipping Service:** The shipping service is in charge of handling the delivery of the orders, by reading and updating any new order. When started, a Kafka consumer will be initialized and will subscribe to the topic “orderCreation”, to retrieve the not delivered orders. Whenever a new order is created, this service will act as a consumer and read all the ongoing orders and store them. If a delivery user decides to update the order information such as changing the status from ongoing to delivered, the shipping service will act as a producer and write to the “orderCreation” service to notify the update, furthermore it will remove this order from its own storage since historical orders do no concern the shipping service.

Design Choices

Fault Tolerance

The services could fail and the state of the services would be lost, we have implemented an recovery method for all the user, the shipping and the order services. Since we are assuming that the Kafka topics cannot be lost, the we implemented a method in each one of the services that would act as a consumer in case there was a failure.

In the user service, after a failure the list of users and their information would be lost, to recover it the service would start polling the “userCreation” topic again from the earliest record with the seekToBeginning method, in order to recover all the users, and store them into the user service state. In the shipping service, after a failure, the orders that are currently being delivered would be lost, to recover them the service would start polling the ”orderService” topic and store the orders that are not in the delivered state.

Finally the order service, after a failure it would lose both the order list as well as the available item list. To recover them, the service would start polling “orderService” topic from the earliest record, and the “itemCheckpoint” topic from the latest record since for item availability we do not care about past availability. Since the list of order is updated by both the order service and the shipping service,

there is a version attached to the order as well to ensure that when these two services do recovery they would get the latest version of the orders.

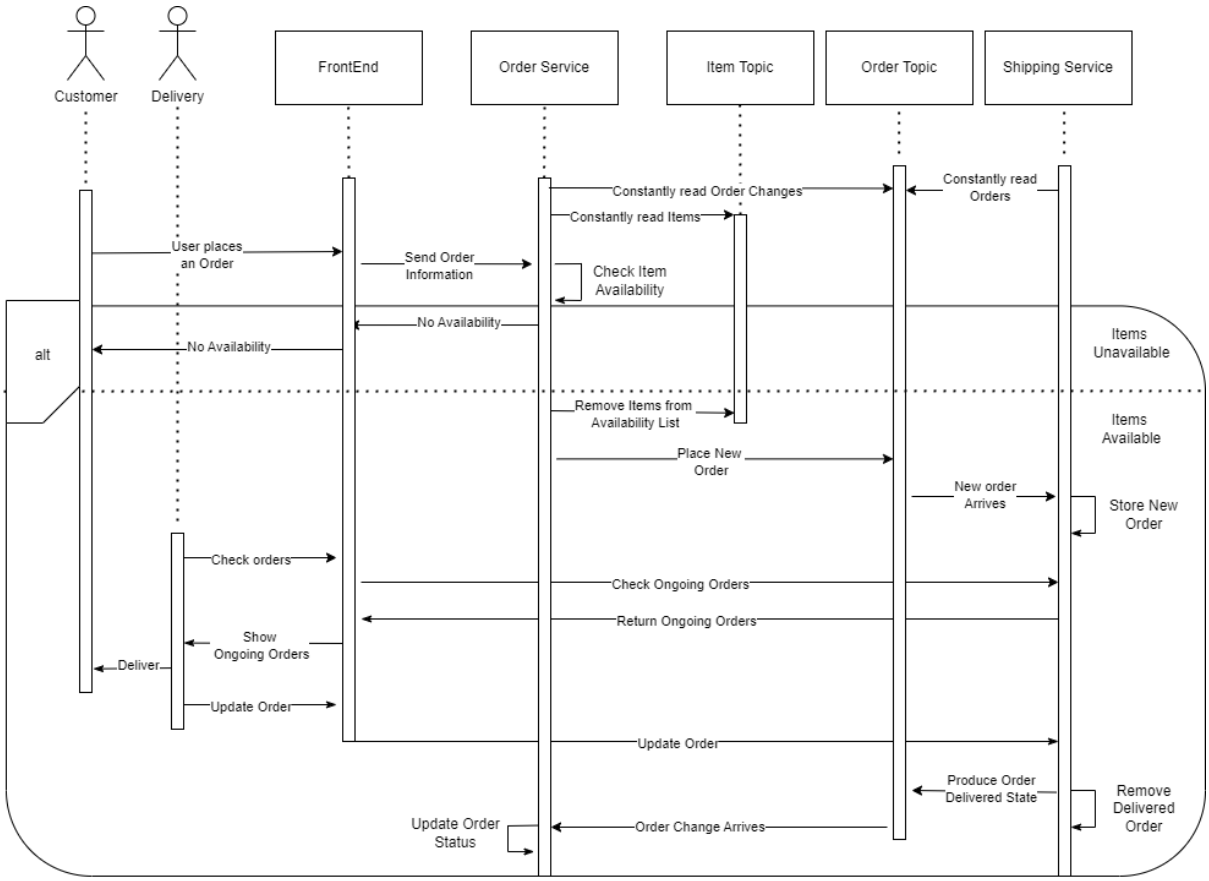
Entities

In order to organize the project better and serialize information in an easier manner whenever it was sent to any of the Kafka topics two types of entities were developed:

- User: The user contains an email which was its unique identifier, a password, an address and a type which could be either a Customer, Delivery or Administrator.
- Order: The orders are identified by a unique id number, and also include the email of the user that placed the order, the address to deliver, the status whether it is ongoing or delivered, the list of items included and the version.

Runtime View

Below is an example of the processes and interactions that happen when a user places an order and the order is delivered by the delivery man. We chose it as it encapsulates the most important functionalities of the system.



1. The services are initialized and the consumers at the order service will subscribe and therefore constantly read the item and order topics (since the order service is the one creating the orders, only updates to the orders will be read by this service from the order topic). While the shipping service will only be subscribed to the order topic.
2. Now, a user places an order through the Frontend

3. The frontend sends the order information to the order service
4. Since the order service is always reading the item topics, it has already stored the currently available items, therefore it can perform the check of item availability on its own
5. If the items are not available, a message will return to the front end for it to be shown to the user saying that the items are unavailable
6. On the other hand, if items are available then the order service will remove the items from the item list by producing a message and sending it to the item topic.
7. Then the order service will send the order information to the order topic
8. Since the shipping service is subscribed to the order topic, when the new order arrive to the order topic, the shipping service will see it and update is local current orders.
9. Then if a delivery man checks the available orders to deliver, the shipping service will show the delivery man the updated available order list
10. The delivery man can deliver the order to the customer
11. After delivery, the delivery man can update the order status through the front end
12. The front end will send the shipping service the update to the order
13. The shipping service will internally update its order list by removing the already delivered order
14. The shipping service will then act as a producer and send the order status update to the order topic
15. Since the order service is constantly reading from the order topic, when the order is updated the order service will see it and update its local order list as well.