Introduction

Date: December 2023

Course: Software Architectures and Design 2023

Study Project

Table of Contents

Project Description:

Drone based delivery system allows restaurants to delegate deliveries to their end customers, to drones.

Software Description:

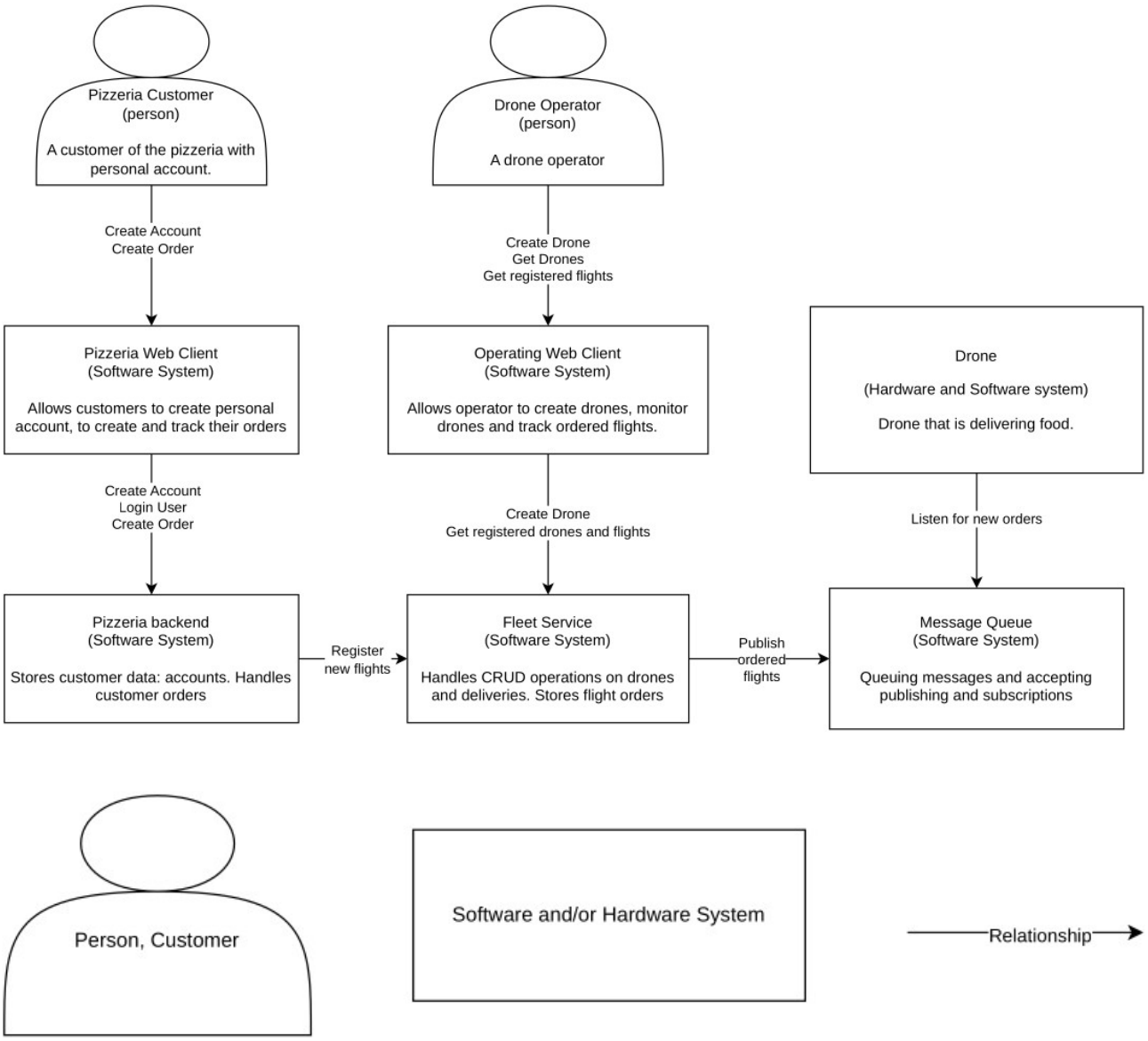
Drone based delivery system, including data persistence in NoSQL Mongo Database, message distribution into Message Queue utilizing RabbitMQ, Java based drone application, Java based fleet control service, NodeJS based pizza store backend handling customers ordering food, Pizzeria web client, and Web Control Panel, every service is running inside of docker containers.

