



Master Distributed Systemes and Artificial Intelligence

Distributed Systemes

réalisé par : **Beidja Cheikh**

Professor : Pr. **YOUSSEF Mohamed**

| |
|---------------|
| Le 01-01-2024 |
|---------------|

Contents

| | |
|--|----|
| Établir une architecture technique du projet | 5 |
| 2. Créer un Projet Maven incluant les micro-services suivants : ressources-service, reservation-service, gateway-service, discovery-service, config-service et angular-front-app | 6 |
| 3. Développer et tester les micro-services discovery-service et gateway-service et config-service ... | 6 |
| Gateway-service avec le port 9999..... | 7 |
| getAllResources Method | 9 |
| getResourceById Method | 9 |
| addResource Method | 10 |
| updateResource Method..... | 10 |
| 5. Développer et tester le micro-service reservation-service (Entities, DAO, service, DTO, Mapper,11 | |
| getAllPersons Method | 12 |
| getPersonById Mthod..... | 12 |
| addPerson Method | 13 |
| updatePerson Method..... | 13 |
| reservations | 13 |
| 6. Développer un simple frontend web pour l'application..... | 14 |
| 7. Sécuriser l'application avec une authentification Keycloak..... | 16 |
| Keycloak..... | 16 |
| SECURITÉ DES MICROSERVICES | 16 |
| Keycloak..... | 17 |
| Gestion des ressources | 18 |
| Gestion des persons | 19 |
| Gestion des réservations | 20 |
| 8. Déployer l'application avec Docker et Docker compose..... | 21 |

1- Introduction

Ce projet vise à développer une application de gestion de réservations reposant sur une architecture micro-service. Les principaux composants incluent le "Resources-Service" pour la gestion des ressources, le "Reservation-Service" pour les réservations, et des services complémentaires tels que la passerelle, le service de découverte, et le service de configuration.

L'architecture technique s'appuie sur des technologies telles que Spring Cloud, Angular, Keycloak pour la sécurité, et Resilience4J pour la tolérance aux pannes. Chaque micro-service est conçu comme un module Maven distinct, permettant une gestion modulaire et évolutive du projet.

Le rapport détaille le processus de développement, de la conception initiale à la sécurisation de l'application avec Keycloak. Les points forts incluent la génération de documentation OpenAPI, l'utilisation de circuit breakers pour la fiabilité, et le déploiement avec Docker et Docker Compose. Le code source complet du projet, organisé en modules distincts, accompagne le rapport pour faciliter la compréhension et la reproduction du projet.



FIGURE 1 – Frameworks

Les outils de development

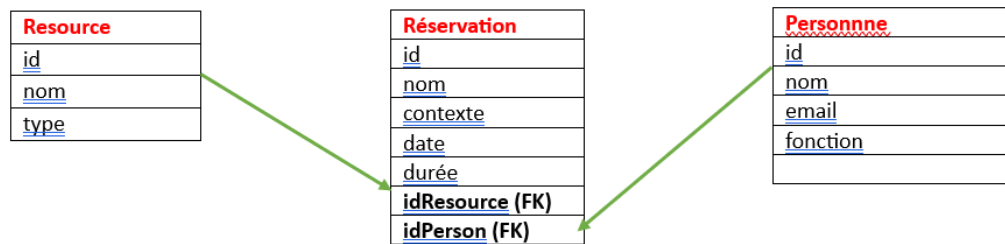


FIGURE 2 – Tools

BEIDJA CHEIKH

Modélisation du projet

|



Architecture

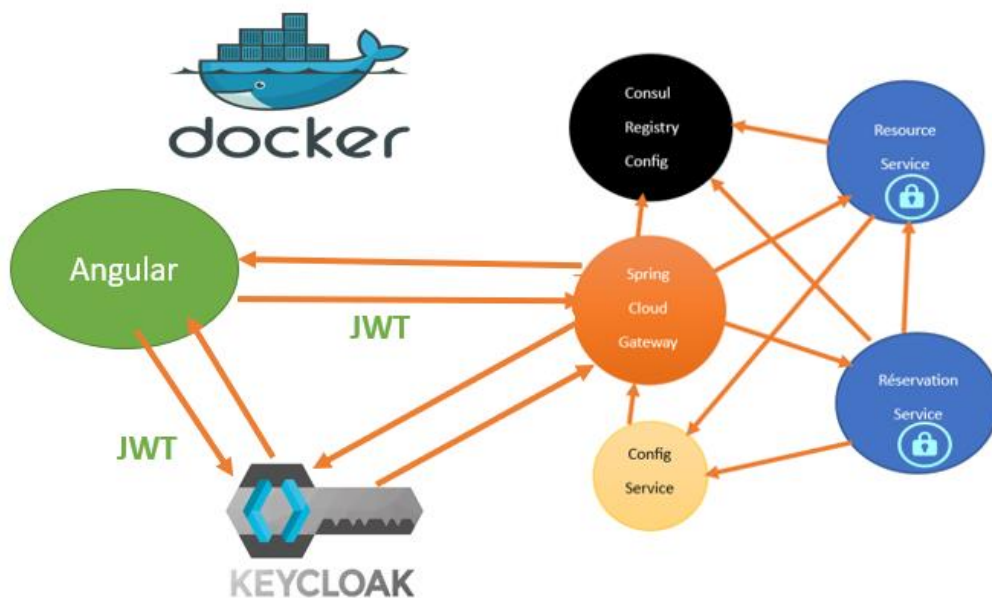
L'architecture du projet repose sur le paradigme micro-service, offrant une approche modulaire et évolutive pour la gestion des réservations. Deux micro-services principaux, le "Resource-Service" et le "Reservation-Service", sont au cœur de cette structure, permettant une gestion indépendante des ressources et des réservations.

La passerelle (Gateway Service) simplifie l'accès aux micro-services depuis le frontend, assurant une interface unifiée et renforçant la sécurité. Le service de découverte (Discovery Service) facilite la localisation et la connexion entre les micro-services, favorisant une communication fluide au sein de l'écosystème.

