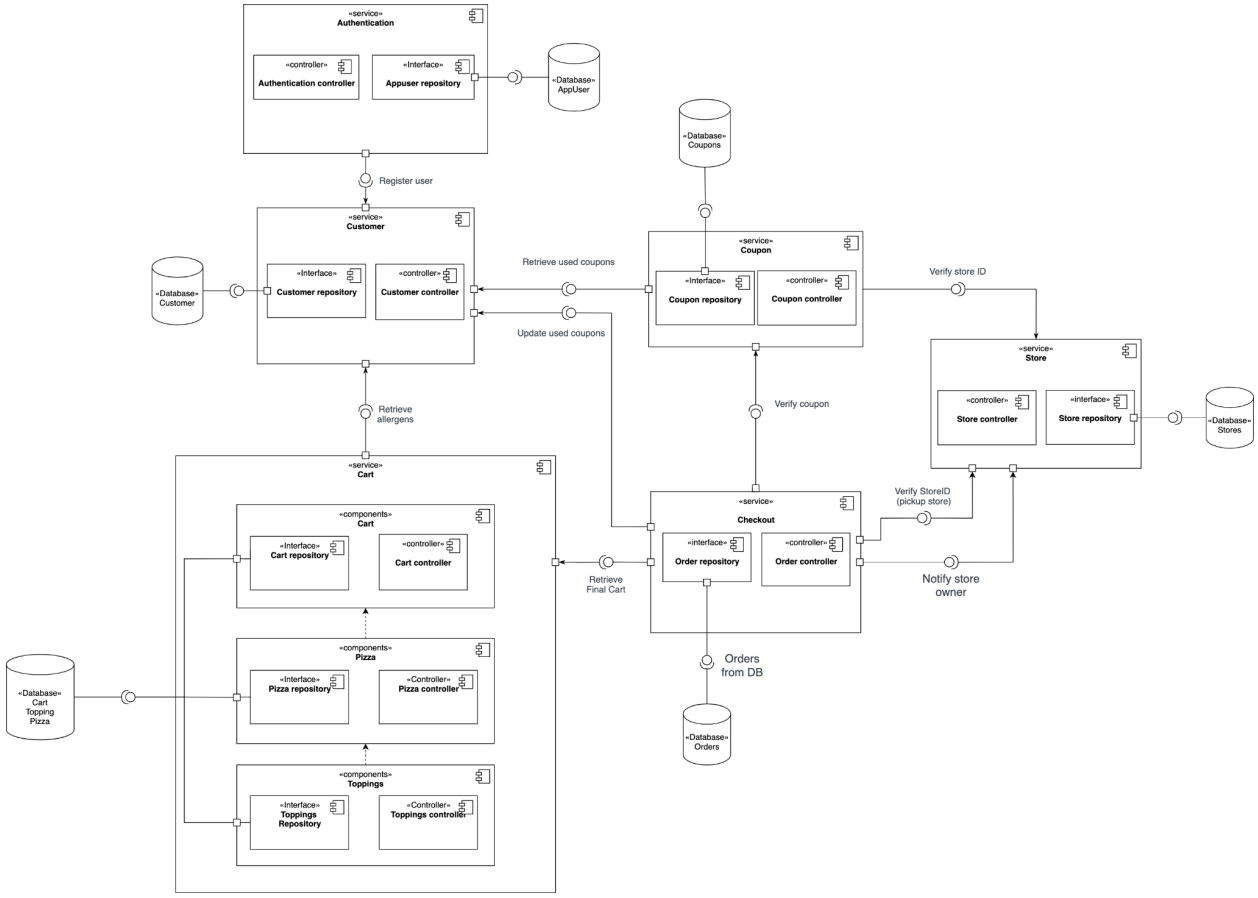
SEM GROUP 01 Assignment 1

Task 1: Software Architecture



Bounded contexts

Authentication

The authentication bounded context should allow customers to create an account with an email and a password. It will generate a JWT token that takes into account the role of the user. The JWT token will be used to access all the other endpoints in the API. Users can be normal customers, regional managers, or store owners.
The authentication service should also persist users to the database. Because of security issues we do not allow regional or store managers to create accounts, their accounts will be prefilled in the database manually when we start the application. The authentication context interacts with all the other services of the application in order to verify the identity and permissions of users.