Features implemented and responsibilities

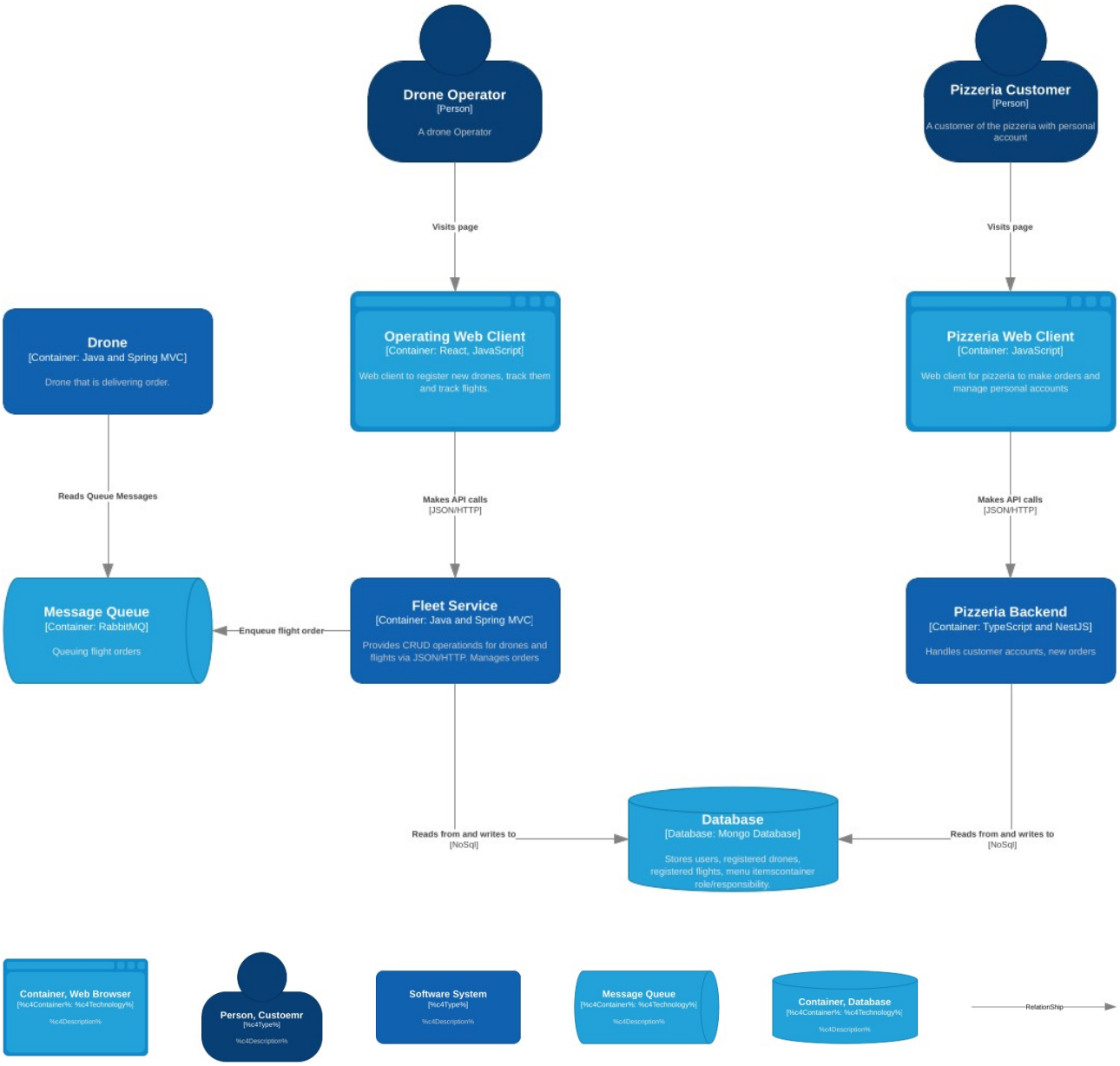
Feature	Technology	Responsibility
Containerization	Docker	Containerize modules
Database	MongoDB	Data persistence
Message Queue	RabbitMQ	Queuing messages
Fleet Service	Java Boot Spring	Register drones and flights. Menage drone and flights
Pizza Store	NestJS	Handle customer orders
Operating Web Client	ReactJS	Create drones. Track existing drones and flights
Pizzeria Web Client	ReactJS	Create orders
Drone	Java Boot Spring	Listen Message Queue. Deliver food