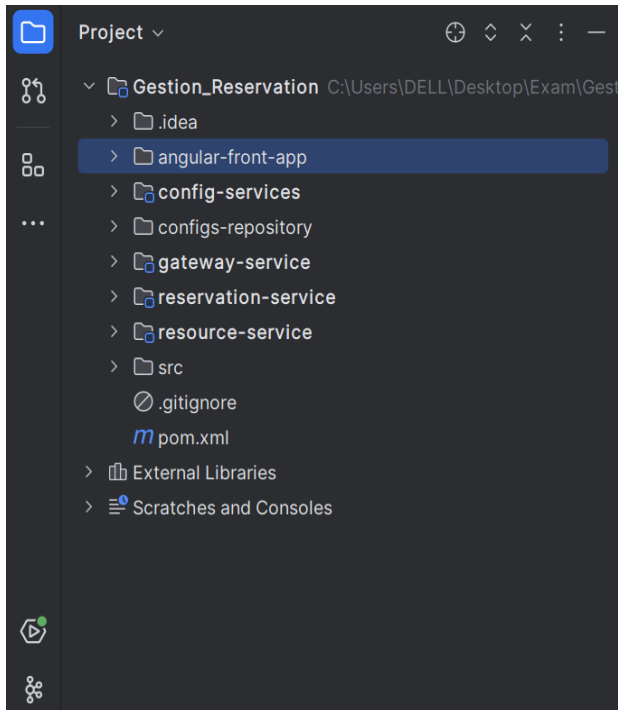
Le service de configuration (Config Service) centralise la gestion des paramètres, apportant une flexibilité accrue dans la gestion des configurations des micro-services. Ces composants travaillent de concert pour créer un environnement distribué et interconnecté.

L'architecture s'appuie sur des technologies modernes telles que Spring Cloud, Angular, Keycloak, et Resilience4J pour assurer la robustesse, la sécurité, et la tolérance aux pannes de l'application. Ce projet adopte ainsi une approche agile et adaptable, prête à répondre aux défis évolutifs du développement logiciel moderne.

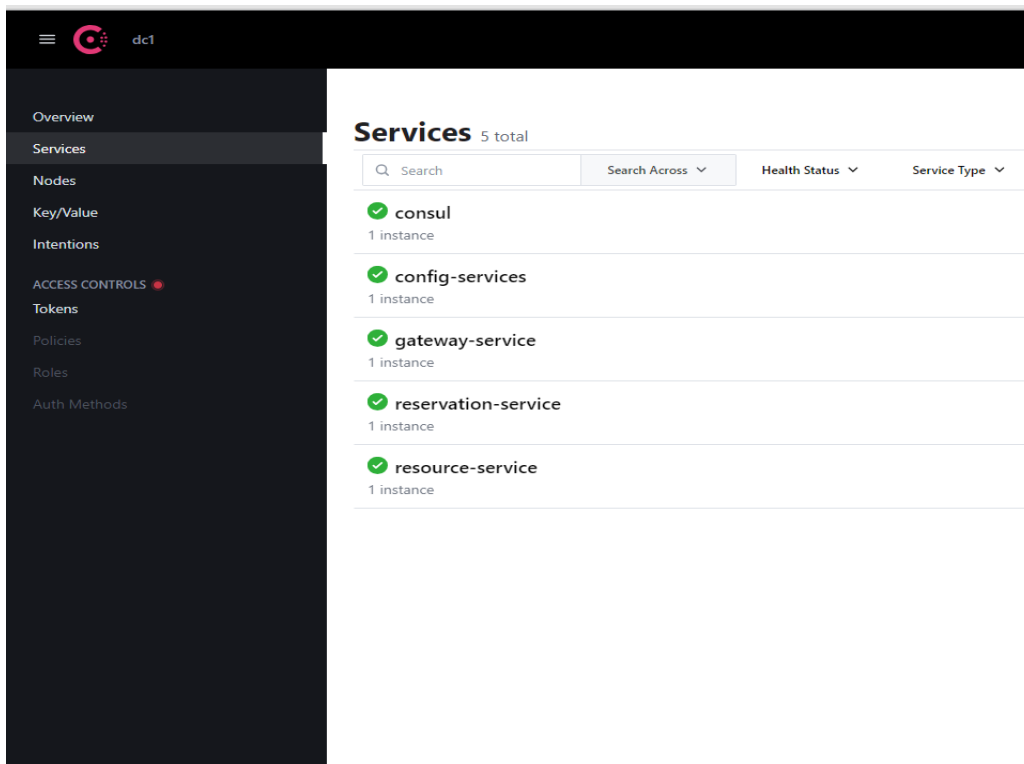
Établir une architecture technique du projet



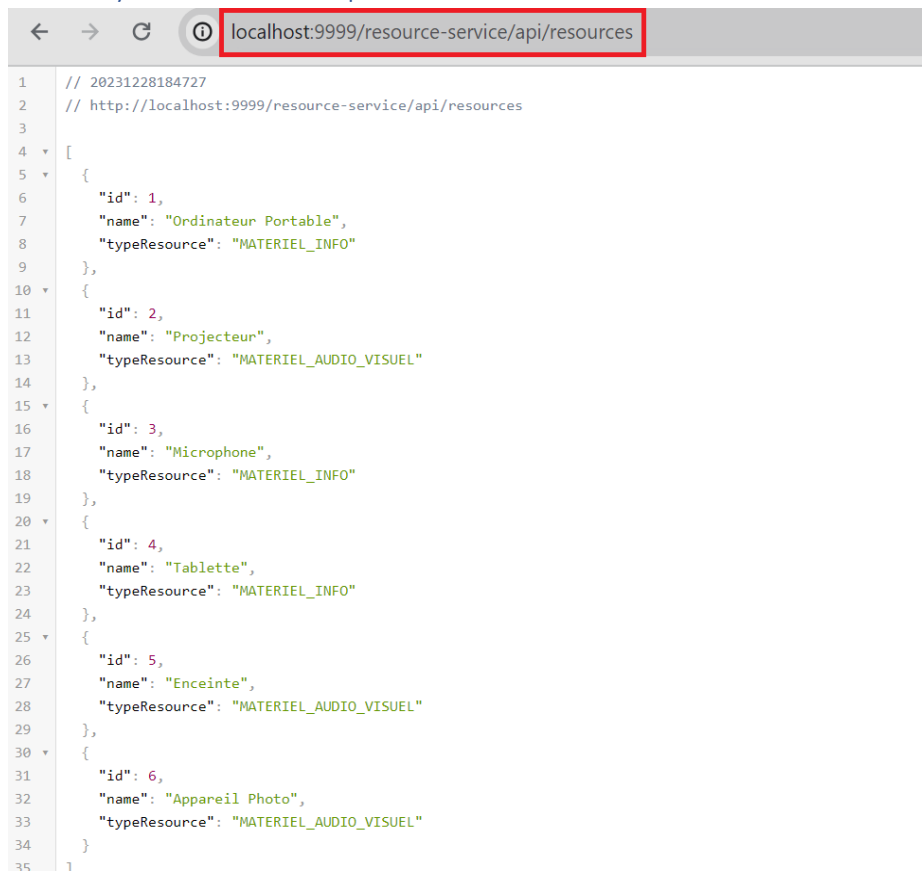
2. Créer un Projet Maven incluant les micro-services suivants : resources-service, reservation-service, gateway-service, discovery-service, config-service et angular-front-app



