Design_v2_JingjingYang

Introduction

A system for a Drone Pizzeria was designed and implemented, including a Control Center, Drone Device, and a web client for ordering pizza. A database was incorporated for data persistence. This software system is a study project for a Software Architecture course in Autumn 2023.

This system is expected to receive orders from customers, prepare pizzas, and use drones for delivery. Customers place orders via a web client connected to a Pizza Restaurant system. Upon receipt of an order, the system communicates with the Control Center, which assigns a drone for delivery. The drones then communicate their status and updates using a Message Queue (MQ) system. All components of the system, including the Control Center, a Drone Device, and the web client, are isolated and scalable independently, enhancing the overall performance and reliability of the system.

Features implemented and responsibilities

Feature	Responsibility
Drone Pizzeria	Handles orders and communicates with drones
Control Center	Manages drone operations
Drone device	Operates based on instructions from the Control Center
Web client	Interface for ordering pizza
ActiveMQ	Facilitate communication between different system parts

SW Architecture description

The architecture style implemented in this project is a combination of Microservices and Event-Driven architecture.

1. **Microservices Architecture:** The system is broken down into smaller, independent services (Drone Control Center, Drone Device, Drone Pizzeria). Each of these services is developed, deployed, and scaled independently. They communicate with each other using well-defined APIs and protocols, resulting in a system that is easier to maintain and scale.
2. **Event-Driven Architecture:** This style of architecture is used for communication between the Drone Control Center and the Drone Device. In this setup, a change in state (event) in one service triggers the actions of another service. For instance, when the Drone Control Center dispatches a drone (an event), the Drone Device updates its status. Similarly, when the Drone Pizzeria receives an order (an event), it notifies the Control Center, which then dispatches a drone. Event-driven architecture allows for real-time, asynchronous communication that is highly scalable and resilient.

Context

At this level, the user interacts directly with the system via a web client interface to place pizza orders. Here, the system is perceived as a single, unified entity, with the primary focus being on the interactions between the user (or multiple users) and the system. This interaction can involve various activities such as browsing the menu, selecting pizzas, placing an order, making a payment, and tracking the delivery.

Containers

The system is compartmentalized into three main containers: The Drone Pizzeria, the Control Center, and the Drone Device. Each one of these containers represents

a significant part of the system and is independently deployable, meaning they can function autonomously. The Drone Pizzeria is the customer-facing side handling orders and payments. The Control Center is the backbone of the system, coordinating operations and managing drone deployments. The Drone Device container represents the physical drones used for pizza delivery.

