https://github.com/igormatos02/checkmarxtest.git

# The Restaurant

Welcome to Checkmarx,

your task for today is to implement an application to run our new restaurant.

Your application should provide the following functionality:

1. Take an order from table with customers
   1. Table can have 1 customer or more
   2. Customer can choose one or more dishes from the menu
   3. Each dish can have different price and preparation-time
   4. Table order includes orders from all customers
2. Send the order to the kitchen by a waiter
3. In the kitchen a chef is working on the order
   1. The chef can work only on one order every time
4. Once the order is ready, the order should be delivered to the table
   1. Order preparation time calculated by the preparation-time of dish with the longest preparation-time included in the order
   2. When the order is ready one of the waiters should be notified and serve the dishes to the table
5. After the waiter serve the dishes, he’s also giving the bill to the table
6. Add an option to add more chefs to the restaurant (although currently they have budget only to one).

More Chefs can work on DIFFERENT orders in the same time and reduce waiting time to the customers

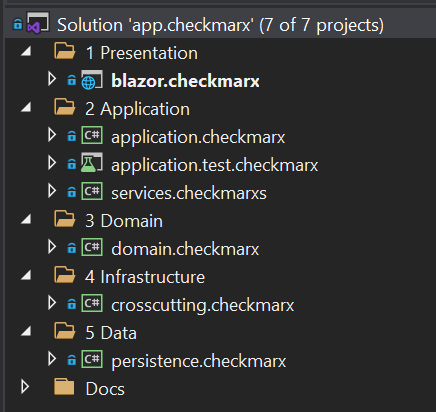
## Step 1

Provide a design of your solution and the architecture of the program.

CQRS with DDD

The application was designed to be scalable and easy to maintain.  
The code is decoupled following SOLID principles as classes with single responsibility and injected by dependency.

The application is divided by the following projects.



1 – The presentation layer is made using Blazor server side to create user interface.  
This technology uses SignalR.

2 – The application layer is composed of a project for services as RabbitMQ and another one for the CQRS implementation. A test project is also added to test the commands and queries.

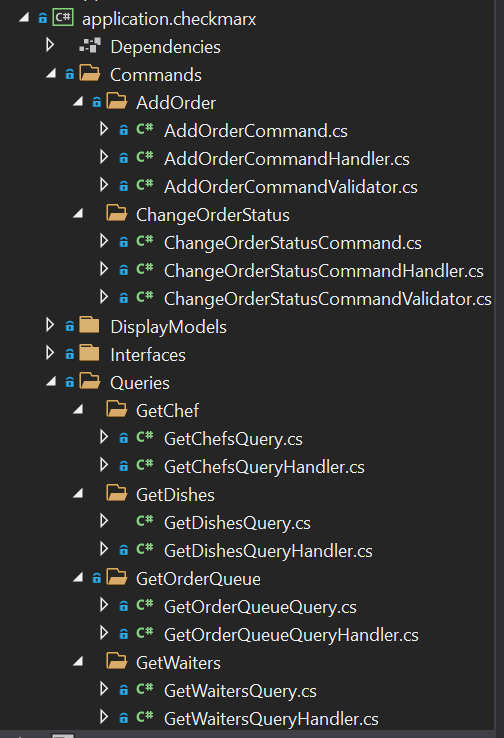
3 – The domain Layer is where we define our entities.

4 – The infrastructure layer is where we have the crosscutting parts of the whole application

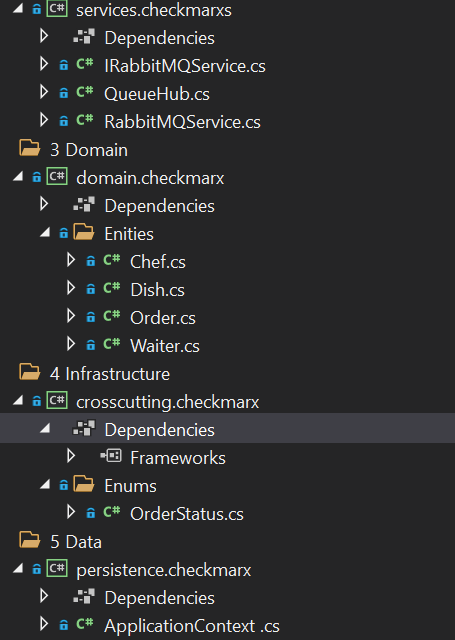
5 – The data layer is where we persist the data.

OBS: The project was not divided into UI and API to simplify this small project, but because the parts are decoupled it is very easy to do so if needed.