3. Développer et tester les micro-services discovery-service et gateway-service et config-service

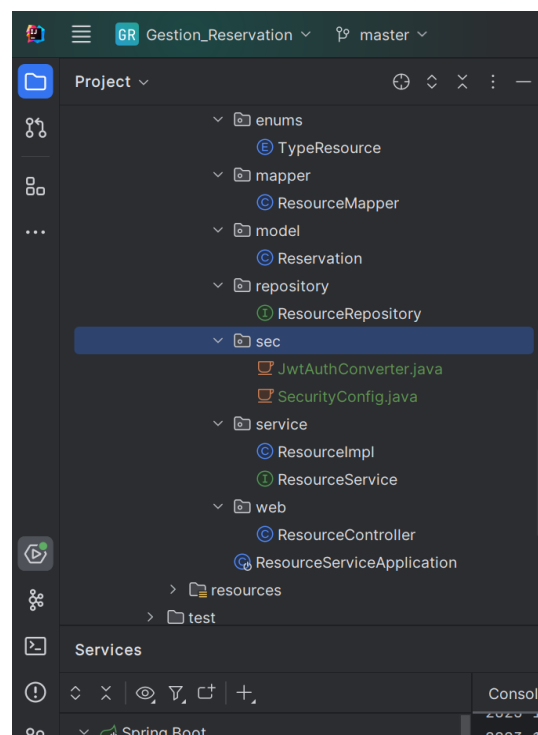
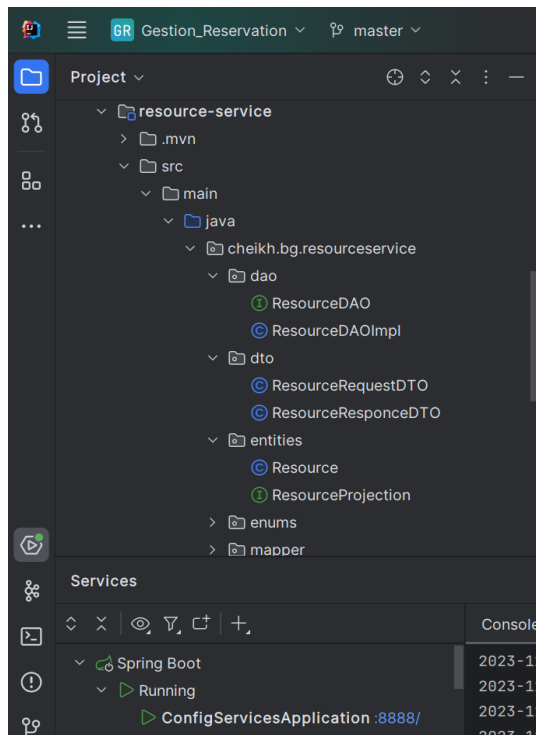


Gateway-service avec le port 9999



```
1 // 20231228184727
2 // http://localhost:9999/resource-service/api/resources
3
4 [
5   {
6     "id": 1,
7     "name": "Ordinateur Portable",
8     "typeResource": "MATERIEL_INFO"
9   },
10  {
11    "id": 2,
12    "name": "Projecteur",
13    "typeResource": "MATERIEL_AUDIO_VISUEL"
14  },
15  {
16    "id": 3,
17    "name": "Microphone",
18    "typeResource": "MATERIEL_INFO"
19  },
20  {
21    "id": 4,
22    "name": "Tablette",
23    "typeResource": "MATERIEL_INFO"
24  },
25  {
26    "id": 5,
27    "name": "Enceinte",
28    "typeResource": "MATERIEL_AUDIO_VISUEL"
29  },
30  {
31    "id": 6,
32    "name": "Appareil Photo",
33    "typeResource": "MATERIEL_AUDIO_VISUEL"
34  }
35 ]
```

4. Développer et tester le micro-service resources-service (Entities, DAO, service, DTO, Mapper, RestController)

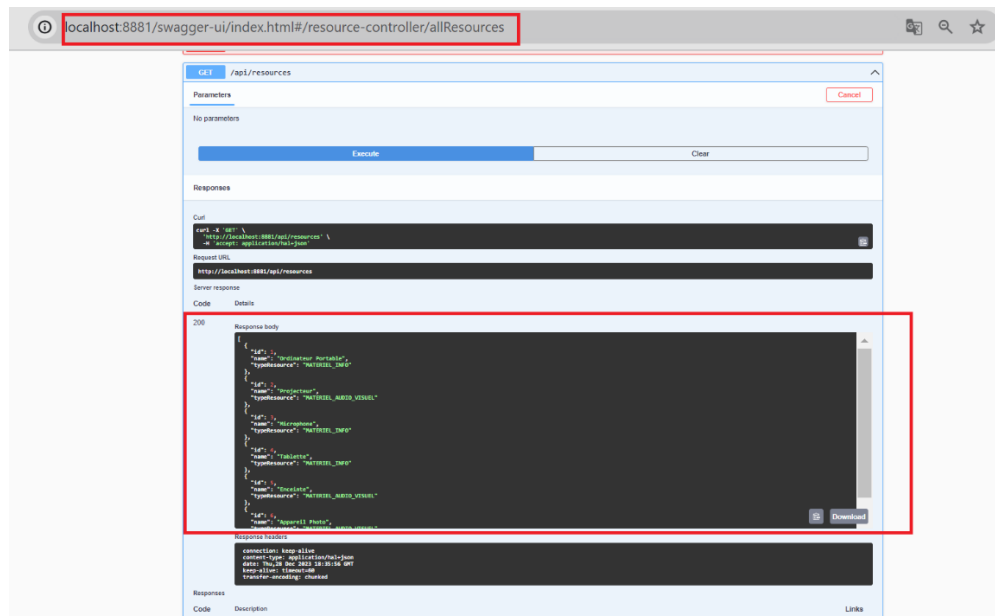


Test with OpenAPIDoc (Swagger)