Software Architecture Description

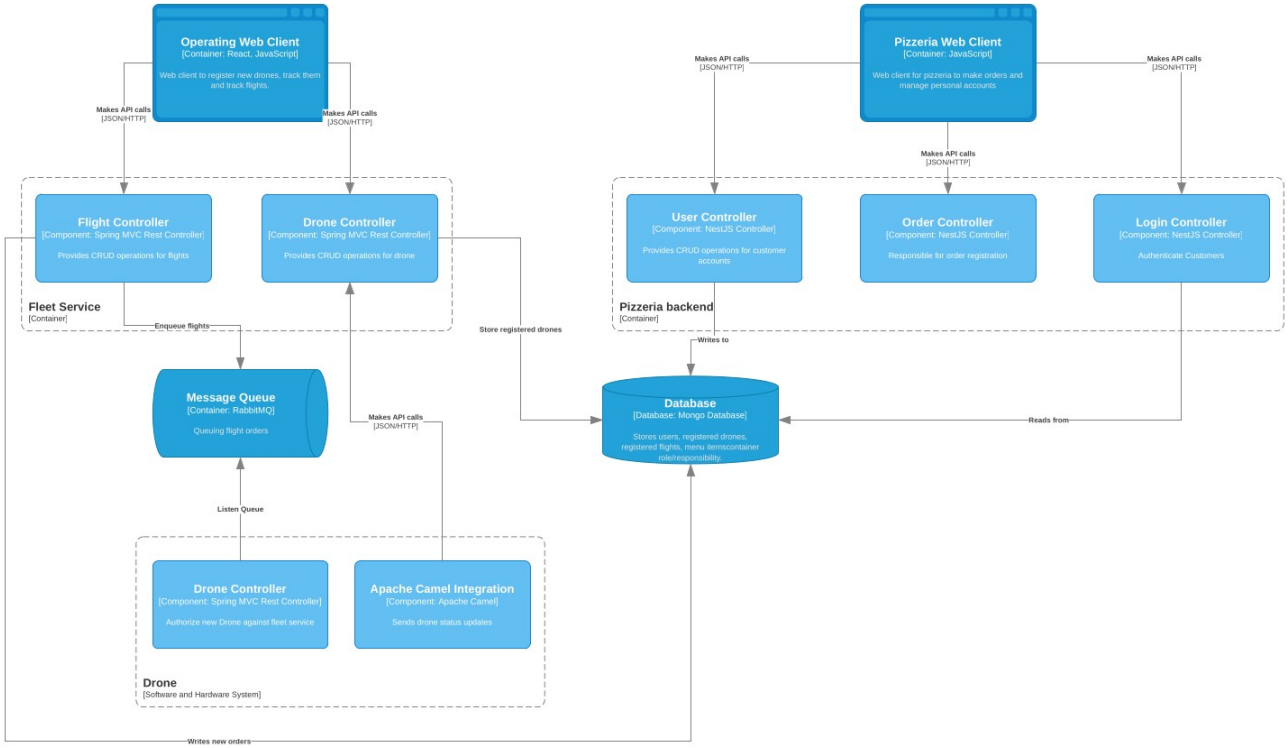
Context



Containers

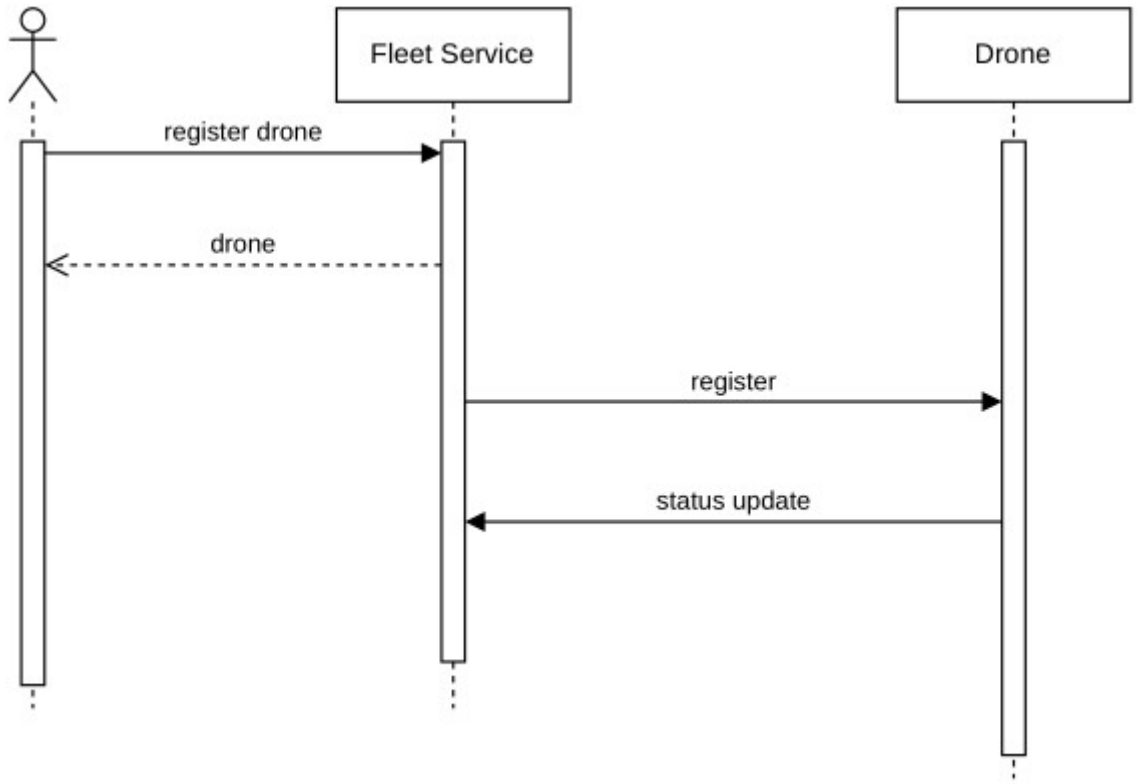

Drone Operator
[Person]
A drone Operator
Pizzeria Customer
[Person]
A customer of the pizzeria with personal account
Visits page
Visits page
Drone
[Container: Java and Spring MVC]
Drone that is delivering order.
Operating Web Client
[Container: React, JavaScript]
Web client to register new drones, track them and track flights.
Pizzeria Web Client
[Container: JavaScript]
Web client for pizzeria to make orders and manage personal accounts
Reads Queue Messages
Makes API calls
[JSON/HTTP]
Makes API calls
[JSON/HTTP]
Message Queue
[Container: RabbitMQ]
Queuing flight orders
Enqueue flight order
Fleet Service
[Container: Java and Spring MVC]
Provides CRUD operationds for drones and flights via JSON/HTTP. Manages orders
Pizzeria Backend
[Container: TypeScript and NestJS]
Handles customer accounts, new orders
Reads from and writes to
[NoSql]
Database
[Database: Mongo Database]
Stores users, registered drones, registered flights, menu itemscontainer role/responsibility
Reads from and writes to
[NoSql]
Container, Web Browser
[%c4Container%: %c4Technology%]
%c4Description%
Person, Custoemr
[%c4Type%]
%c4Description%
Software System
[%c4Type%]
%c4Description%
Message Queue
[%c4Container%: %c4Technology%]
%c4Description%
Container, Database
[%c4Container%: %c4Technology%]
%c4Description%
RelationShip

