Annis's Pizza wants a system to enable their customers to order pizzas online. The system is required to have complete functionality for the customer ordering, as well as functionalities for other management roles such as the store and regional manager.

The most convenient system to implement the application is using a design involving microservices, and a gateway that connects the client to all these microservices. This design choice enables the microservices to be protected from outside access and users will have one easy way to use all their functionalities.

The microservices at-hand are *User*, *Basket* and *Order*. The customer first goes through authentication and logging in, which is followed by creating and populating the order basket. After checkout, orders can be monitored by the users of the system.

Aforementioned microservices are based on five bounded contexts: Authentication, User, Basket, Coupon, Order.

The Authentication and User contexts are encapsulated by the User microservice; since they have extremely close functionality, we considered that there was no need to make these two separate microservices.

A similar logic was applied to the Basket and Coupon contexts. Coupon has a small role in the ordering process, thus we thought that including it as an additional component within Basket was the easier and more organized decision.

This document goes in-depth to the microservices and the total functionality.

User

Anni's pizza wants to store all the customers in their app. The User microservice is responsible for that. It includes data such as allergies, roles, and functionalities such as registration of new customers and authentication.

This microservice will be used upon entering the program/gateway. Users are given a choice to either register or log in. Creating a new account is only available for customers, as the creation of store and regional manager users cannot be completed without the rights and other details required.

A connection with the user database will be made to verify the username-password combination. In case no user with such a username exists, the user will be prompted for invalid credentials. Otherwise, a token will be sent back to the user if

the authentication is done successfully. From now on, every time the user interacts with a microservice they will send this token along. This token shall contain all the info needed of customers by the other microservices

Upon creation, if the user is a customer, they will be asked to select from a list of allergies. These selections will be stored in the Users microservice database as a wrapper class that has a *List[Ingredient]* as an attribute. From this list, the customer will later be warned if he/she adds a pizza, containing one or more of the ingredients. The user can also see their stated allergies.

If a customer chooses not to add any allergies we consider them without any allergies. If the user has already created their account and has forgotten to add allergies, they can still go to the allergy selection menu through the microservice.

Basket

After login and checking out the menu, it's time for the customer to create their order. From this point on and until the final checkout, the Basket microservice provides all necessary functionality. The Basket microservice is used to handle all functionalities regarding the basket of the customer, which hopefully is full of delicious pizzas. The microservice handles *Pizza*, *Ingredient* and *Coupon* entities with their repositories, as well as the *Basket* object which joins them all.

Functionalities of the microservice involving the basket of the customer include:

- Creating a basket (takes place the first time a pizza is added)
- Adding a pizza from the menu to the basket
- Adding a custom pizza to the basket
- Removing a pizza from the basket
- Applying a coupon to the basket
- Calculate the total order price

On the other hand, the microservice also has a separate set of functionalities —specific for the use of system or responsible roles—to handle the databases. The customer can:

- Retrieve the pizza menu
- See all available ingredients (used when building a custom pizza)

In addition, functionalities exclusive to the Store are:

- Adding new pizzas to the menu
- Adding new ingredients available

All entities are dependent, meaning the validity of one is required for the handling of the other. For example, it's checked whether every ingredient is actually available when adding a new pizza to the menu, or every pizza is checked for a duplicate before adding it to the database.

The Basket microservice also handles additional requests of the customer such as filtering out pizzas with allergenic ingredients before the display or checking the selected pizzas for allergens before checkout.

Anni’s Pizza also allows **customers** to **apply coupons.** When a customer enters the coupon code, the coupon is first verified in the database. The reduction brought by the coupon is then calculated. Anni’s Pizza only allows one coupon to be applied per order. Whenever a coupon is applied, the controller compares the prices of the currently applied coupon and the new one and applies the cheapest one.

There are three types of coupons: discount coupon, buy-one-get-one-free coupons, and the custom coupon - which is a feature to be added in the future.