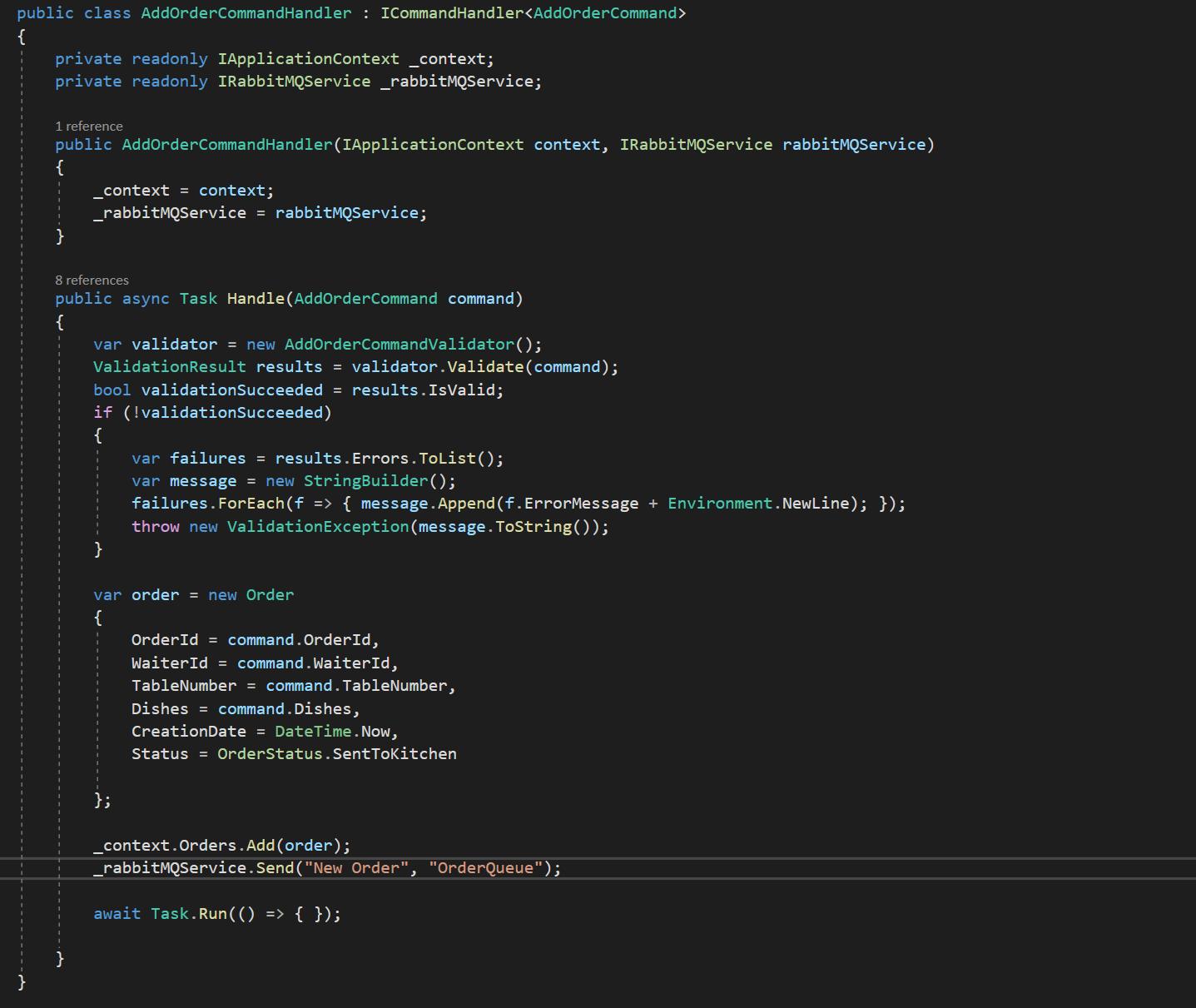
CQRS implementaion



Other Inner Classes

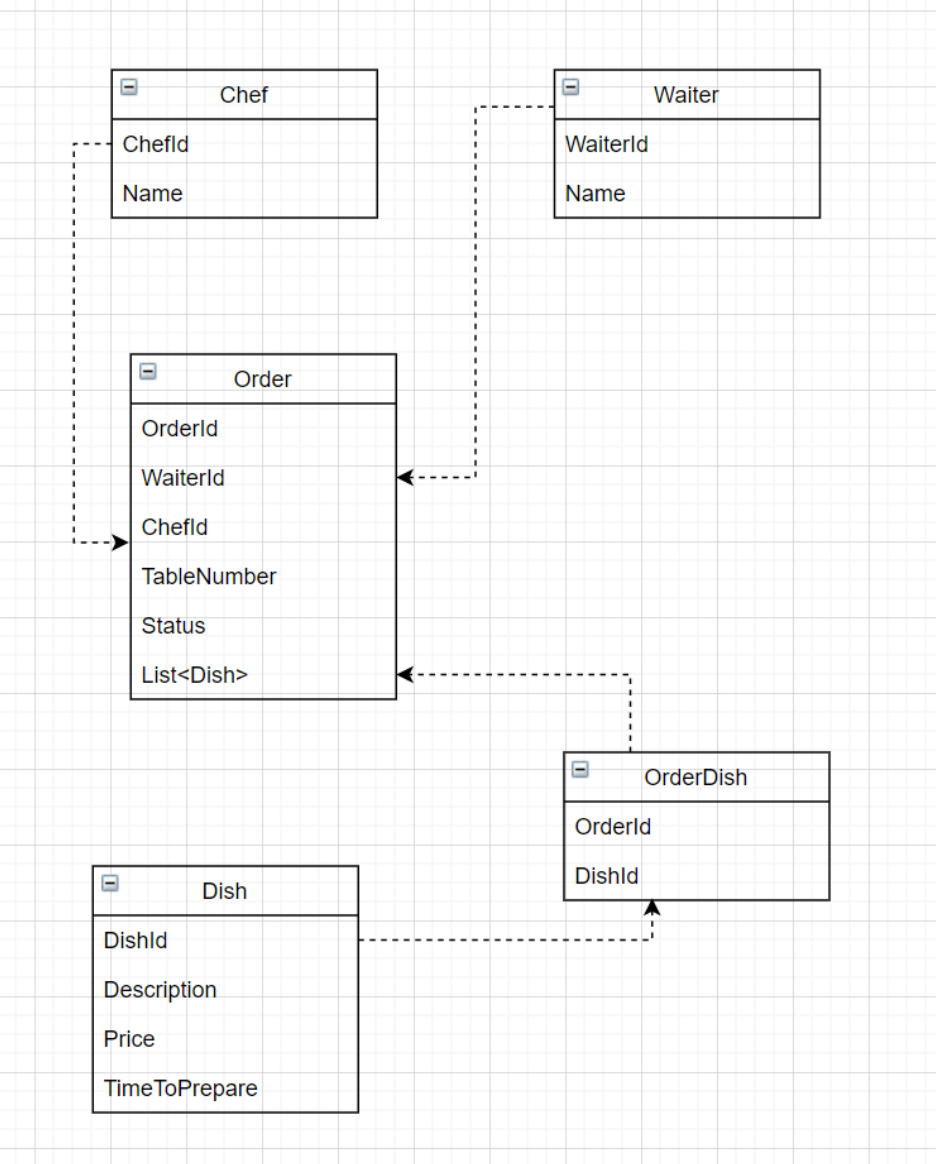


Command Handler Example:



Show which entities participate in your solution and describe the flow of the solution

The ENTITIES:



The identified entities are:

1 – Chef

2 – Waiter

3 – Order

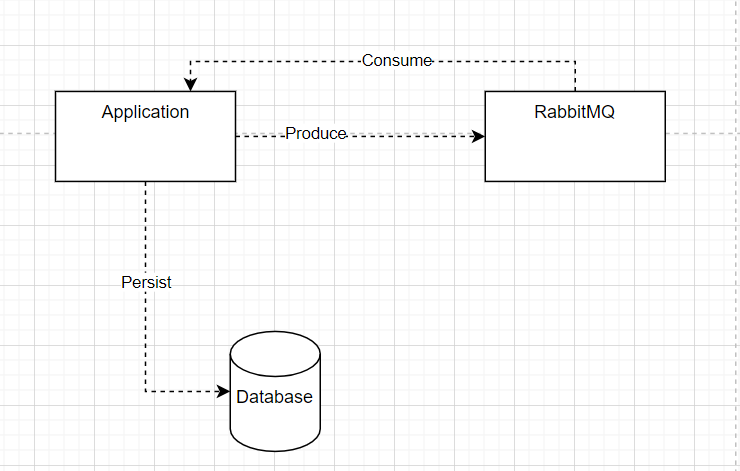
4 – Dish

5 – OrderDish

The table could be an entity if the restaurant business scale, but in this case, I used a table number knowing that this is a small restaurant.