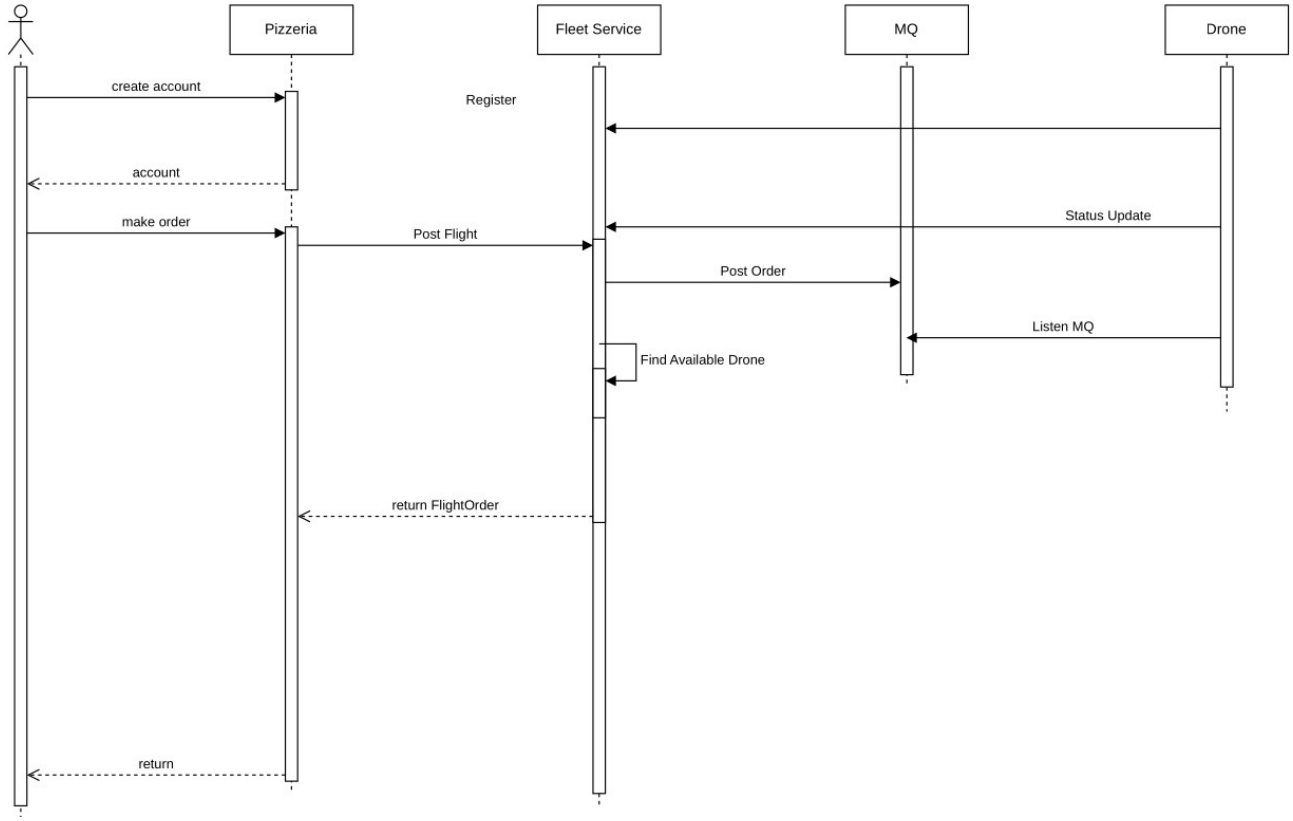
Components


Operating Web Client
[Container: React, JavaScript]
Web client to register new drones, track them and track flights.
Pizzeria Web Client
[Container: JavaScript]
Web client for pizzeria to make orders and manage personal accounts
Makes API calls
[JSON/HTTP]
Makes API calls
[JSON/HTTP]
Makes API calls
[JSON/HTTP]
Makes API calls
[JSON/HTTP]
Makes API calls
[JSON/HTTP]
Flight Controller
[Component: Spring MVC Rest Controller]
Provides CRUD operations for flights
Drone Controller
[Component: Spring MVC Rest Controller]
Provides CRUD operations for drone
Fleet Service
[Container]
User Controller
[Component: NestJS Controller]
Provides CRUD operations for customer accounts
Order Controller
[Component: NestJS Controller]
Responsible for order registration
Login Controller
[Component: NestJS Controller]
Authenticate Customers
Pizzeria backend
[Container]
Enqueue flights
Store registered drones
Writes to
Message Queue
[Container: RabbitMQ]
Queuing flight orders
Makes API calls
[JSON/HTTP]
Database
[Database: Mongo Database]
Stores users, registered drones, registered flights, menu itemscontainer role/responsibility.
Reads from
Listen Queue
Drone Controller
[Component: Spring MVC Rest Controller]
Authorize new Drone against fleet service
Apache Camel Integration
[Component: Apache Camel]
Sends drone status updates
Drone
[Software and Hardware System]
Writes new orders