Here are the methods

| resource-controller | | |
|---------------------|---------------------|---|
| GET | /api/resources/{id} | ✓ |
| PUT | /api/resources/{id} | ✓ |
| DELETE | /api/resources/{id} | ✓ |
| GET | /api/resources | ✓ |
| POST | /api/resources | ✓ |

getAllResources Method

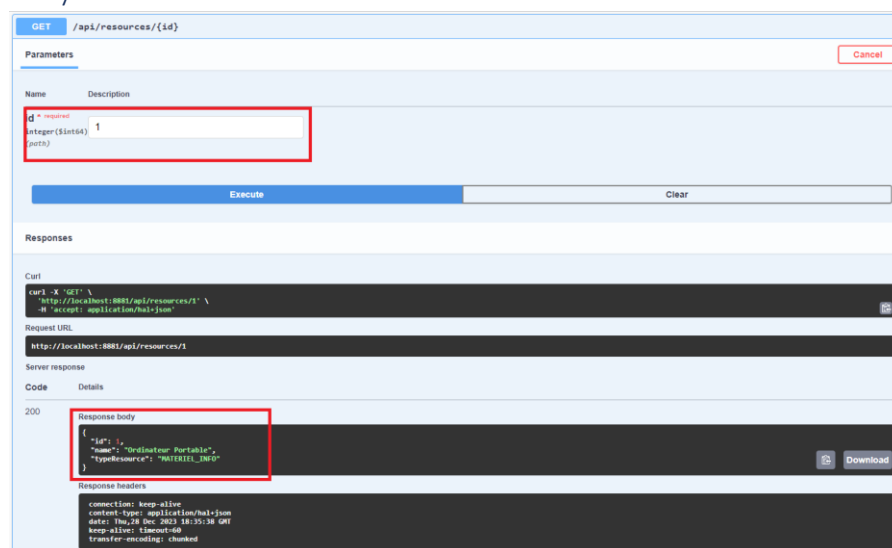


The screenshot shows the Swagger UI for the `getAllResources` method. The URL bar highlights `localhost:8881/swagger-ui/index.html#/resource-controller/allResources`. The interface shows the method `GET /api/resources` with no parameters. The response status is 200. The response body is a JSON array of resource objects, including `Ordinator Portable` and `MATERIEL_INF0`. The response headers show `connection: keep-alive`, `content-type: application/hal+json`, `date: Thu, 28 Dec 2023 18:35:38 GMT`, `keep-alive: timeout=60`, and `transfer-encoding: chunked`.

```
curl -X GET -s -H "Accept: application/hal+json" http://localhost:8881/api/resources
```

```
{
  "id": 1,
  "name": "Ordinator Portable",
  "typeResource": "MATERIEL_INF0"
}
```

getResourceById Method

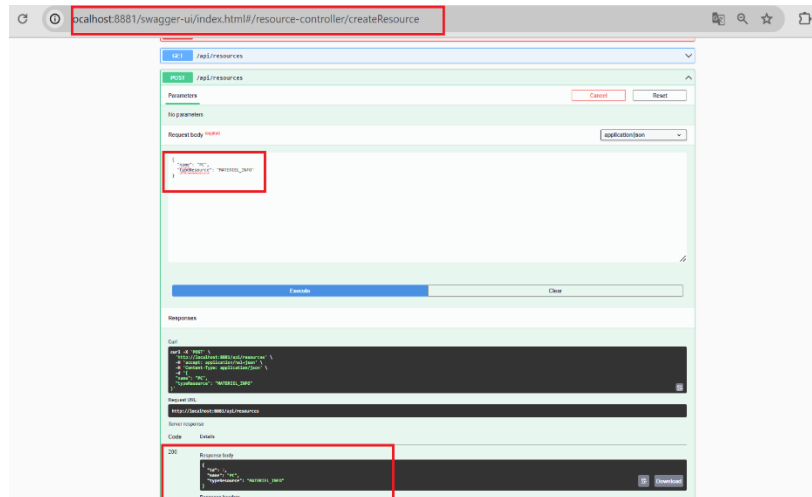


The screenshot shows the Swagger UI for the `getResourceById` method. The URL bar highlights `localhost:8881/swagger-ui/index.html#/resource-controller/getId`. The interface shows the method `GET /api/resources/{id}` with a required parameter `id` of type `integer (int64)`. The response status is 200. The response body is a JSON object representing a resource, including `Ordinator Portable` and `MATERIEL_INF0`. The response headers show `connection: keep-alive`, `content-type: application/hal+json`, `date: Thu, 28 Dec 2023 18:35:38 GMT`, `keep-alive: timeout=60`, and `transfer-encoding: chunked`.

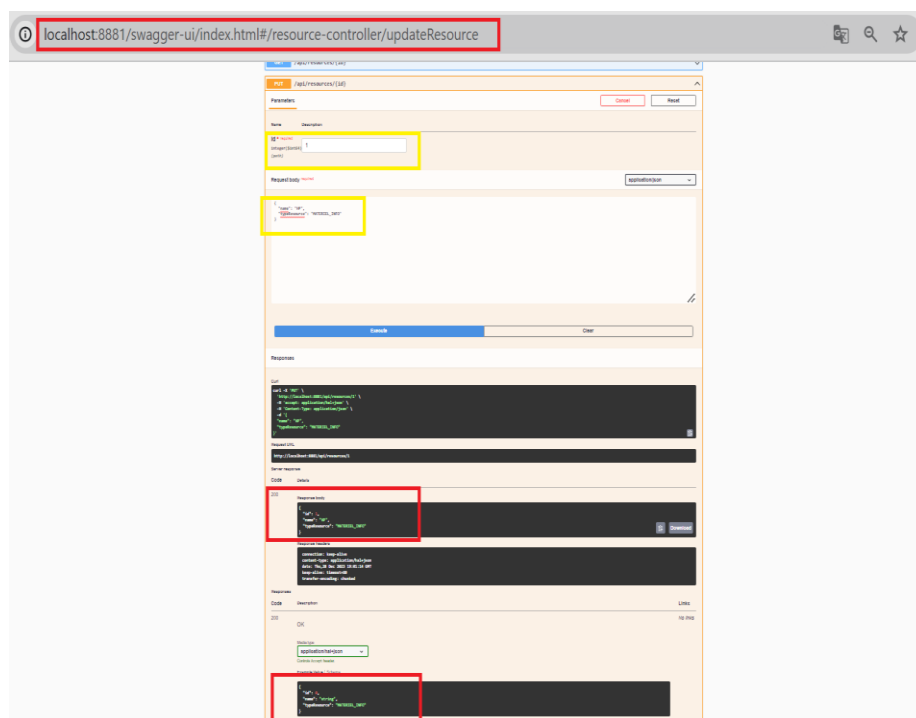
```
curl -X GET -s -H "Accept: application/hal+json" http://localhost:8881/api/resources/1
```

```
{
  "id": 1,
  "name": "Ordinator Portable",
  "typeResource": "MATERIEL_INF0"
}
```