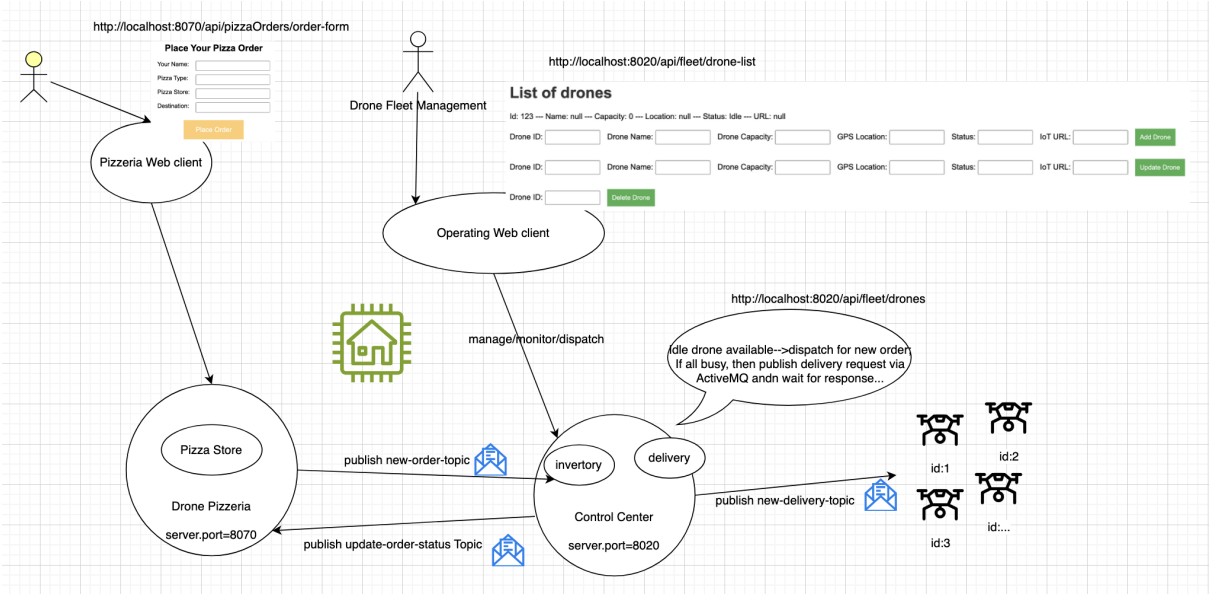
Components

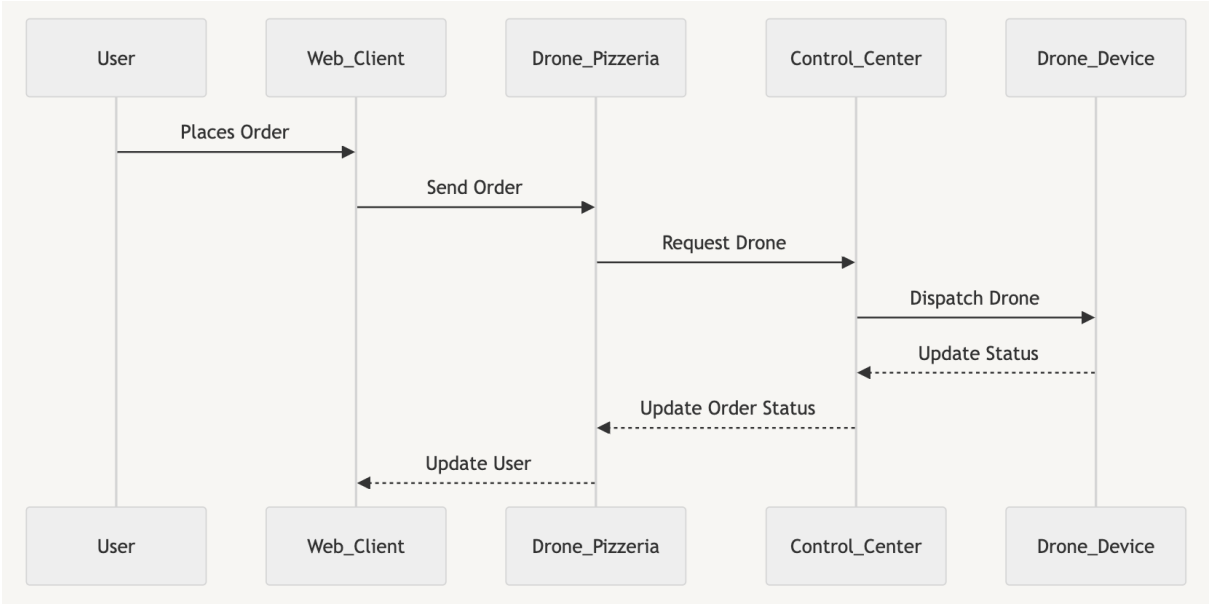
Each of the main containers of the system—the Drone Pizzeria, Control Center, and Drone Device—consists of several subcomponents, each designed to carry out specific tasks.

1. **Drone Pizzeria:** This component could include subcomponents for order processing, payment handling, and order tracking. They ensure that customer orders are processed correctly, payments are received, and customers can track their orders in real time.
2. **Control Center:** This component includes subcomponents for drone dispatch, drone monitoring, and system communication. They work together to assign drones for delivery, monitor the status of drones, and facilitate communication between the different parts of the system.
3. **Drone Device:** This component could include subcomponents for navigation, delivery, and communication. They control the drone's flight path, ensure delivery of the pizza, and communicate the drone's status back to the Control Center.
4. **Web Client:** This component could include subcomponents for user interface, user authentication, and order placement. They provide a user-friendly interface, secure user login, and a seamless order placement process.

Each subcomponent is engineered to operate independently yet cohesively with other subcomponents. This ensures the system operates effectively and efficiently, providing a seamless user experience.