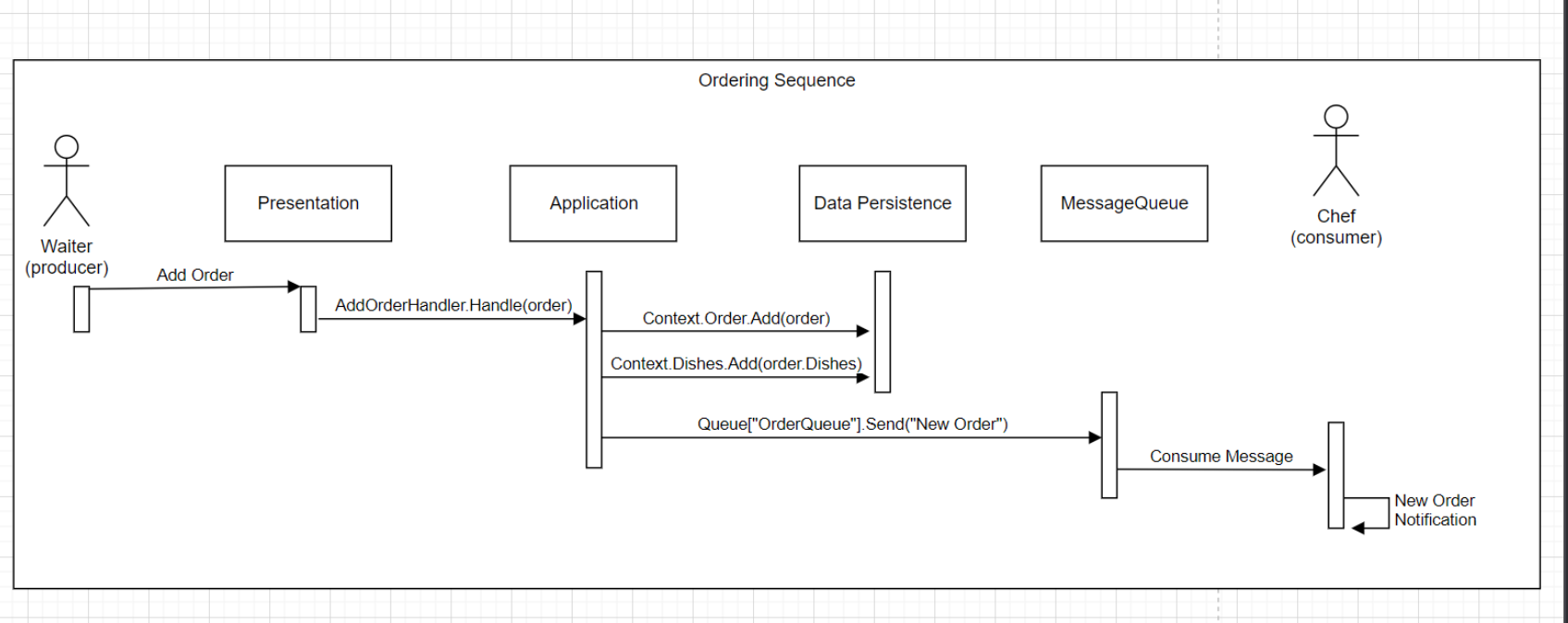
The customer is not an entity because there are no requirements or rules for separated customers.  
This could change if new requirements emerge to control number of customers, etc...

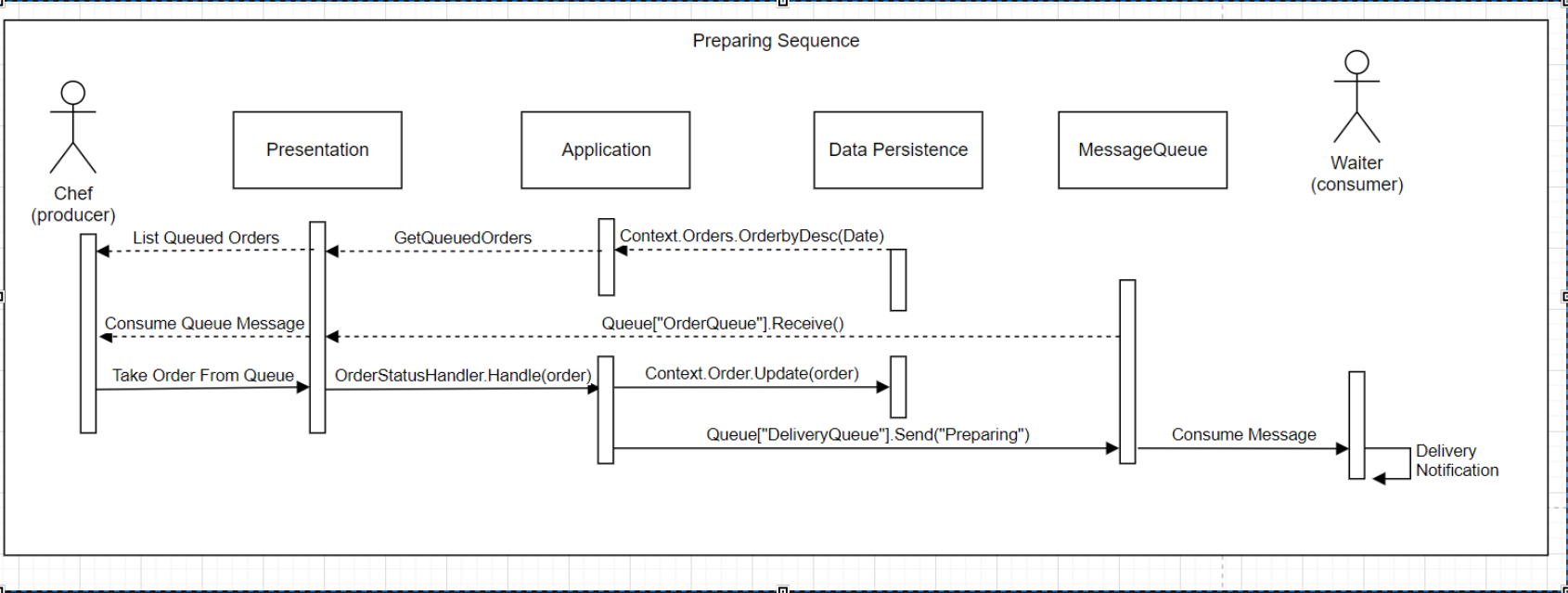
The FLOWS:

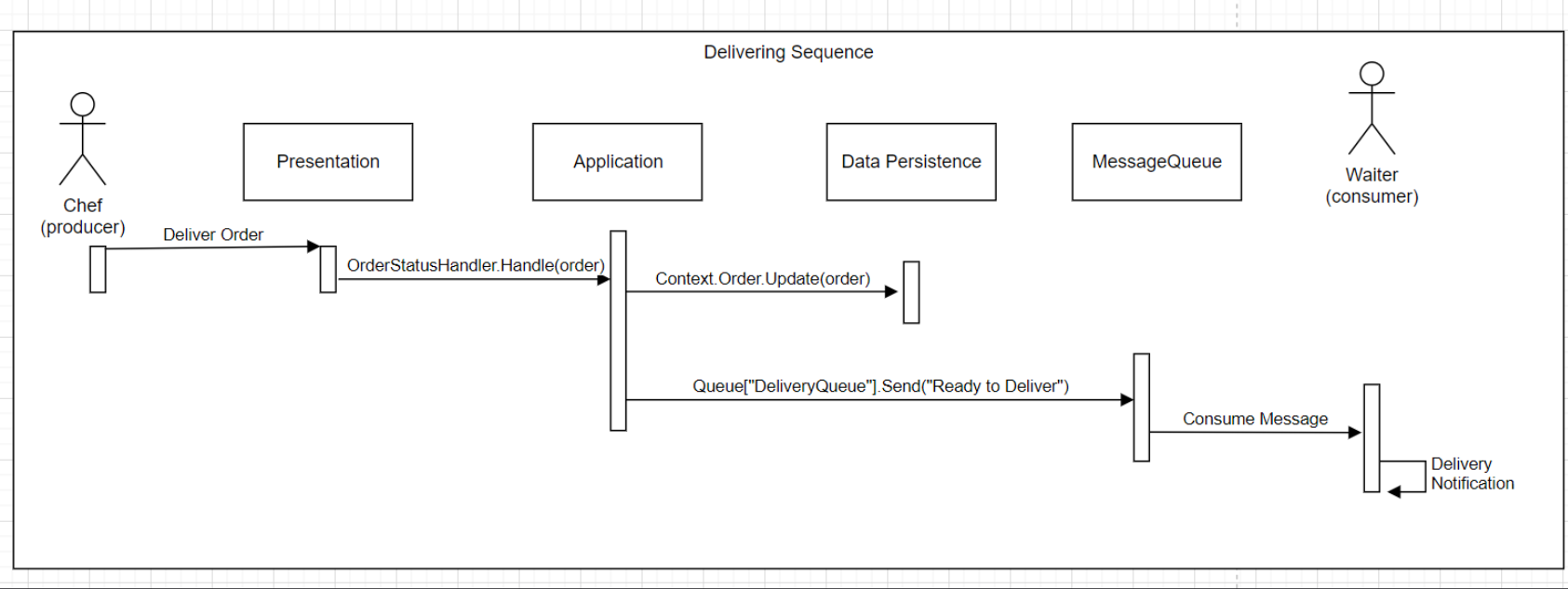


As part of the solution, a message queue as kafka or RabbitMq could be used to notify waiters and Chefs on the flow events.

SEQUENCE DIAGRAMS



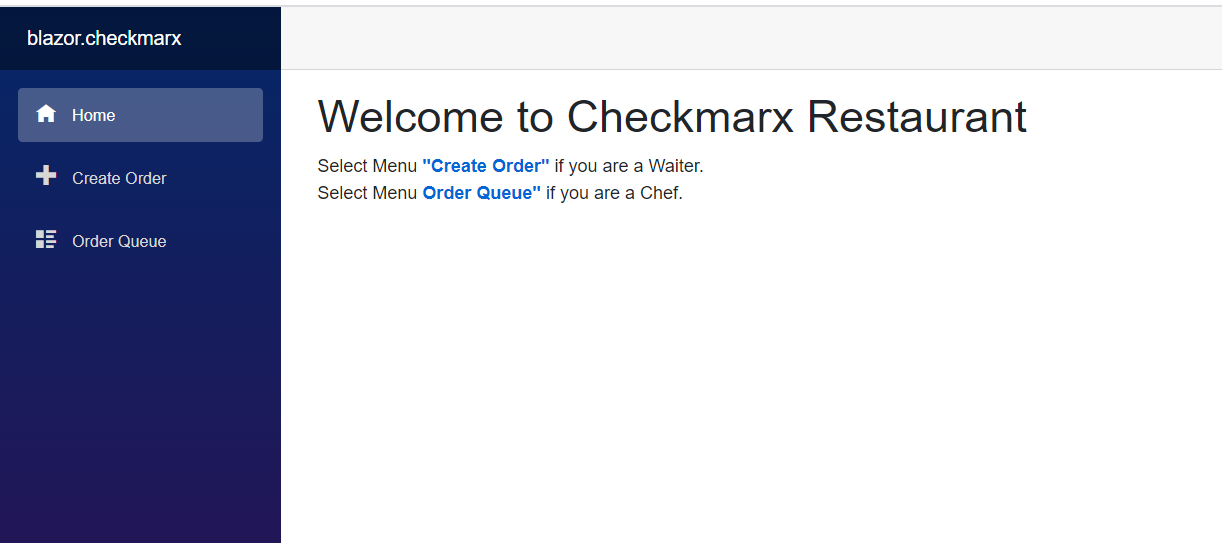




## Step 2

Implement the program.