addResource Method

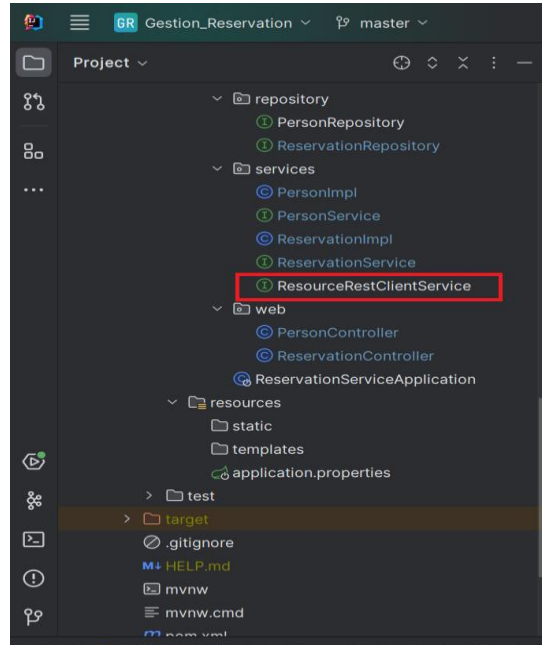
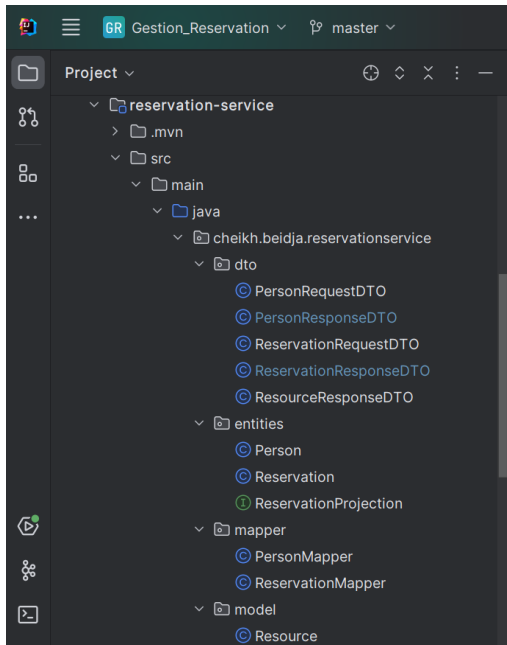


updateResource Method



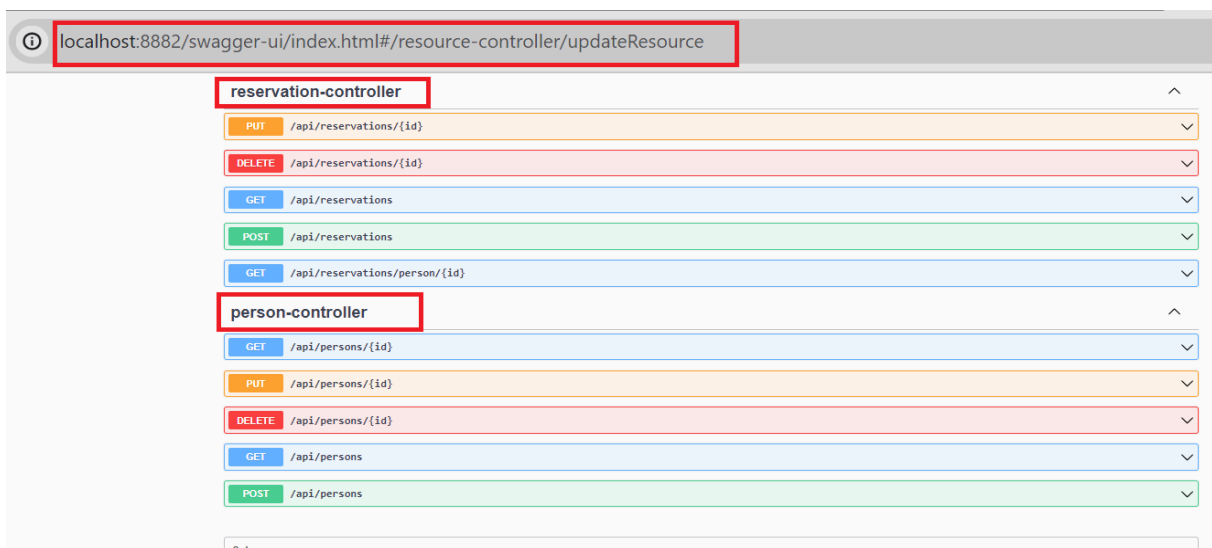
5. Développer et tester le micro-service reservation-service (Entities, DAO, service, DTO, Mapper,

RestController, Client Rest Open Feign)