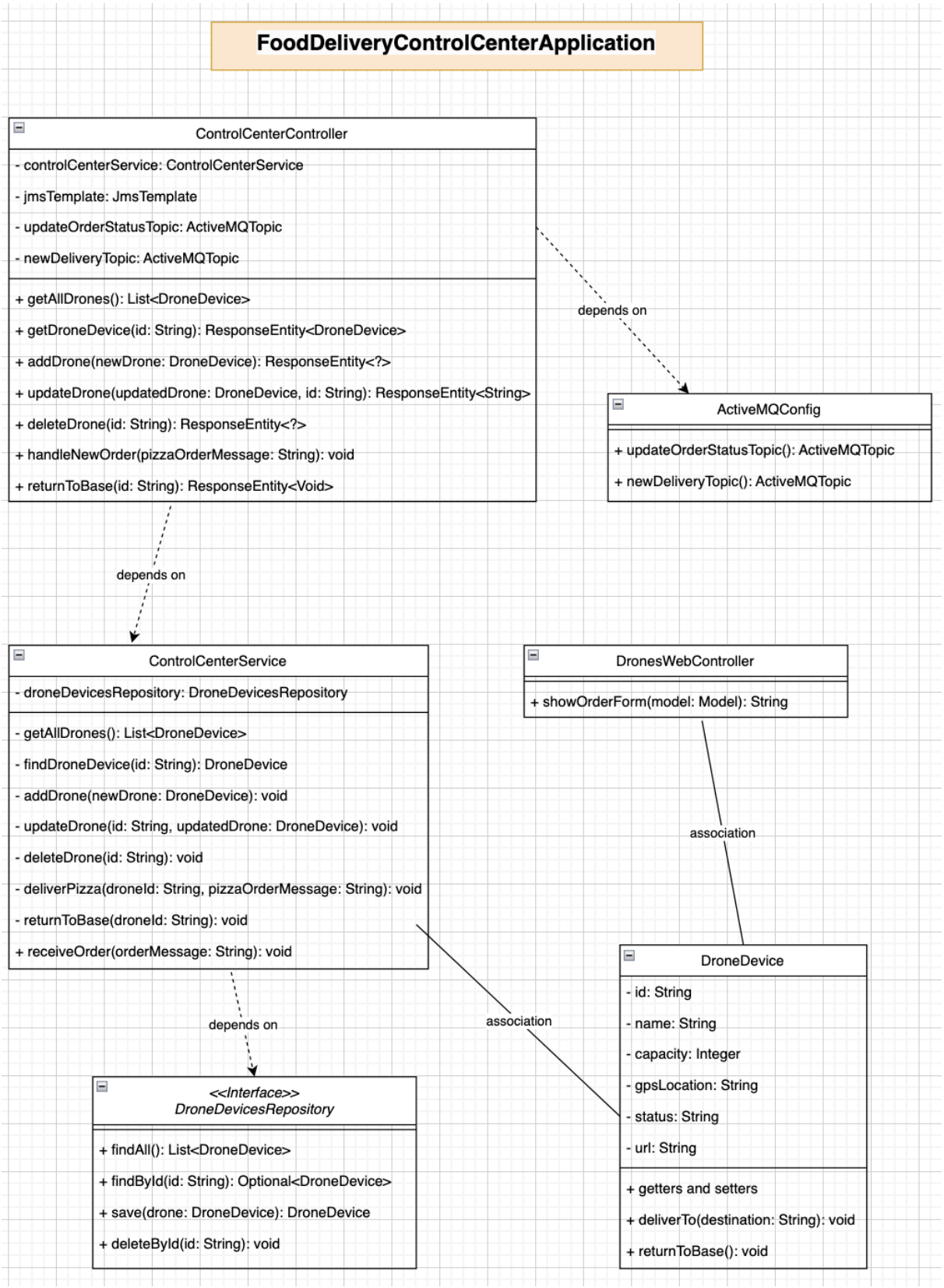
Components Interaction

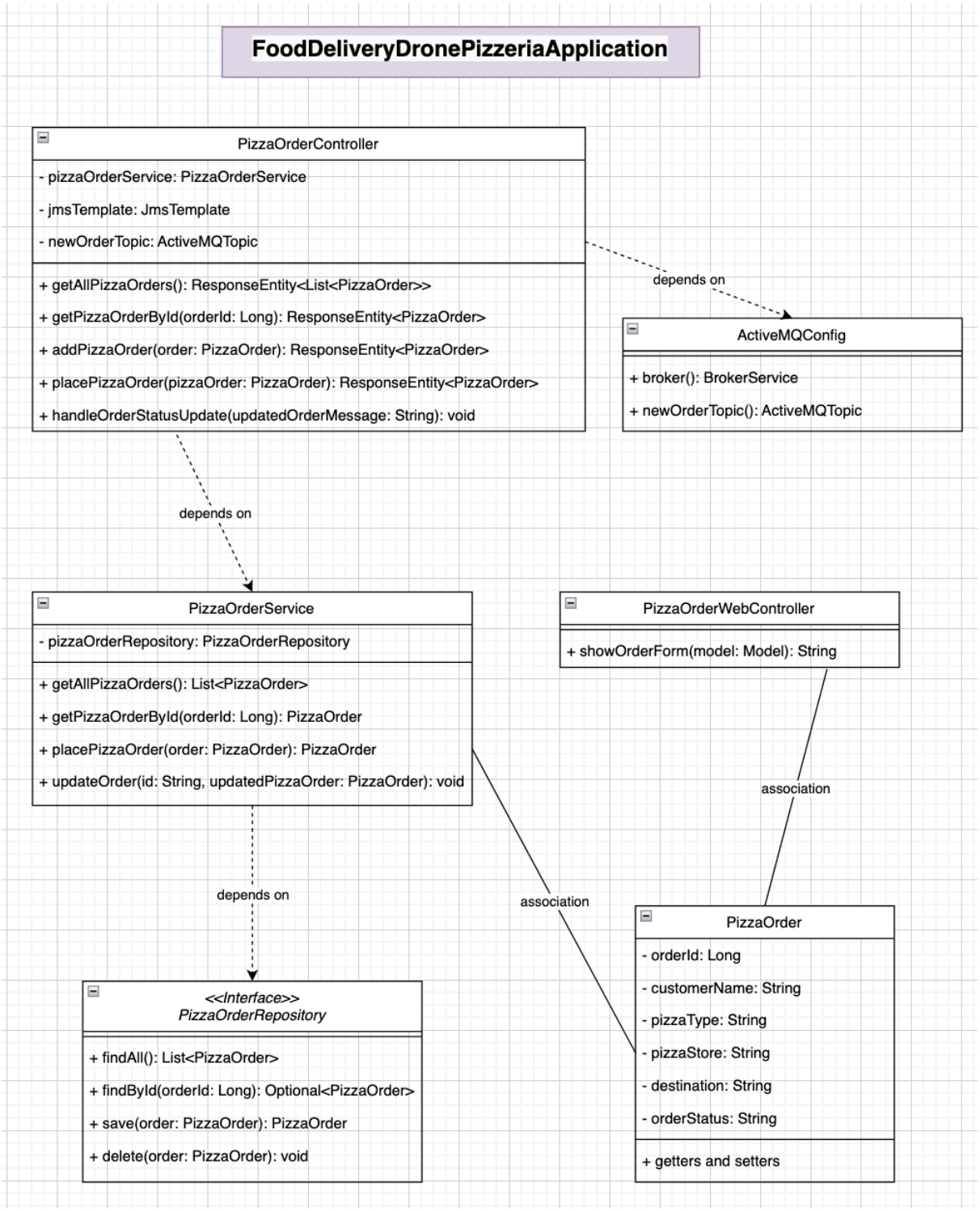




Code

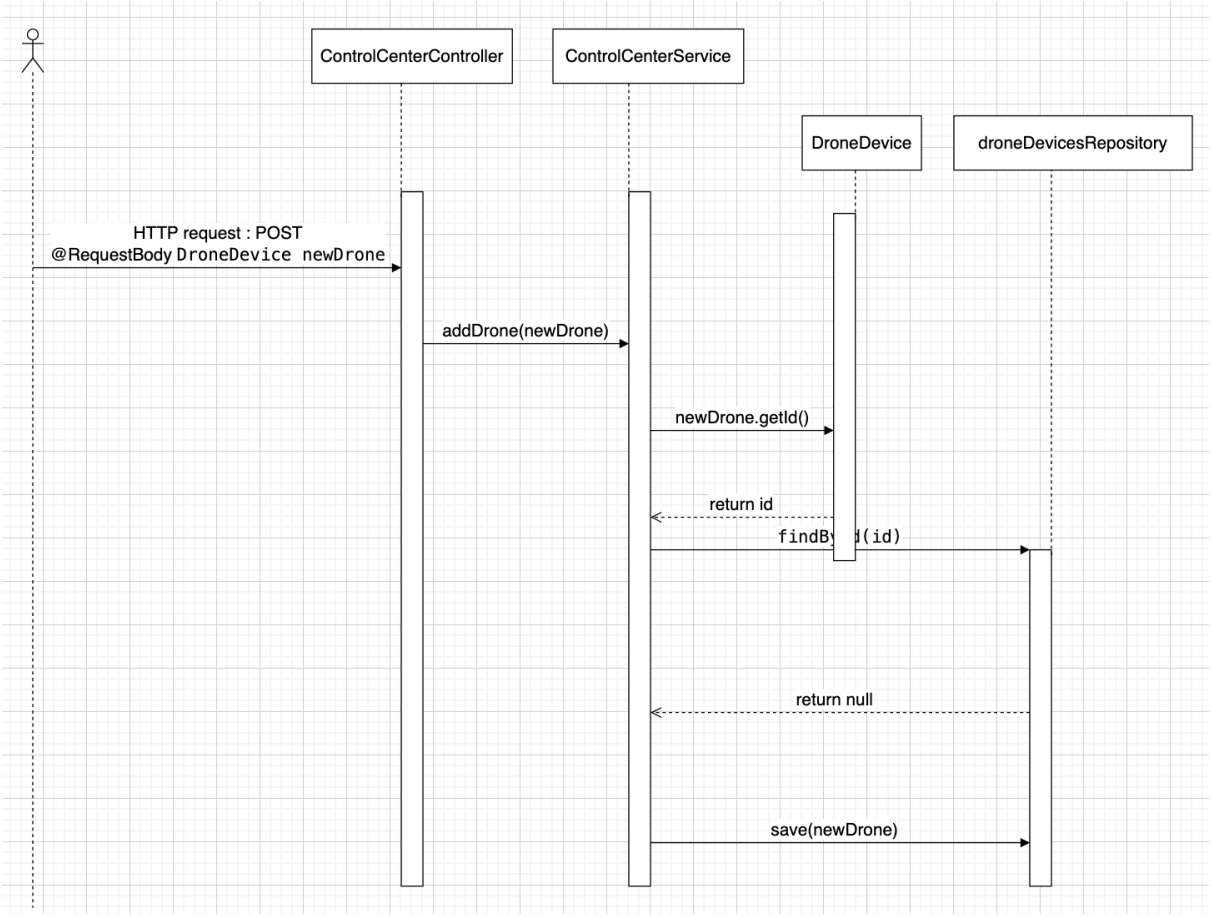
class diagrams


FoodDeliveryControlCenterApplication
ControlCenterController
- controlCenterService: ControlCenterService
- jmsTemplate: JmsTemplate
- updateOrderStatusTopic: ActiveMQTopic
- newDeliveryTopic: ActiveMQTopic
+ getAllDrones(): List<DroneDevice>
+ getDroneDevice(id: String): ResponseEntity<DroneDevice>
+ addDrone(newDrone: DroneDevice): ResponseEntity<?>
+ updateDrone(updatedDrone: DroneDevice, id: String): ResponseEntity<String>
+ deleteDrone(id: String): ResponseEntity<?>
+ handleNewOrder(pizzaOrderMessage: String): void
+ returnToBase(id: String): ResponseEntity<Void>
depends on
ActiveMQConfig
+ updateOrderStatusTopic(): ActiveMQTopic
+ newDeliveryTopic(): ActiveMQTopic
depends on
ControlCenterService
- droneDevicesRepository: DroneDevicesRepository
- getAllDrones(): List<DroneDevice>
- findDroneDevice(id: String): DroneDevice
- addDrone(newDrone: DroneDevice): void
- updateDrone(id: String, updatedDrone: DroneDevice): void
- deleteDrone(id: String): void
- deliverPizza(droneId: String, pizzaOrderMessage: String): void
- returnToBase(droneId: String): void
+ receiveOrder(orderMessage: String): void
DronesWebController
+ showOrderForm(model: Model): String
association
DroneDevice
- id: String
- name: String
- capacity: Integer
- gpsLocation: String
- status: String
- url: String
+ getters and setters
+ deliverTo(destination: String): void
+ returnToBase(): void
association
depends on
<<Interface>>
DroneDevicesRepository
+ findAll(): List<DroneDevice>
+ findById(id: String): Optional<DroneDevice>
+ save(drone: DroneDevice): DroneDevice
+ deleteById(id: String): void



sequence diagram

case to add a drone



Operating Instructions

1. **Clone the Repository:**

```
git clone https://gitlab.tamk.cloud/sw-architectures-and-design-2023-jingjing-yan
g/pizza-drone-miniproject.git
```

2. **Build and Run:** You need to have a MySQL database set up with the appropriate schema. The `spring.jpa.hibernate.ddl-auto=update` property in the application properties will automatically create the necessary tables based on the entity classes.
   You also need to have an ActiveMQ broker running at `tcp://localhost:61616` as specified in the application properties.
   Once these are set up, you can generate Maven Wrapper files by running `mvn wrapper:wrapper` from the root directory of your project, then navigate to each submodule directory and run the applications using the Spring Boot Maven plugin with the command `mvn spring-boot:run`.