The **Store** can **create and delete coupons**. When a store creates a new coupon, they provide the coupon code, the type of the coupon, the rate in case the coupon is a discount coupon, and whether the coupon is one-time-use only or not. If it is, the coupon that is applied gets deleted automatically. In the other case, the store manually deletes the coupon by providing the activation code.

Order

Annis’s Pizza wants their users to be able to place orders. They also want users to be able to cancel their own orders and they want stores and regional managers to respectively check off and cancel orders made by users. To deal with these functionalities the Order microservice has been created.

The Order microservice is responsible for handling orders that have been submitted through the Basket microservice. The main functionality is to direct the flow of orders, in contrast to Basket which composes the orders. All the requests that it can receive are related to the flow of orders:

- Enqueue order
- Finish order
- Cancel order
- Show order

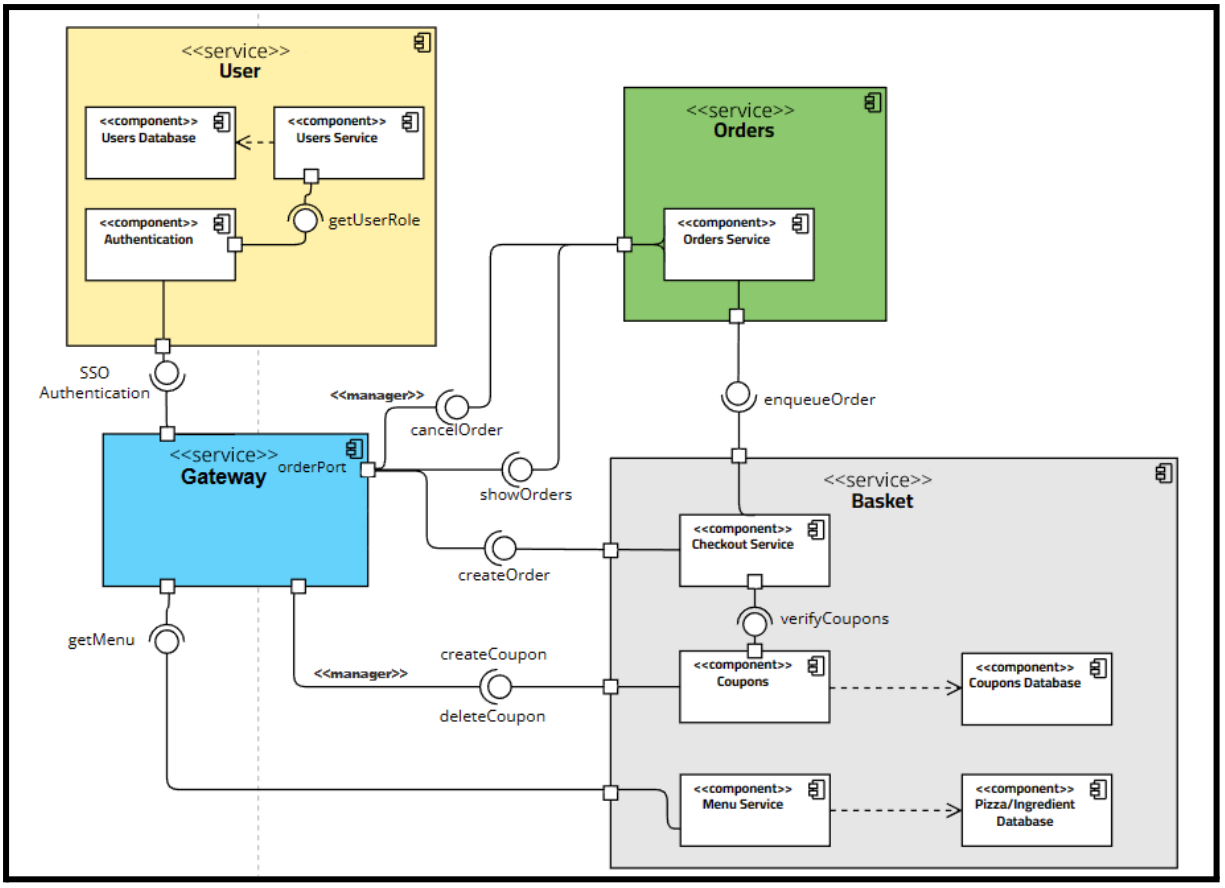
When customers order pizzas, the Enqueue functionality is used. This receives an order and adds it to a list of orders, sorted by the time the user selected for it to be finished. The collection of orders is then accessed by using “Show order”, which is how users get to see their pending orders and stores see the orders they have to deliver.
All types of users are authorized to make use of this function, but all get different results:

- **Customer** can see their own orders and cancel them until the last 30 minutes before delivery.
- **Store** can see and check off all orders made at their store.
- **Regional manager** can see and cancel all orders.

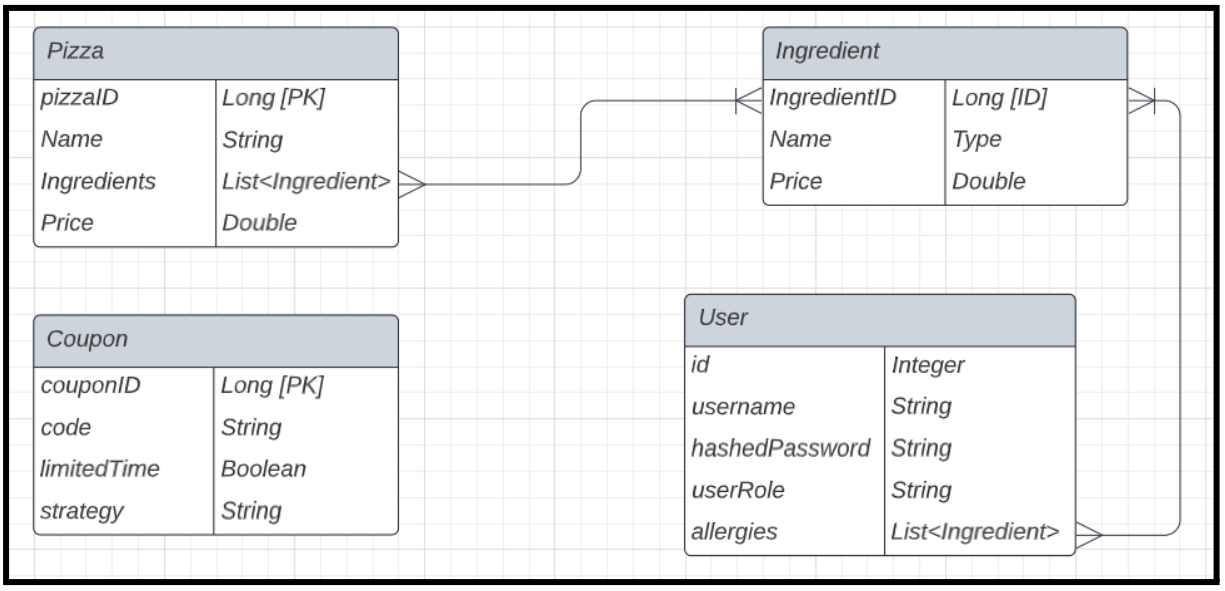
Both canceling and checking off an order removes the order from the collection. In the case that a regional manager cancels an order, the user that submitted the order will be notified in both cases.

Conclusion

Overall, as explained above, all functionalities and data storage required to build the application desired by Anni’s Pizza restaurant is satisfied by the design choice at hand: 3 microservices that are intricately divided for their specific purposes. With the Gateway building a clean communication for the user to use the system, all subsystems communicate properly for the application to function as desired.



1) UML Diagram



2) Relational database schema