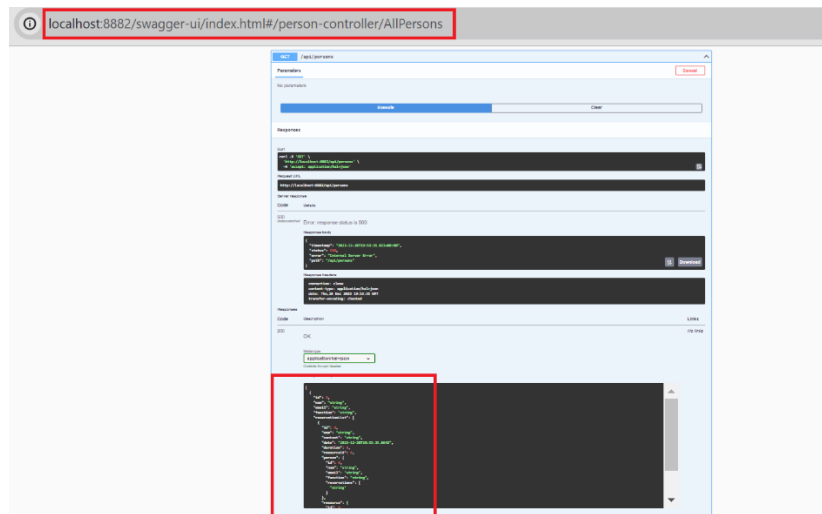


Test with OpenAPIDoc (**Swagger**)