Customer

The customer bounded context is responsible for managing the information and actions related to the customers of the system. It is where the “Customer” entity with an id, a list of allergens, and a list of used coupons will be saved. The allergens are a list of toppings the customer should be able to set for themselves by picking out of the list of default toppings that the pizza store offers. This list will later be used in the cart. The used coupons are a list of coupons that the user has already used in past orders. This list gets updated when the checkout is complete and the order is placed by the customer, while it is kept because of the fact that a user can make use of a unique coupon code only once. Therefore, if there was a coupon included in that order, it is added to the customer’s list while if an order is canceled, either by the customer themself or by the regional manager, the coupon used is removed from the list, as it is “refunded” to the customer.

Store

The bounded context of the store keeps the Store entities and their functionality. Each store has a store owner, a regional manager, and a location. The Store will be linked to a store owner in a one-to-one relation, while many stores can share a single regional manager. The reason we save the ID of the store owner is to verify whether the person has permission for alterations at this store, such as adding coupons or altering orders. This way we assure that permissions are only given to eligible people. The store has an address so that customers can be aware of where to pick up their order. The store is notified of an order placed by the checkout microservice.

Cart

The bounded context of the Cart consists of functionality related to the process of making an order from the user’s perspective. This involves every step of the ordering process prior to checkout. Specifically, Cart first provides the customer the choice of pickup location, where they can choose the store from which they’d want to order. Following this selection, Cart handles the customer’s ability to select a default pizza from the store’s selection or create a custom pizza by creating a custom configuration of toppings (selected from a list of toppings the store provides which is the same for all stores). The customer can edit or remove any and all toppings on any pizza in his cart, whether it is custom or default. Additionally, the option to filter the default selection of pizzas the customer sees based on the different allergens that they have set in their profile is given, in order to make the order process easier for such dietary restrictions. This also allows for the customer to get a warning when choosing a pizza that contains a topping they

have listed as an allergen, regardless if they have filtered the selection of pizzas or not, while using information from the Customer context.

Checkout

Checkout contains all the logic and management before the completion of the order is held. It is mostly responsible for communication. We decided to separate this part from the actual order context, as there are multiple aspects to be managed from all the user's, the store owner's, the regional manager's and the application's side.
Firstly, the user has the option to select the pickup time for his order at the store he had earlier selected. The user can from here try out multiple coupons for his current cart contents, while only the one that alone gives the lowest final price will be applied. The one applied will be updated in the customer's used coupon list. In case the order gets canceled, this is where the functionality for the coupon refund will also be done here so that the code can be used sometime in the future.
Finally, during checkout, the customer can view the price of his order by seeing how much each pizza costs, and the coupon's contribution. From the regional manager's perspective, the ability to see, track, edit, or cancel any incoming orders assigned to him is provided. When checkout is completed, it is time for the store to prepare the order so it gets notified and receives all the relative information.

Coupon

The coupon application logic is its own bounded context due to its potential complexity. The user can try out multiple coupons to reduce the price of his purchase. The vouchers are either specific to the selected store or exist in the whole region the store belongs to. The application will be responsible for checking whether the requested coupon is valid at that point in time, according to its expiry date, but also if it can be used at the store selected for the ongoing order. Finally, it needs to be verified whether the specific customer can make use of a given code since all users can only apply a unique coupon code once. After all the above checks are done for the list of coupons provided by the user, the one giving the lowest final price is applied and sent back to checkout. Note that only one code can be used per order.
It is also possible for a store owner or a regional manager to add a coupon to the database, either for his store or the region he is responsible for respectively. Right now the application offers two types of coupons, "percentage discount" or "1+1 free" coupons, but the architecture is scalable to other types.