3. **Access the Application:**

- The Control Center API is available at http://localhost:8020.
- The Drone Pizzeria API is available at http://localhost:8070.
- The Control Center UI form is available at http://localhost:8020/api/fleet/drone-list.
- The Drone Pizzeria Order Form is available at http://localhost:8070/api/pizzaOrders/order-form.
- Control Center API Endpoints:
  - `GET /api/fleet/drones`: Returns a list of all drones in the fleet.
  - `GET /api/fleet/drones/{id}`: Returns the drone with the specified ID.
  - `POST /api/fleet/drones`: Adds a new drone to the fleet.
  - `PUT /api/fleet/drones/{id}`: Updates the details of the drone with the specified ID.
  - `DELETE /api/fleet/drones/{id}`: Deletes the drone with the specified ID.
  - `PUT /api/fleet/drones/{id}/return`: Instructs the drone with the specified ID to return to base.
- Drone Pizzeria API Endpoints:
  - `GET /api/pizzaOrders`: Retrieves a list of all pizza orders.
  - `GET /api/pizzaOrders/{orderId}`: Retrieves a specific pizza order by its `orderId`.
  - `POST /api/pizzaOrders`: Adds a new pizza order.
  - `POST /api/pizzaOrders/place-order`: Places a new pizza order using the web-based pizza order form.
  - `POST /api/pizzaOrders/update-order-status`: Update the status of an order.

Testing instructions

Commands to test the system

1. **Get All Drones:**

   ```
   curl -X GET http://localhost:8020/api/fleet/drones
   ```

2. **Get Specific Drone by ID:**

`curl -X GET http://localhost:8020/api/fleet/drones/{id}`

3. **Add a Drone:**

   `curl -X POST -H "Content-Type: application/json" -d '{"id": 123, "name": "Drone One"}' http://localhost:8020/api/fleet/drones`

4. **Update a Drone:**

   `curl -X PUT -H "Content-Type: application/json" -d '{"name": "Updated Drone One", "capacity": 10}' http://localhost:8020/api/fleet/drones/{id}`

5. **Delete a Drone:**

   `curl -X DELETE http://localhost:8020/api/fleet/drones/{id}`

6. **Return to Base:**

   `curl -X PUT http://localhost:8020/api/fleet/drones/{id}/return`

Replace `{id}` with the actual id of the drone and the json payload with the drone values.

Example requests with verified results

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X POST -H "Content-Type: applicati
on/json" -d '{
  "id": "1",
  "name": "Drone 1",
  "capacity": 50,
  "gpsLocation": "Fleet Center",
  "status": "Idle",
  "url": "http://localhost:8011/api/drone1"
}' http://localhost:8020/api/fleet/drones
Drone added successfully%

(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X GET http://localhost:8020/api/fl
eet/drones
[{"id":"1","name":"Drone 1","capacity":50,"gpsLocation":"Fleet Center","status":"Idl
e","url":"http://localhost:8011/api/drone1"}]%

(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X PUT http://localhost:8020/api/fl
eet/drones/1/return

(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X GET http://localhost:8020/api/fl
eet/drones/1
{"id":"1","name":"Drone 1","capacity":50,"gpsLocation":"Fleet Center","status":"Return
ing","url":"http://localhost:8011/api/drone1"}%

(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X PUT -H "Content-Type: applicatio
n/json" -d '{"name": "Updated Drone One", "capacity": 80}' http://localhost:8020/api/f
leet/drones/123
A drone with this ID does not exist.%

(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X PUT -H "Content-Type: applicatio
```

```
n/json" -d '{"name": "Updated Drone One", "capacity": 80}' http://localhost:8020/api/f
leet/drones/1
Drone updated successfully%

(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X DELETE http://localhost:8020/ap
i/fleet/drones/1
Drone deleted successfully%

(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X GET http://localhost:8020/api/fl
eet/drones
[]%
```

Example of running and testing the system

Scenario: Order with No Available Drones

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl http://localhost:8020/api/fleet/dro
nes
[]%
```

Bob Johnson places an order for a Veggie pizza to be delivered to 789 Pine St. This is done via a POST request to the `/api/pizzaOrders` endpoint with the order details sent in JSON format.

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X POST -H "Content-Type: applicati
on/json" -d '{
  "customerName": "Bob Johnson",
  "pizzaType": "Veggie",
  "destination": "789 Pine St"
}' http://localhost:8070/api/pizzaOrders
{"orderId":3,"customerName":"Bob Johnson","pizzaType":"Veggie","pizzaStore":null,"dest
ination":"789 Pine St","orderStatus":null}%
```

When the order is placed, the system receives a POST request with the order details in JSON format. The system creates a new PizzaOrder object with the provided details and saves it to the database. The order details are then converted to a JSON string and sent as a message to the `new-order-topic` in ActiveMQ.