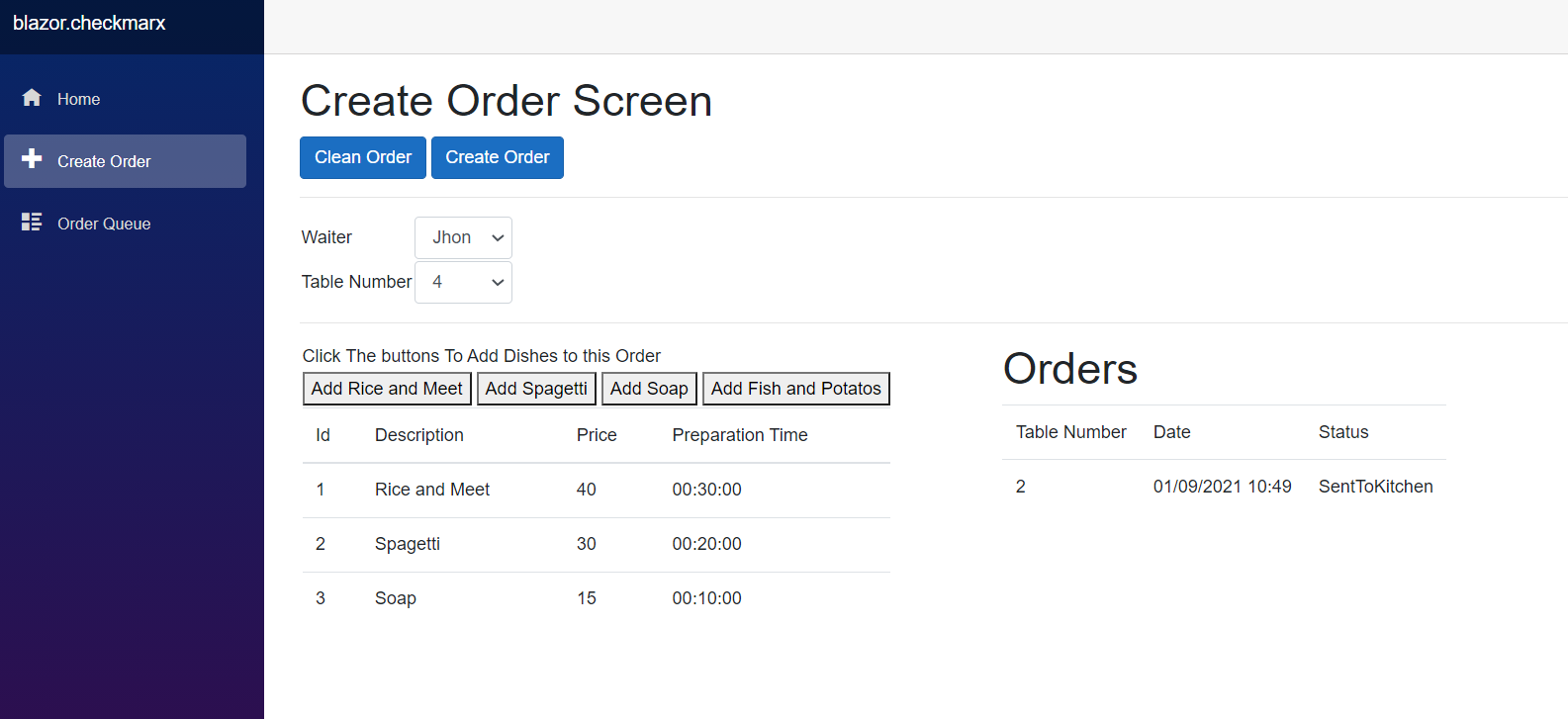
You can implement the user interface to demo the flow in every way you want (command lines, basic UI, etc.)

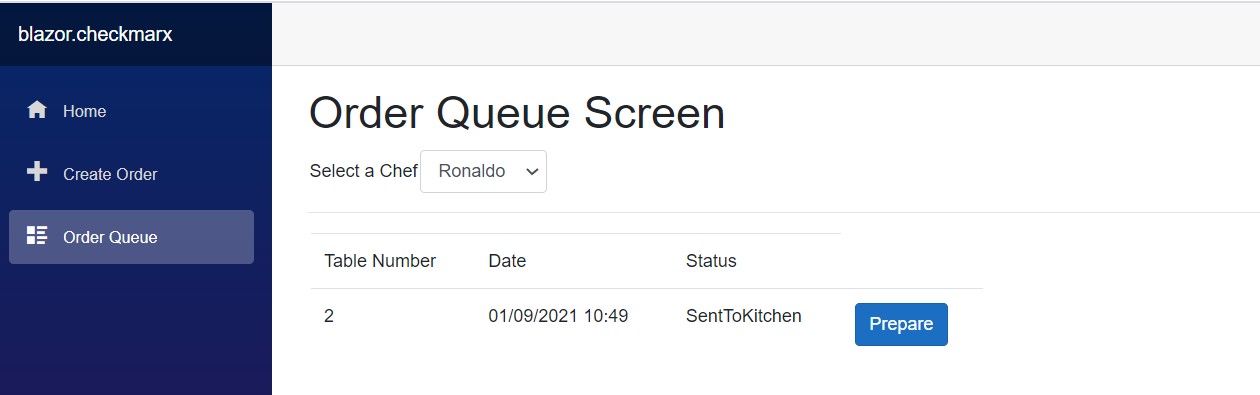


2 screens were created to the restaurant application.