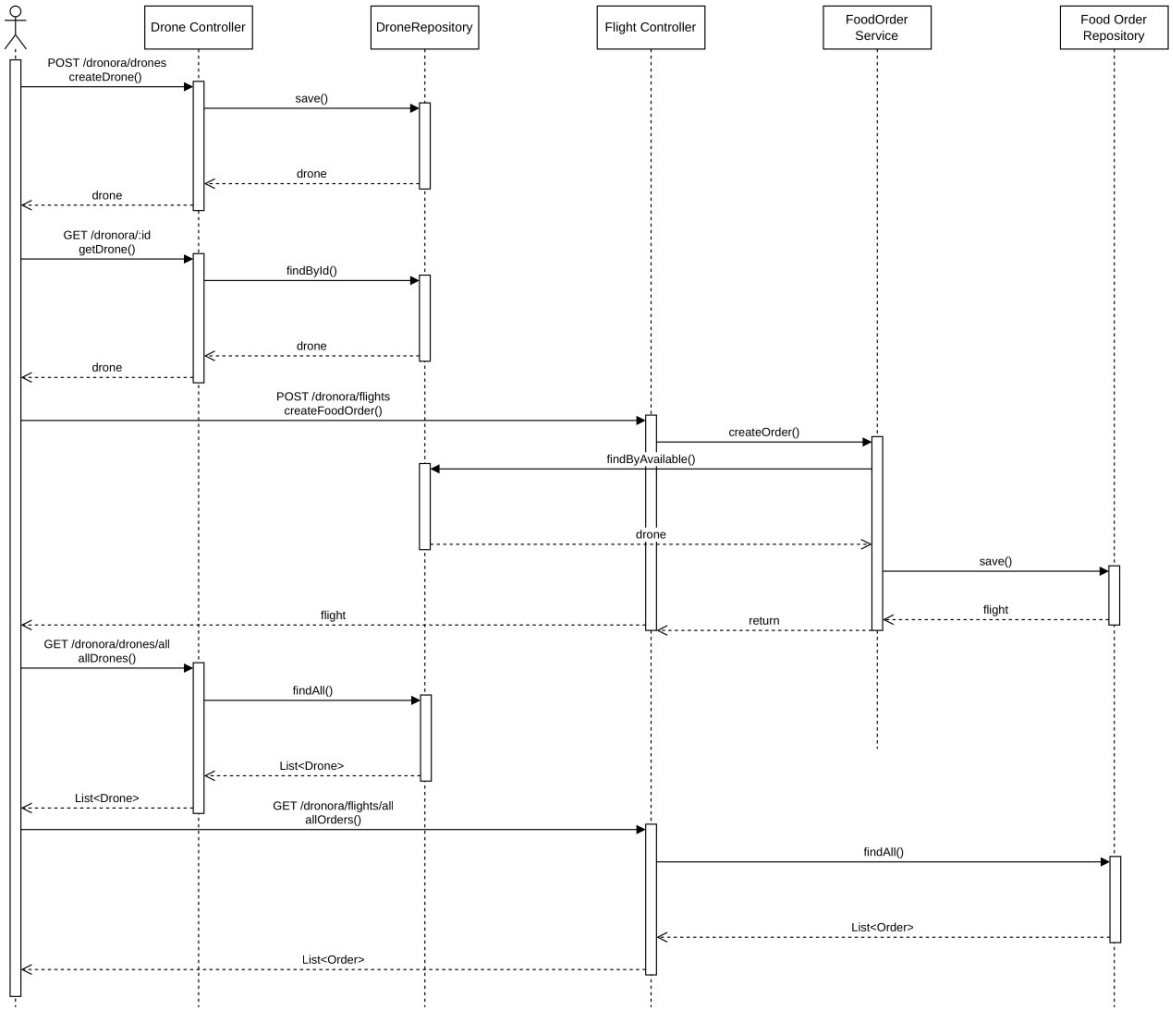

Container, Web Browser
[%c4Container%: %c4Technology%]
%c4Description%
Software System
[%c4Type%]
%c4Description%
Message Queue
[%c4Container%: %c4Technology%]
%c4Description%
Container, Database
[%c4Container%: %c4Technology%]
%c4Description%
RelationShip

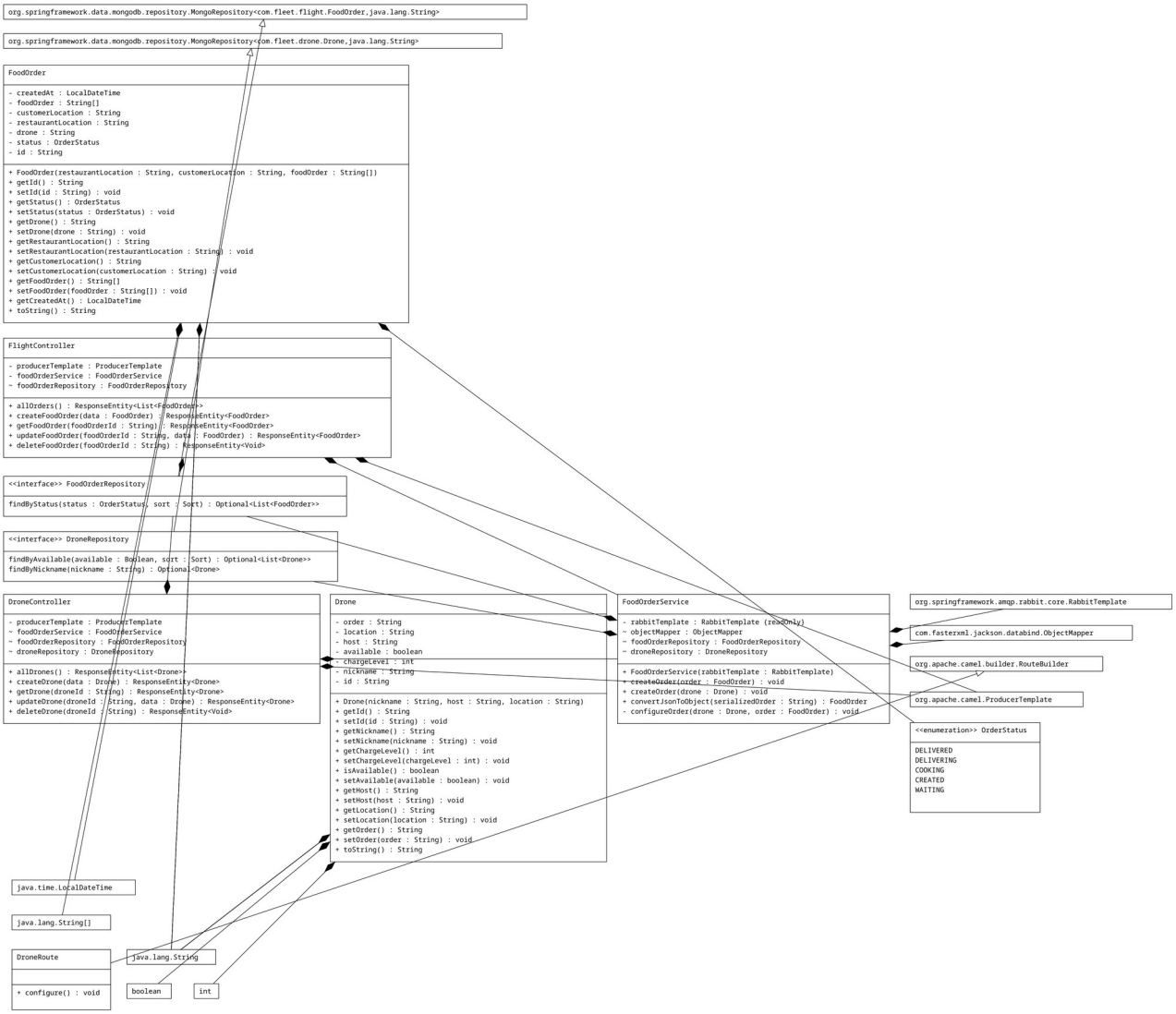
Components Interaction





Code


Drone Controller
DroneRepository
Flight Controller
FoodOrder Service
Food Order Repository
POST /dronora/drones
createDrone()
save()
drone
drone
GET /dronora/:id
getDrone()
findById()
drone
drone
POST /dronora/flights
createFoodOrder()
createOrder()
findByAvailable()
drone
save()
flight
return
flight
GET /dronora/drones/all
allDrones()
findAll()
List<Drone>
List<Drone>
GET /dronora/flights/all
allOrders()
findAll()
List<Order>
List<Order>