```
Topic: new-order-topic
Sending new order message: {"orderId":3,"customerName":"Bob Johnson","pizzaType":"Vegg
ie","pizzaStore":null,"destination":"789 Pine St","orderStatus":null}
```

Upon receiving the new order message, the system tries to assign a drone for delivery. However, as there are no drones available at the moment, the system updates the order status to "Wait for shipping". The order request is then sent to drones and the system publishes this status update message.

```
Received new order message: {"orderId":3,"customerName":"Bob Johnson","pizzaType":"Veg
gie","pizzaStore":null,"destination":"789 Pine St","orderStatus":null}

Published order request message to drones: {"orderId":3,"customerName":"Bob Johnso
n","pizzaType":"Veggie","pizzaStore":null,"destination":"789 Pine St","orderStatus":nu
ll}

Published order status message: {"orderId":3,"customerName":"Bob Johnson","pizzaTyp
e":"Veggie","pizzaStore":null,"destination":"789 Pine St","orderStatus":"Wait for ship
ping"}
```

```
Order request delivery Message Received: {"orderId":3,"customerName":"Bob Johnson","pi
zzaType":"Veggie","pizzaStore":null,"destination":"789 Pine St","orderStatus":"Wait fo
r shipping"}
Processing the order now...
```

The system then listens for this update message, updates the order status in the database, and returns the updated PizzaOrder object. When Bob Johnson requests to view his pizza order, the order he placed is shown with the status as "Wait for shipping".

```
Received order status update message: {"orderId":3,"customerName":"Bob Johnson","pizza
Type":"Veggie","pizzaStore":null,"destination":"789 Pine St","orderStatus":"Wait for s
hipping"}
```

Scenario: Order with Available Drones

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X POST -H "Content-Type: applicati
on/json" -d '{
  "id": "1",
  "name": "Drone 1",
  "capacity": 50,
  "gpsLocation": "Fleet Center",
  "status": "Idle",
  "url": "http://localhost:8011/api/drone1"
}' http://localhost:8020/api/fleet/drones
Drone added successfully%
```

Emma Watson places an order for a Hawaiian pizza to be delivered to 321 Maple Ave. This is done via a POST request to the `/api/pizzaOrders` endpoint with the order details sent in JSON format.

```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X POST -H "Content-Type: applicati
on/json" -d '{
  "customerName": "Emma Watson",
  "pizzaType": "Hawaiian",
  "destination": "321 Maple Ave"
}' http://localhost:8070/api/pizzaOrders
{"orderId":4,"customerName":"Emma Watson","pizzaType":"Hawaiian","pizzaStore":null,"de
stination":"321 Maple Ave","orderStatus":null}%
```

When the order is placed, the system receives a POST request with the order details in JSON format. The system creates a new PizzaOrder object with the provided details and saves it to the database. The order details are then converted to a JSON string and sent as a message to the `new-order-topic` in ActiveMQ.

```
Topic: new-order-topic
Sending new order message: {"orderId":4,"customerName":"Emma Watson","pizzaType":"Hawa
iian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":null}
```

Upon receiving the new order message, the system finds a drone available for delivery. The drone is then dispatched to the destination address, and the system updates the order status to "In Transit" and publishes this status update message.

```
Received new order message: {"orderId":4,"customerName":"Emma Watson","pizzaType":"Haw
aiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":null}

Drone with ID 1 has been dispatched for order: {"orderId":4,"customerName":"Emma Watso
n","pizzaType":"Hawaiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatu
s":null}

Published order status message: {"orderId":4,"customerName":"Emma Watson","pizzaTyp
e":"Hawaiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":"In Transi
t"}
```

```
Drone with ID 1 is flying to 321 Maple Ave
```

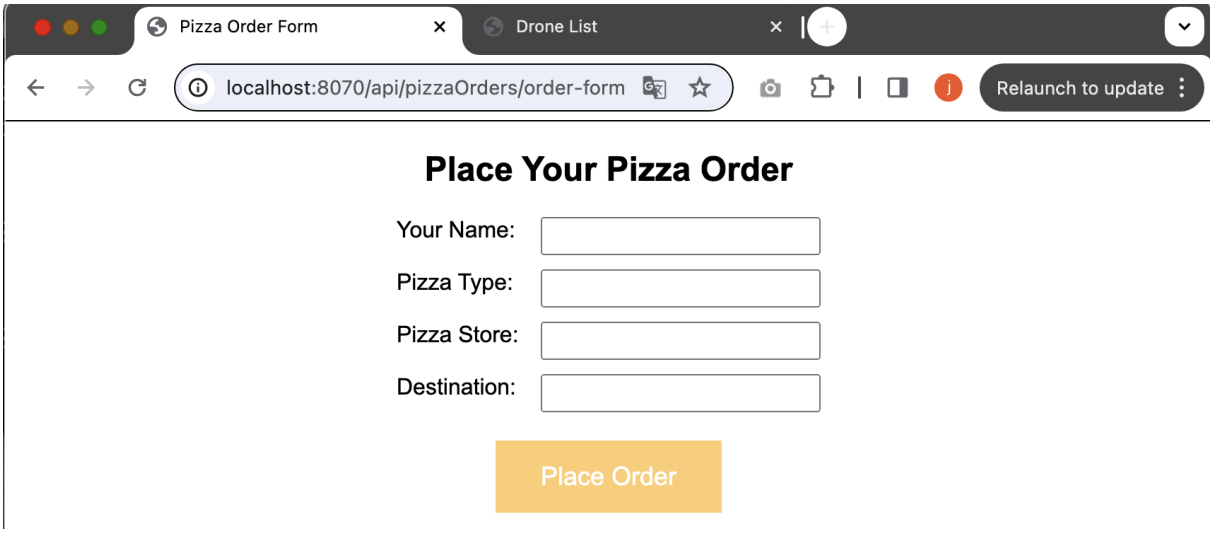
The system then listens for this update message, updates the order status in the database, and returns the updated PizzaOrder object. When Emma Watson requests