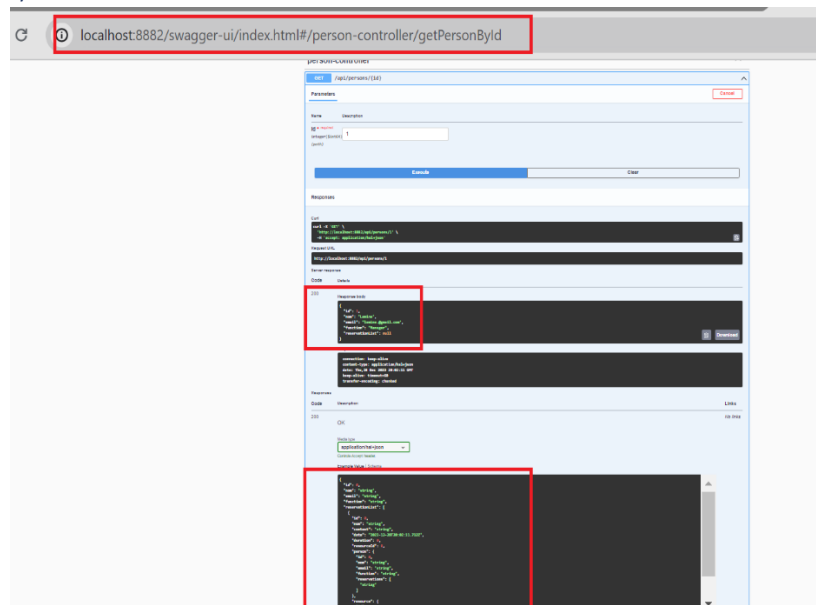
Here are the methods



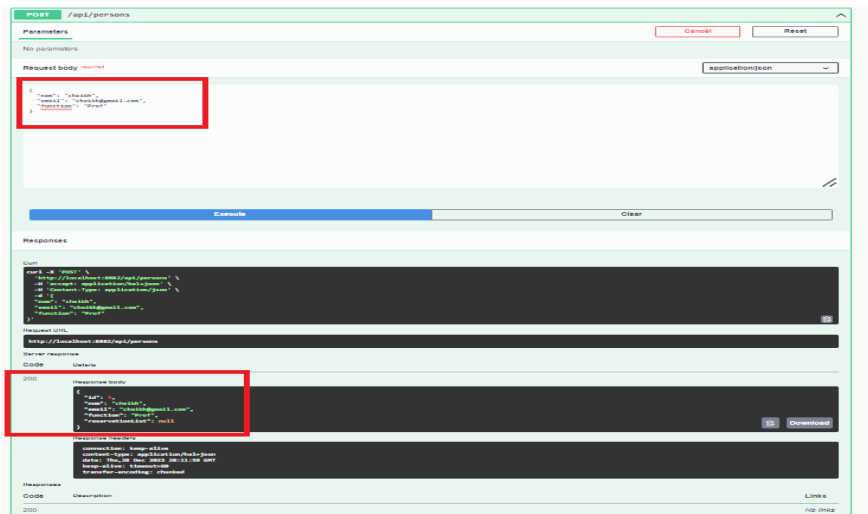
getAllPersons Method



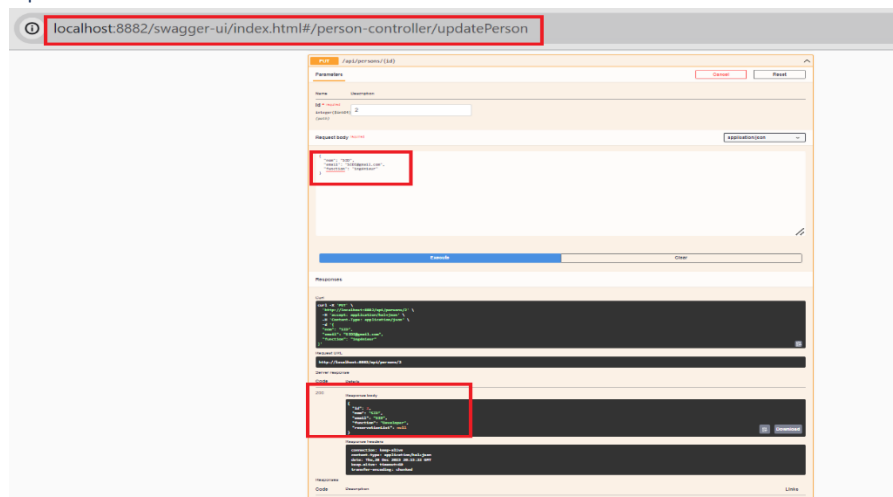
getPersonByld Mthod



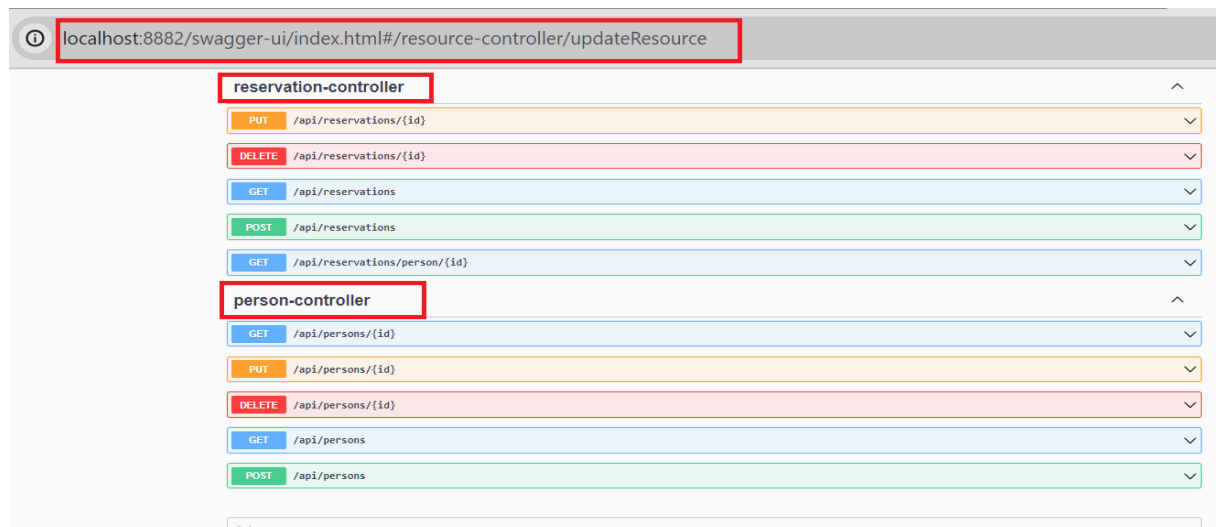
addPerson Method



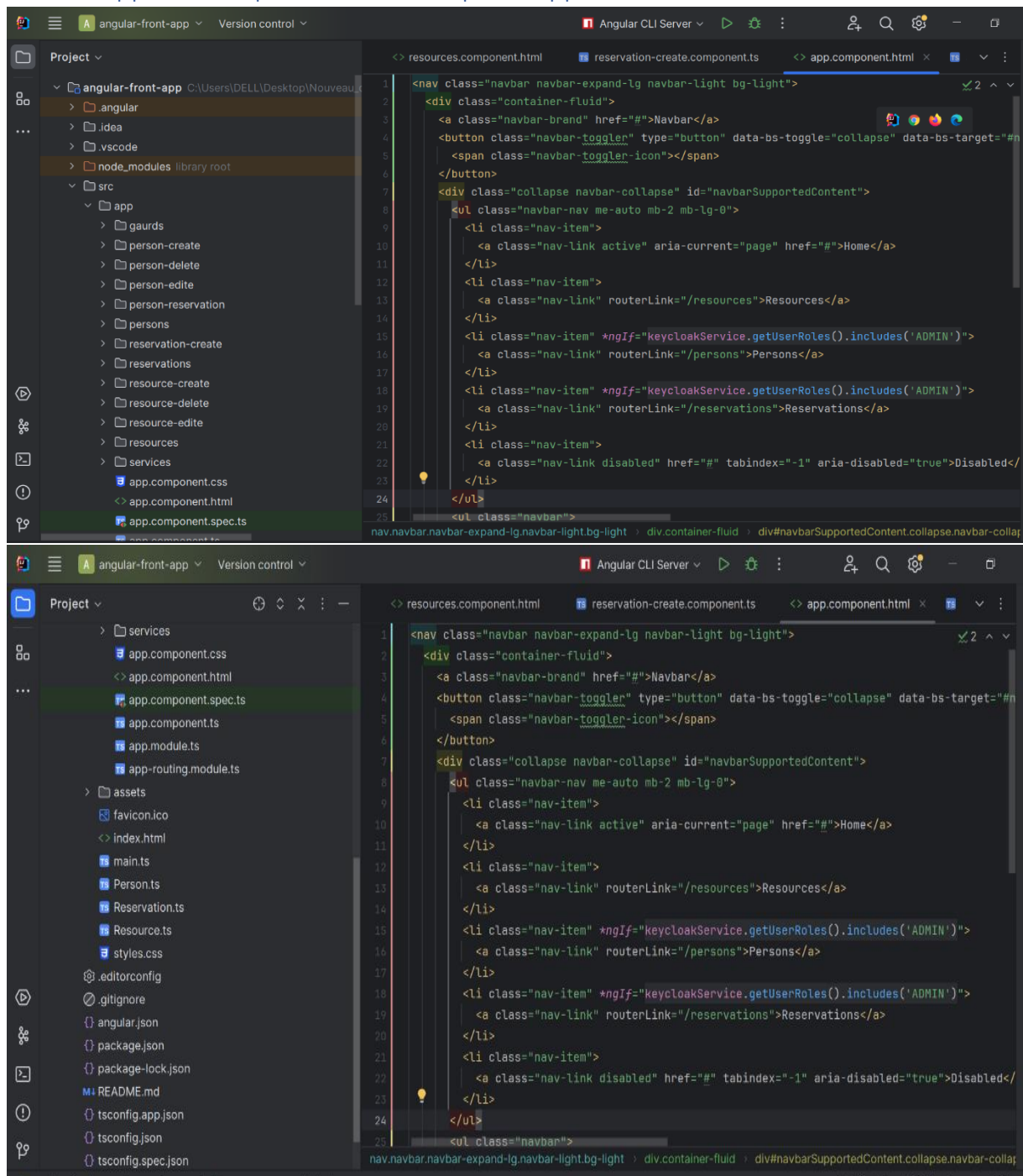
updatePerson Method



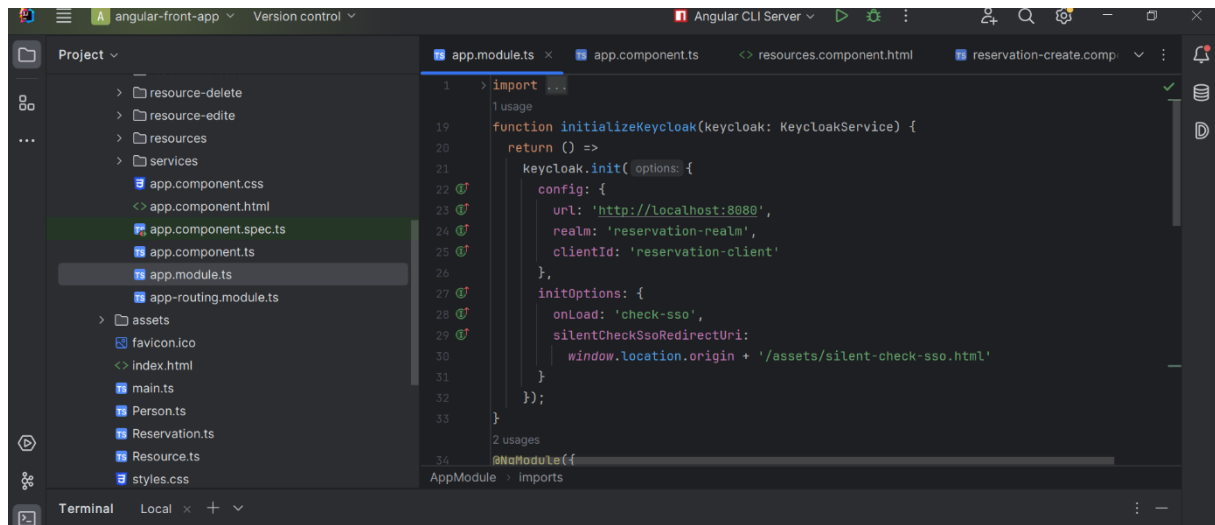
reservations



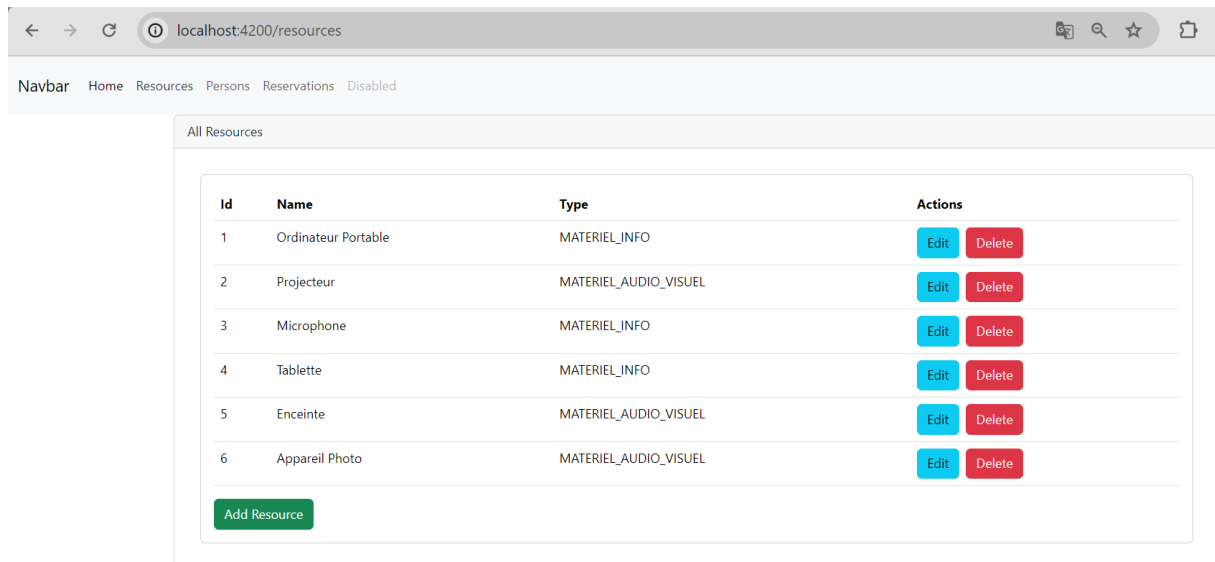
6. Développer un simple frontend web pour l'application



app.module.ts



```
1 > import { NgModule } from '@angular/core';
2
3 function initializeKeycloak(keycloak: KeycloakService) {
4   return () => {
5     keycloak.init({ options: {
6       config: {
7         url: 'http://localhost:8080',
8         realm: 'reservation-realm',
9         clientId: 'reservation-client'
10       },
11       initOptions: {
12         onLoad: 'check-sso',
13         silentCheckSsoRedirectUri:
14           window.location.origin + '/assets/silent-check-sso.html'
15       }
16     });
17   };
18 }
19
20 @NgModule({
21   imports: [
22     // ...
23   ],
24 })
25 export class AppModule {}
```



localhost:4200/resources

Navbar Home Resources Persons Reservations Disabled

All Resources

| Id | Name | Type | Actions |
|----|---------------------|-----------------------|---|
| 1 | Ordinateur Portable | MATERIEL_INFO | <button>Edit</button> <button>Delete</button> |
| 2 | Projecteur | MATERIEL_AUDIO_VISUEL | <button>Edit</button> <button>Delete</button> |
| 3 | Microphone | MATERIEL_INFO | <button>Edit</button> <button>Delete</button> |
| 4 | Tablette | MATERIEL_INFO | <button>Edit</button> <button>Delete</button> |
| 5 | Enceinte | MATERIEL_AUDIO_VISUEL | <button>Edit</button> <button>Delete</button> |
| 6 | Appareil Photo | MATERIEL_AUDIO_VISUEL | <button>Edit</button> <button>Delete</button> |