1 – Create order, where waiters can add dishes to an order on a specific table and submit to the kitchen and also change the order status to delivered and closed after paid.

2 – Order Queue, where the Chef can see the orders summited and change the status to preparing and then to ready to deliver.





The chef can only change the status of one order each time from the queue.

## Step 3

Describe which automation tests you would add to your application

This application would could have 3 kinds of test.

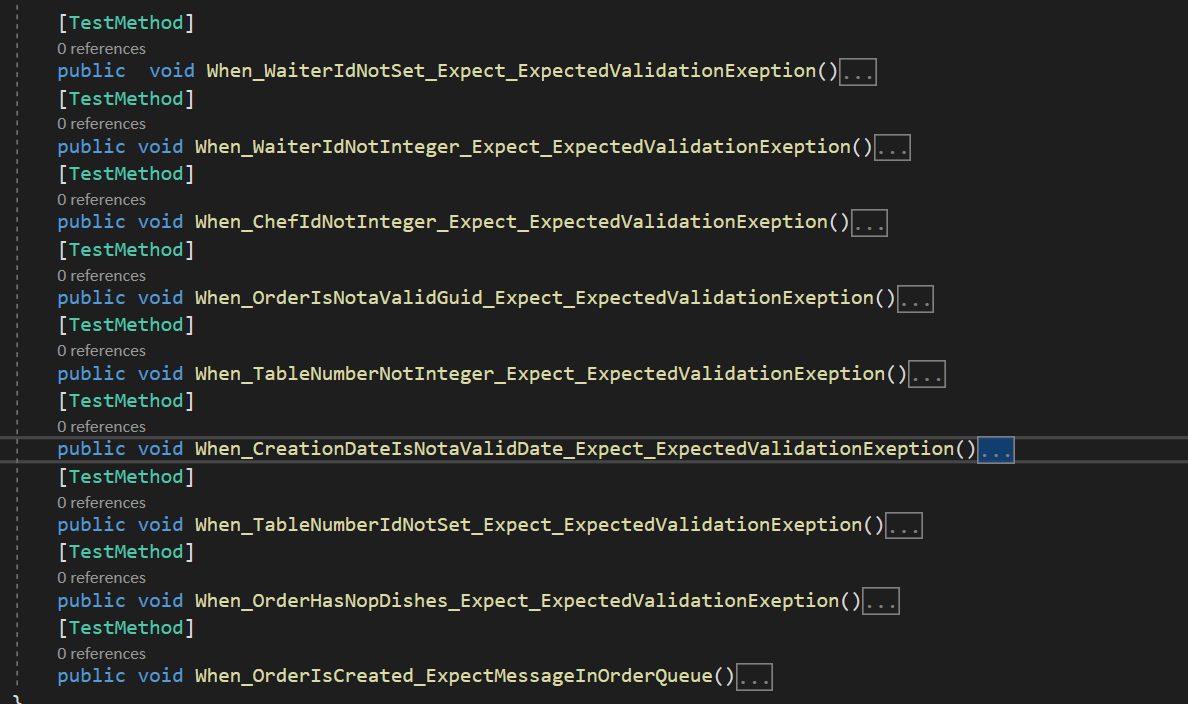
1 – Integration Test to test the interaction between components

2 – Unit Test to test the rules

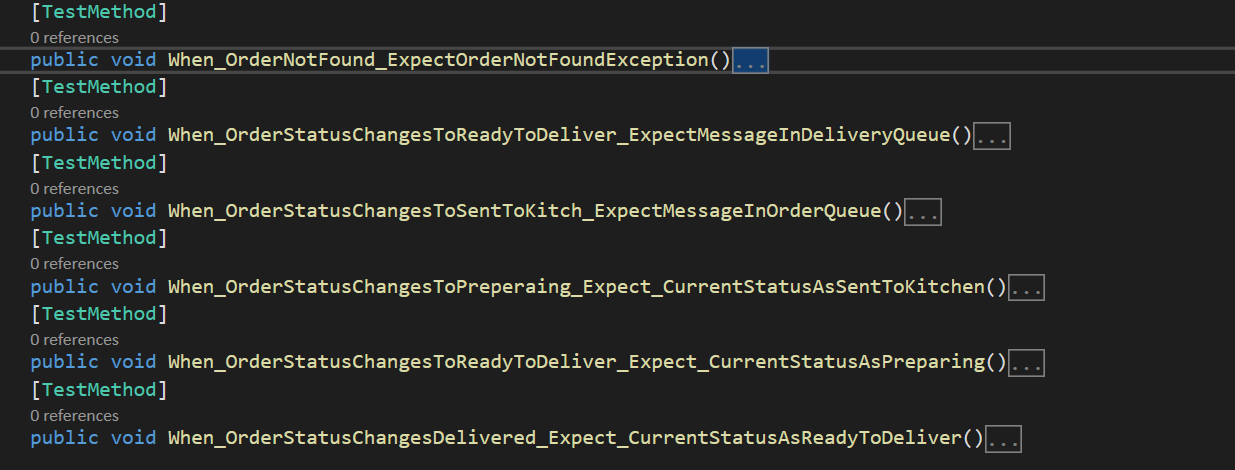
3 – UI Test like Selenium to test if the interface is as specified