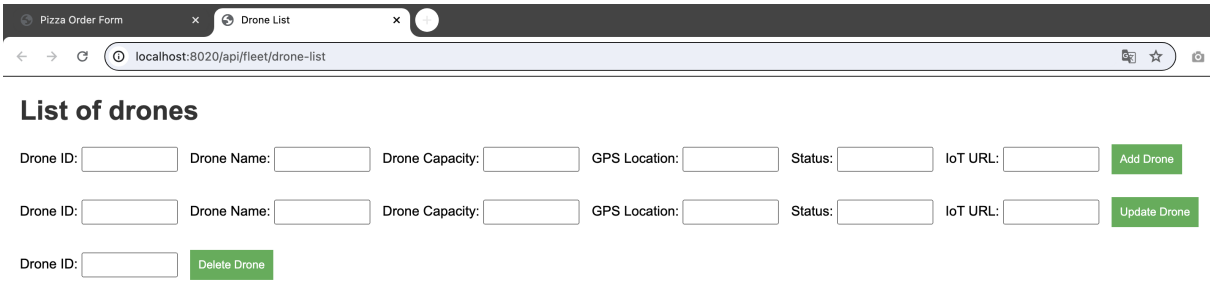
to view her pizza order, the order she placed is shown with the status as "In Transit".

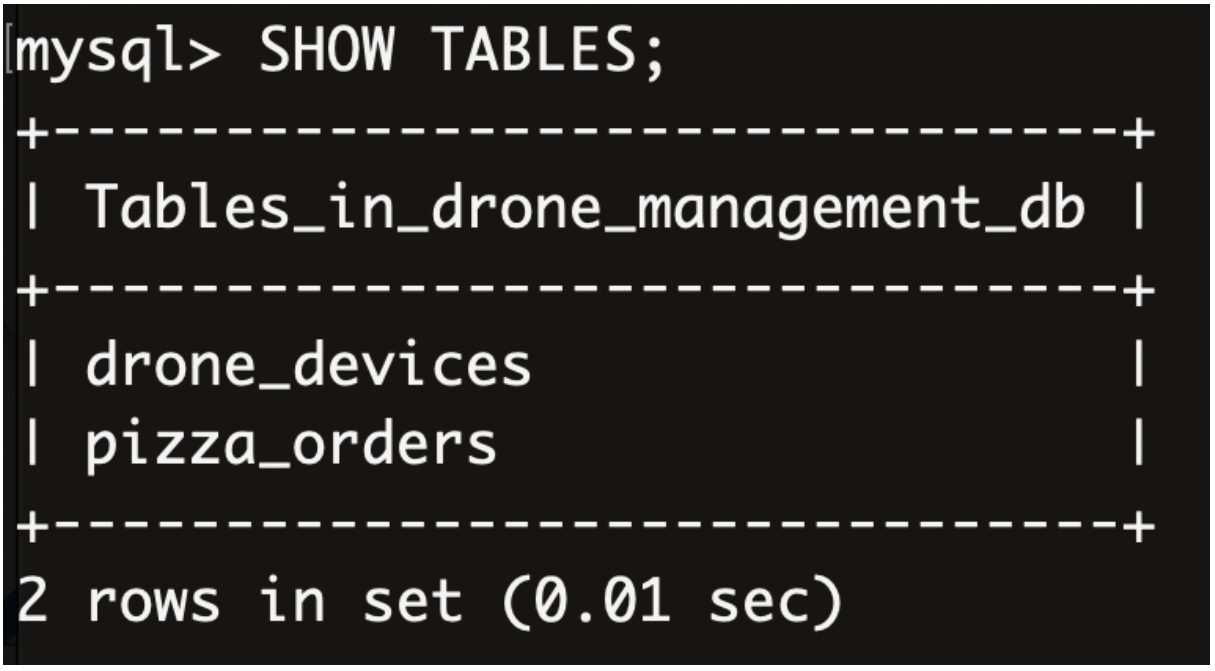
```
Received order status update message: {"orderId":4,"customerName":"Emma Watson","pizza
Type":"Hawaiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":"In Tra
nsit"}
```

Screenshots

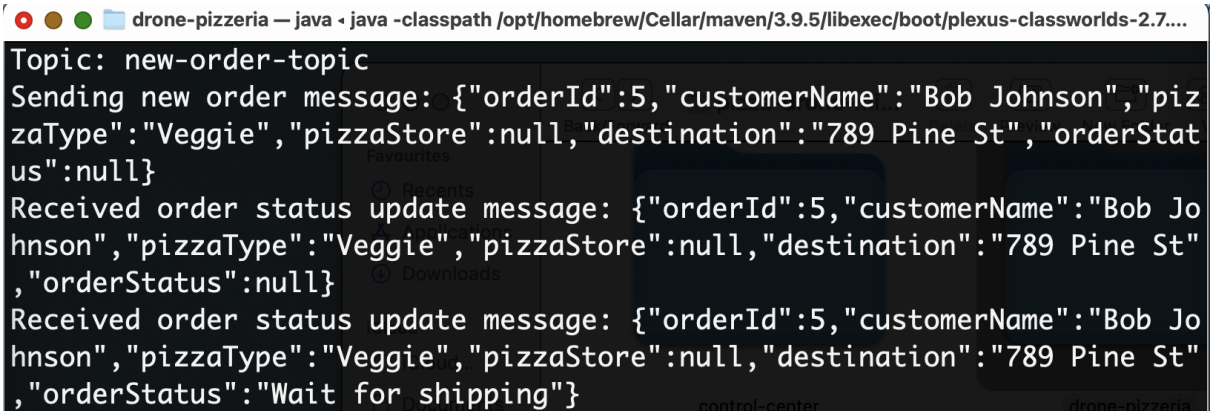
1. **Scenario: Order with No Available Drones**