Mapping to microservices

To map the bounded contexts to microservices, we decided to decompose by subdomain.
This follows naturally from the Domain Driven Design principle that the program should adhere to. The reason we made this decision was to have each microservice be relatively autonomous and to deliver a different logical part of the project.
Each bounded context is mapped into a microservice of its own. Thus, we have 6 microservices:

Authentication → Authentication microservice

The authentication is separate from the business logic of the pizza restaurant. It works as a service that all other microservices will use, so it was deemed logical for it to be isolated to a microservice of its own. It is a generic microservice.

Customer → Customer microservice

We decided to make the customer a separate microservice because it has separate logic from the authentication. For a customer we store the allergens and the coupons that they used. It does not make sense to put the customer in the authentication because regional managers and store owners do not have allergens and coupons that they used.

Store → Store microservice

We decided to make ‘store’ a microservice because it represents all the interactions between an actual store and the customer. Merging the store with the *Checkout* or with the *Cart* does not make sense because Regional Managers can make changes to stores.

Cart → Cart microservice

We could have chosen that the client side application takes care of the adding items in cart with a cookie for instance, but since the requirements mention cart we decided to implement the logic on the server side.
We decided not to make a microservice just for pizzas, since that would require a pizza to be verified every time a customer wants to add it to the cart. Thus we merged the idea of default pizzas and the cart because they are tightly coupled.

Checkout → Checkout microservice

We chose to have a separate microservice for checkout because we need to handle the orders that a customer makes and the pick-up time and the store required. We could have merged *Checkout* and *Cart* together because they interact a lot because they are tightly coupled. It is a big communication microservice and merging it with the *Cart* microservice will make the *Cart* microservice very complex.

Coupon → Coupon microservice

Coupon functionality was initially discussed to be included in the checkout microservice. After starting to work on the code, we came to the realization that it was taking a big part of that microservice so that it could be isolated in a microservice of its own. This also allows for more scalability on the coupon logic and testing.

Microservice Interactions

Authentication → Customer: Register user

After a user created an account the authentication microservice will create a customer profile by default with no allergens. The user will after that update the customer profile with their allergens that they have.

Coupon → Customer: The coupon will have to check the used coupons of a customer

When verifying a coupon that is to be used in an order, the coupon microservice will retrieve the list of used coupons of the customer. This will be used to check whether this user has already used this coupon or not.

Cart → Customer: Allergens

When a user adds a pizza to their cart, the cart-microservice will retrieve the list of allergens of the customer from the customer microservice. It uses this list of allergens to notify the user if they add a pizza that contains one of their defined allergens.

Checkout → Cart: Final cart

When the customer checks out their order, the Checkout Microservice will retrieve the current cart of the user from the cart micro-service. It will use the items in the cart for the order.

Checkout → Customer: Update used coupons (adding and removing)

When the order is checked out and finalized, the Checkout microservice sends the used coupon code and the user's netID to the Customer microservice in order for the customer's used coupons list to be updated. If an order is canceled, the Checkout microservice sends the used coupon code and the user's netID to the Customer microservice in order for it to be removed from the list of used coupons so that the customer can use it again in the future.

Checkout → Store: Notify store & verify storeID

When the user selects a pickup store for the order, the Checkout microservice will send this store to the Store microservice to verify that this is an actual store. The Store will then return the storeID corresponding with the storename. The Checkout microservice also notifies the Store when an order gets placed.

Checkout → Coupon: Apply coupon

The checkout receives a list of couponIDs from the user. The Checkout microservice then sends the list together with the order to the coupon service. There the final price will be calculated. And the price and used coupon will be returned.

Coupon → Store: Verify store owner

When a store owner tries to add a new coupon, the coupon microservice communicates with the store microservice to verify that his netId corresponds to that of the store owner he claims to be.