org.springframework.data.mongodb.repository.MongoRepository<com.fleet.flight.FoodOrder,java.lang.String>
org.springframework.data.mongodb.repository.MongoRepository<com.fleet.drone.Drone,java.lang.String>
FoodOrder
- createdAt : LocalDateTime
- foodOrder : String[]
- customerLocation : String
- restaurantLocation : String
- drone : String
- status : OrderStatus
- id : String
+ FoodOrder(restaurantLocation : String, customerLocation : String, foodOrder : String[])
+ getId() : String
+ setId(id : String) : void
+ getStatus() : OrderStatus
+ setStatus(status : OrderStatus) : void
+ getDrone() : String
+ setDrone(drone : String) : void
+ getRestaurantLocation() : String
+ setRestaurantLocation(restaurantLocation : String) : void
+ getCustomerLocation() : String
+ setCustomerLocation(customerLocation : String) : void
+ getFoodOrder() : String[]
+ setFoodOrder(foodOrder : String[]) : void
+ getCreatedAt() : LocalDateTime
+ toString() : String
FlightController
- producerTemplate : ProducerTemplate
- foodOrderService : FoodOrderService
~ foodOrderRepository : FoodOrderRepository
+ allOrders() : ResponseEntity<List<FoodOrder>>
+ createFoodOrder(data : FoodOrder) : ResponseEntity<FoodOrder>
+ getFoodOrder(foodOrderId : String) : ResponseEntity<FoodOrder>
+ updateFoodOrder(foodOrderId : String, data : FoodOrder) : ResponseEntity<FoodOrder>
+ deleteFoodOrder(foodOrderId : String) : ResponseEntity<Void>
<<interface>> FoodOrderRepository
findByStatus(status : OrderStatus, sort : Sort) : Optional<List<FoodOrder>>
<<interface>> DroneRepository
findByAvailable(available : Boolean, sort : Sort) : Optional<List<Drone>>
findByNickname(nickname : String) : Optional<Drone>
DroneController
- producerTemplate : ProducerTemplate
~ foodOrderService : FoodOrderService
~ foodOrderRepository : FoodOrderRepository
~ droneRepository : DroneRepository
+ allDrones() : ResponseEntity<List<Drone>>
+ createDrone(data : Drone) : ResponseEntity<Drone>
+ getDrone(droneId : String) : ResponseEntity<Drone>
+ updateDrone(droneId : String, data : Drone) : ResponseEntity<Drone>
+ deleteDrone(droneId : String) : ResponseEntity<Void>
Drone
- order : String
- location : String
- host : String
- available : boolean
- chargeLevel : int
- nickname : String
- id : String
+ Drone(nickname : String, host : String, location : String)
+ getId() : String
+ setId(id : String) : void
+ getNickname() : String
+ setNickname(nickname : String) : void
+ getChargeLevel() : int
+ setChargeLevel(chargeLevel : int) : void
+ isAvailable() : boolean
+ setAvailable(available : boolean) : void
+ getHost() : String
+ setHost(host : String) : void
+ getLocation() : String
+ setLocation(location : String) : void
+ getOrder() : String
+ setOrder(order : String) : void
+ toString() : String
FoodOrderService
- rabbitTemplate : RabbitTemplate (readonly)
~ objectMapper : ObjectMapper
~ foodOrderRepository : FoodOrderRepository
~ droneRepository : DroneRepository
+ FoodOrderService(rabbitTemplate : RabbitTemplate)
+ createOrder(order : FoodOrder) : void
+ createOrder(drone : Drone) : void
+ convertJsonToObject(serializedOrder : String) : FoodOrder
- configureOrder(drone : Drone, order : FoodOrder) : void
org.springframework.amqp.rabbit.core.RabbitTemplate
com.fasterxml.jackson.databind.ObjectMapper
org.apache.camel.builder.RouteBuilder
org.apache.camel.ProducerTemplate
<<enumeration>> OrderStatus
DELIVERED
DELIVERING
COOKING
CREATED
WAITING
java.time.LocalDateTime
java.lang.String[]
DroneRoute
+ configure() : void
java.lang.String
boolean
int

Operating instructions

Installing, setting up and starting the containers

The following steps assume you have Docker and Docker compose plugin installed.

1. Clone the code:

   ssh://git@gitlab.tamk.cloud:1022/sw-architectures-n-design-2023-innokentii-kozlov/drone-food-delivery.git

2. Ports Needed

MongoDB	27017
RabbitMQ	15672; 5672
Fleet Service	8080
Pizzeria backend	9100
Pizzeria Web Client	3000
Operating Web Client	3001
Drone 1	8081
Drone 2	8082

3. Configure environment

   Create “pizza-store.env” file in the root directory. Copy and paste content of rood README.md file to *.env file, you should have 2 environmental variables: JWT_PUBLIC_KEY, JWT_PRIVATE_KEY

4. Run services

   docker compose up -d

   OR

   docker-compose up -d

Testing instructions

Fleet Service:

- Drones
  - Create Drone

```
curl --location 'http://localhost:8080/dronora/drones' \
--header 'Content-Type: application/json' \
--data '{
    "nickname": "innokentii",
    "capacity": 1000,
    "host": "drone-1:8081",
    "location": ""
}'
```

- Flights
    - Create flight

```
curl --location 'http://localhost:8080/dronora/flights' \
--header 'Content-Type: application/json' \
--data '{
    "restaurantLocation": "Hämeenkatu 1",
    "customerLocation": "Kuntokatu 3",
    "foodOrder": ["Margherita Pizza", "Caesar Salad"]
}'
```

Pizzeria backend:

- User
    - Create User

```
curl --location 'http://127.0.0.1:9100/api/user/create' \
--header 'Content-Type: application/json' \
--data-raw '{
    "email": "test@test.com",
    "password": "password",
    "location": "Kuntokatu 3"
}'
```

    - Login User

```
curl --location 'http://127.0.0.1:9100/api/user/login' \
--header 'Content-Type: application/json' \
--data-raw '{
    "email": "test@test.com",
    "password": "password"
}'
```

- Menu
    - Fetch Menu

```
curl --location 'http://127.0.0.1:9100/api/menu'
```