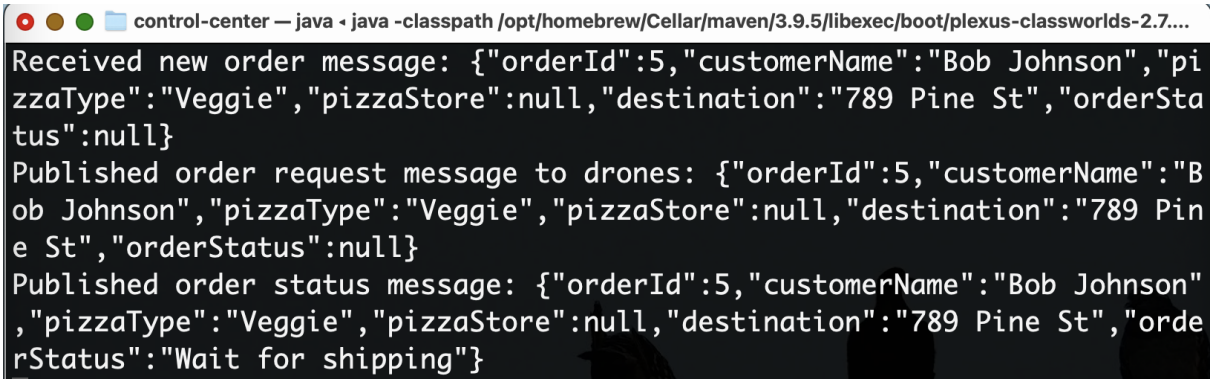




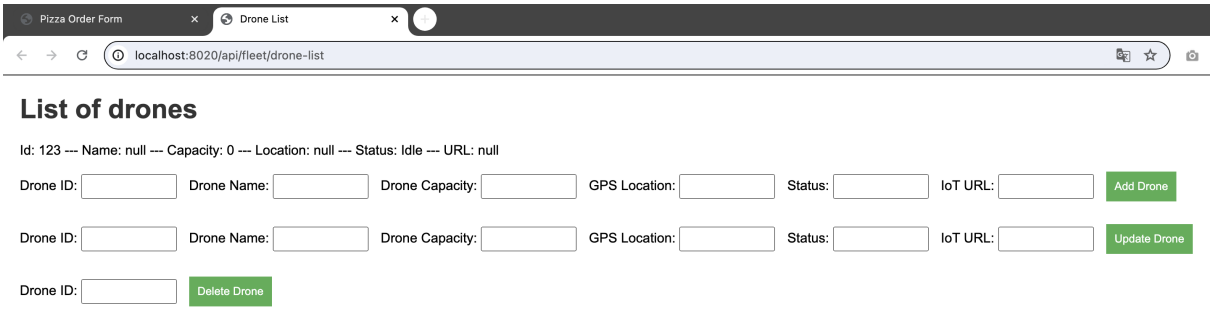



```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X POST -H "Content-Type: application/json" -d '{
  "customerName": "Bob Johnson",
  "pizzaType": "Veggie",
  "destination": "789 Pine St"
}' http://localhost:8070/api/pizzaOrders
[{"orderId":5,"customerName":"Bob Johnson","pizzaType":"Veggie","pizzaStore":null,"destination":"789 Pine S(
```





2. **Scenario: Order with Available Drones**




```
(base) jingjingyang@jingjings-MacBook-Pro ~ % curl -X POST -H "Content-Type: application/json" -d '{
  "customerName": "Emma Watson",
  "pizzaType": "Hawaiian",
  "destination": "321 Maple Ave"
}' http://localhost:8070/api/pizzaOrders
{"orderId":6,"customerName":"Emma Watson","pizzaType":"Hawaiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":null}%
```

drone-pizzeria — java ◂ java -classpath /opt/homebrew/Cellar/maven/3.9.5/libexec/boot/plexus-classworlds-2.7....

```
Topic: new-order-topic
Sending new order message: {"orderId":6,"customerName":"Emma Watson","pizzaType":"Hawaiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":null}
Received order status update message: {"orderId":6,"customerName":"Emma Watson","pizzaType":"Hawaiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":"In Transit"}
```

control-center — java ◂ java -classpath /opt/homebrew/Cellar/maven/3.9.5/libexec/boot/plexus-classworlds-2.7....

```
Received new order message: {"orderId":6,"customerName":"Emma Watson","pizzaType":"Hawaiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":null}
Drone with ID 123 is flying to 321 Maple Ave
Drone with ID 123 has been dispatched for order: {"orderId":6,"customerName":"Emma Watson","pizzaType":"Hawaiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":null}
Published order status message: {"orderId":6,"customerName":"Emma Watson","pizzaType":"Hawaiian","pizzaStore":null,"destination":"321 Maple Ave","orderStatus":"In Transit"}
```