**However**, this is a small application, and for this I decided to create only a Unit Test

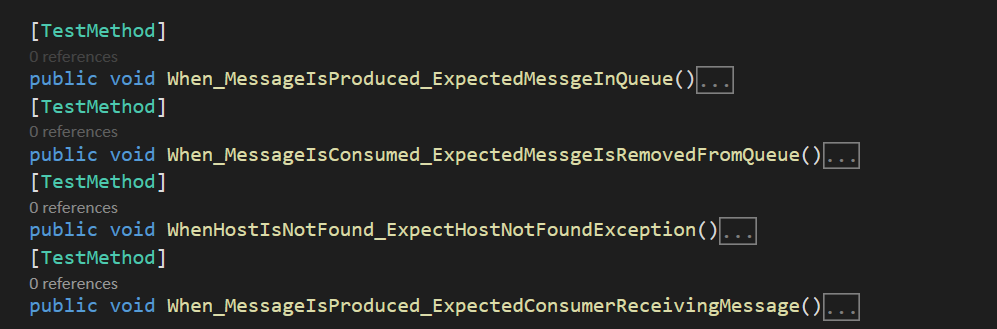
AddOrderCommandTests:



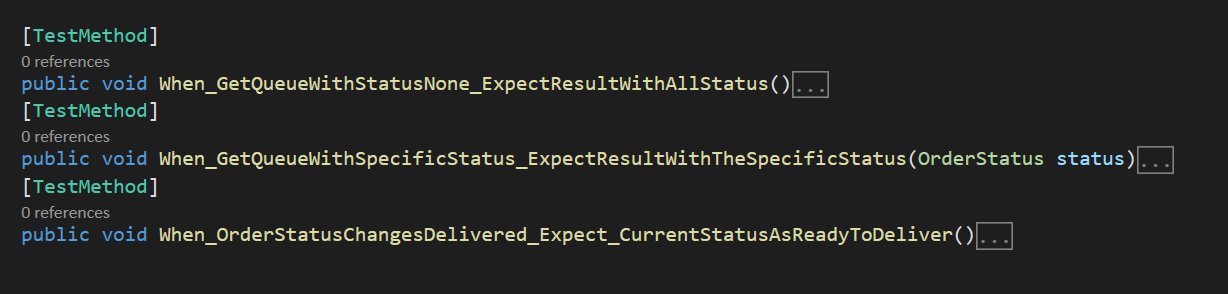
ChangeOrderStatusCommandTests



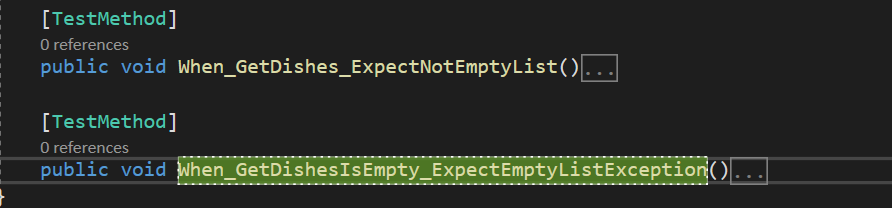
MessageQueueTests



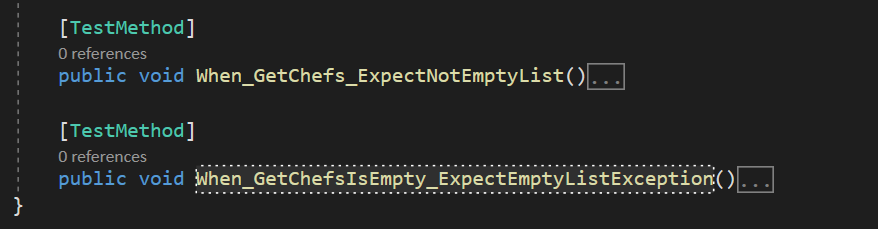
GetOrderQueueTests



GetDishesQuery



GetChefQueryTests



Remember!

1. If I insert a bug to the program the tests should fail
2. If I’m adding **new** functionality I should not change existing tests

For any question feel free to contact us!

Thanks,

Aviv Sevillia [Aviv.Sevillia@checkmarx.com](mailto:Aviv.Sevillia@checkmarx.com)

Mobile: 0528-972335