- Order
    - Create Order

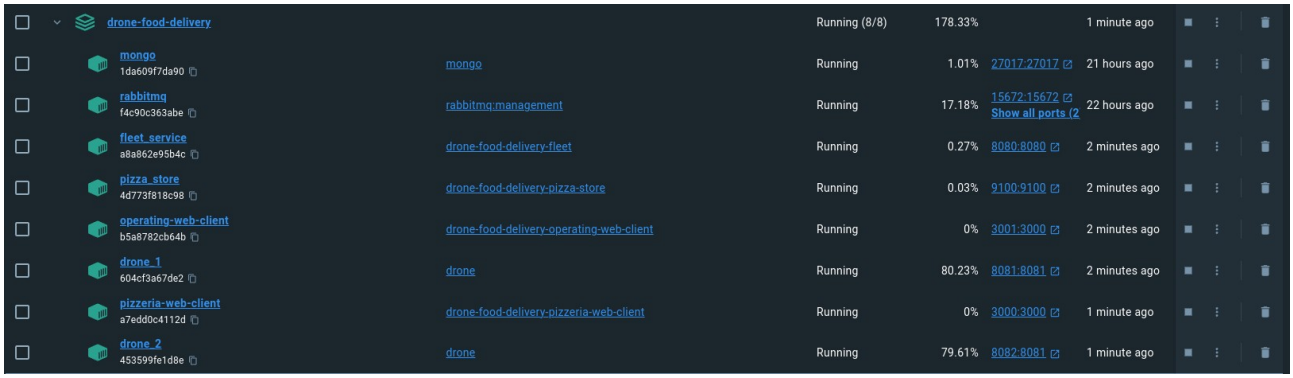
```
curl --location 'http://127.0.0.1:9100/api/order/create' \
--header 'Content-Type: application/json' \
--data '{
    "restaurantLocation": "Hämeenkatu 1",
    "customerLocation": "Kuntokatu 3",
    "foodOrder": ["Margherita Pizza", "Caesar Salad"]
}'
```

Verify results:

1. You’ll get a successful response

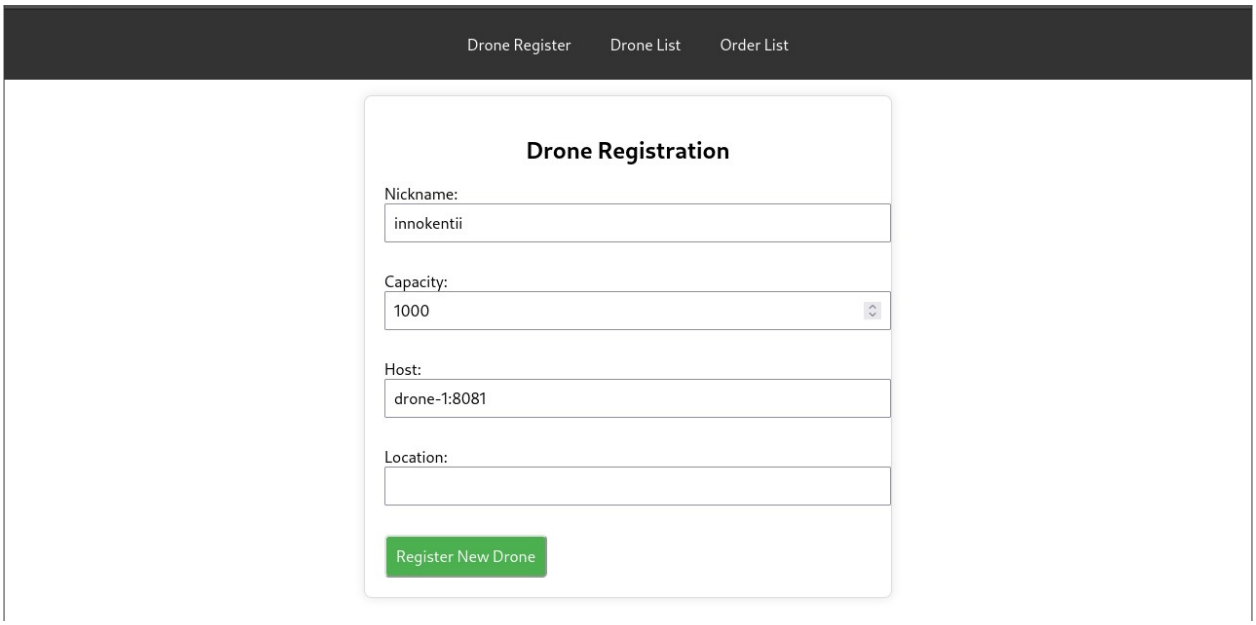
2. You will see a new document in database
3. You can see new drone/flight on Operating Web Client

Example of running and testing the system

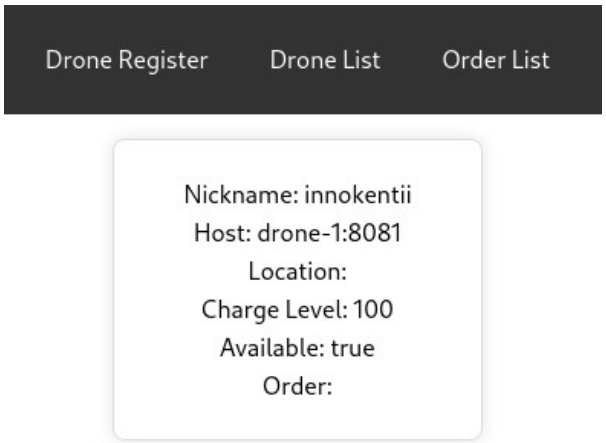


drone-food-delivery		Running (8/8)	178.33%		1 minute ago	
mongo 1da609f7da90	mongo	Running	1.01%	27017:27017	21 hours ago	
rabbitmq f4c90c363abe	rabbitmq:management	Running	17.18%	15672:15672 Show all ports (2	22 hours ago	
fleet_service a8a862e95b4c	drone-food-delivery-fleet	Running	0.27%	8080:8080	2 minutes ago	
pizza_store 4d773f818c98	drone-food-delivery-pizza-store	Running	0.03%	9100:9100	2 minutes ago	
operating-web-client b5a8782cb64b	drone-food-delivery-operating-web-client	Running	0%	3001:3000	2 minutes ago	
drone_1 604cf3a67de2	drone	Running	80.23%	8081:8081	2 minutes ago	
pizzeria-web-client a7edd0c4112d	drone-food-delivery-pizzeria-web-client	Running	0%	3000:3000	1 minute ago	
drone_2 453599fe1d8e	drone	Running	79.61%	8082:8081	1 minute ago	

Register a new Drone

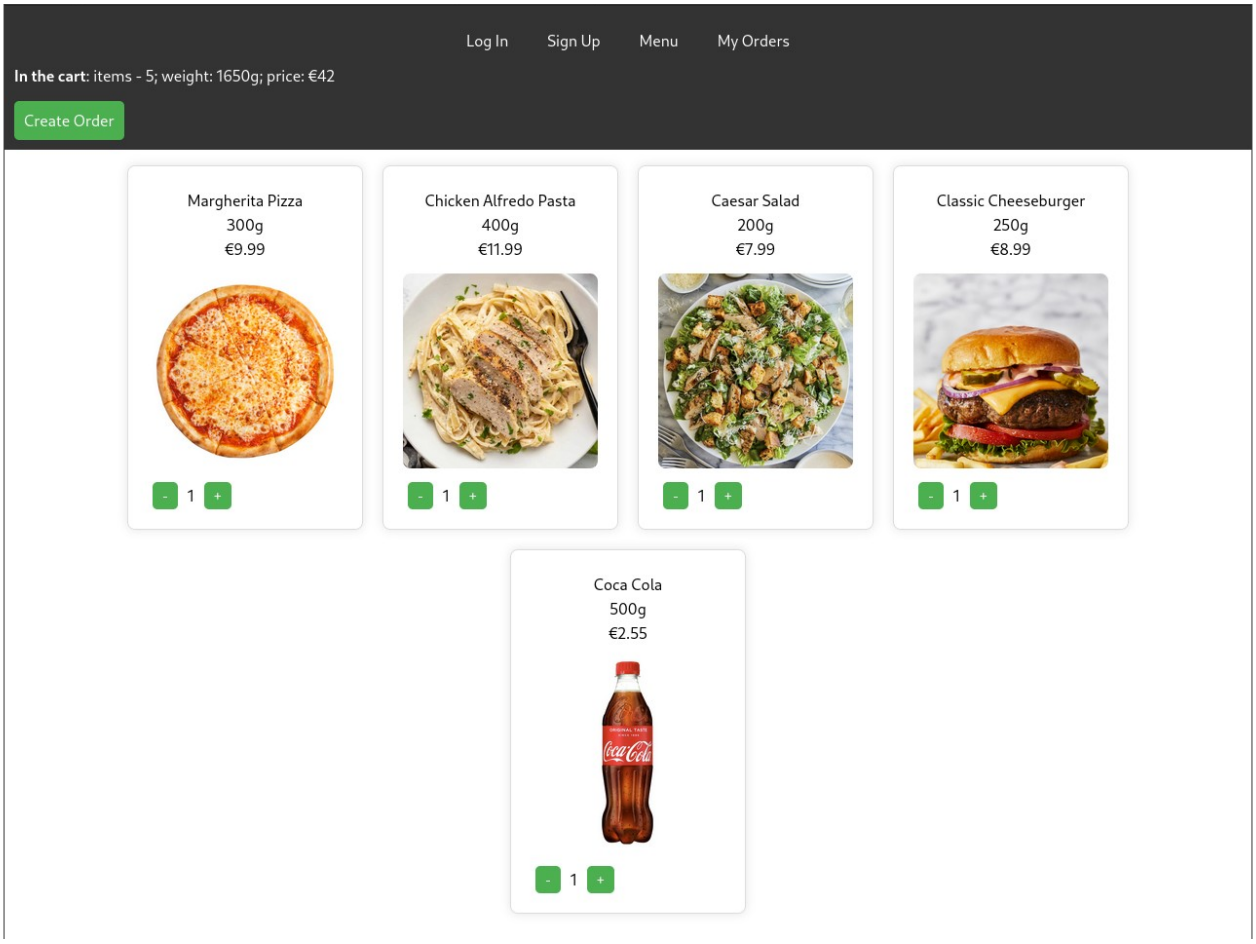


Verify new Drone

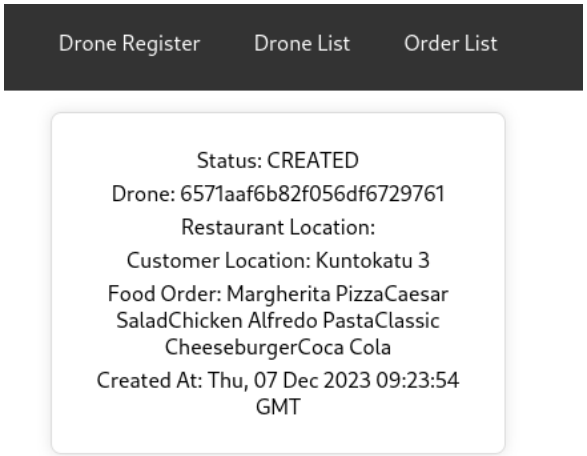



```
_id: ObjectId('6571aaf6b82f056df6729761')
nickname: "innokentii"
chargeLevel: 100
available: true
host: "drone-1:8081"
location: ""
_class: "com.fleet.drone.Drone"
```

Create Order



Verify Order




```
_id: ObjectId('6571ab4ab82f056df6729762')
status: "CREATED"
drone: "6571aaf6b82f056df6729761"
customerLocation: "Kuntokatu 3"
▸ foodOrder: Array (5)
createdAt: 2023-12-07T11:23:54.973+00:00
_class: "com.fleet.flight.FoodOrder"
```