Add Resource

7. Sécuriser l'application avec une authentification Keycloak

Keycloak

The screenshot shows the Keycloak administration interface. On the left is a sidebar with navigation options like 'Manage', 'Clients', 'Client scopes', etc. The main area displays the 'Clients' page with a table of existing clients. A terminal window on the right shows the logs of a Quarkus application starting up, including messages about registering classes, starting the application, and logging in.

| Client ID | Name | Type | Description |
|------------------------|-----------------------------------|----------------|-------------|
| account | \$(client_account) | OpenID Connect | - |
| account-console | \$(client_account-console) | OpenID Connect | - |
| admin-cli | \$(client_admin-cli) | OpenID Connect | - |
| broker | \$(client_broker) | OpenID Connect | - |
| realm-management | \$(client_realm-management) | OpenID Connect | - |
| reservation-client | - | OpenID Connect | - |
| resource-client | - | OpenID Connect | - |
| security-admin-console | \$(client_security-admin-console) | OpenID Connect | - |

SECURITÉ DES MICROSERVICES

package cheikh.bg.resourceservice.sec;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.Customizer;

import

org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.web.cors.CorsConfiguration;

import org.springframework.web.cors.CorsConfigurationSource;

import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.Arrays;

@Configuration

@EnableWebSecurity

@EnableMethodSecurity(prePostEnabled = true)

public class SecurityConfig {

 public SecurityConfig(JwtAuthConverter jwtAuthConverter) {

 this.jwtAuthConverter = jwtAuthConverter;

 }

 private JwtAuthConverter jwtAuthConverter;

 @Bean

 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

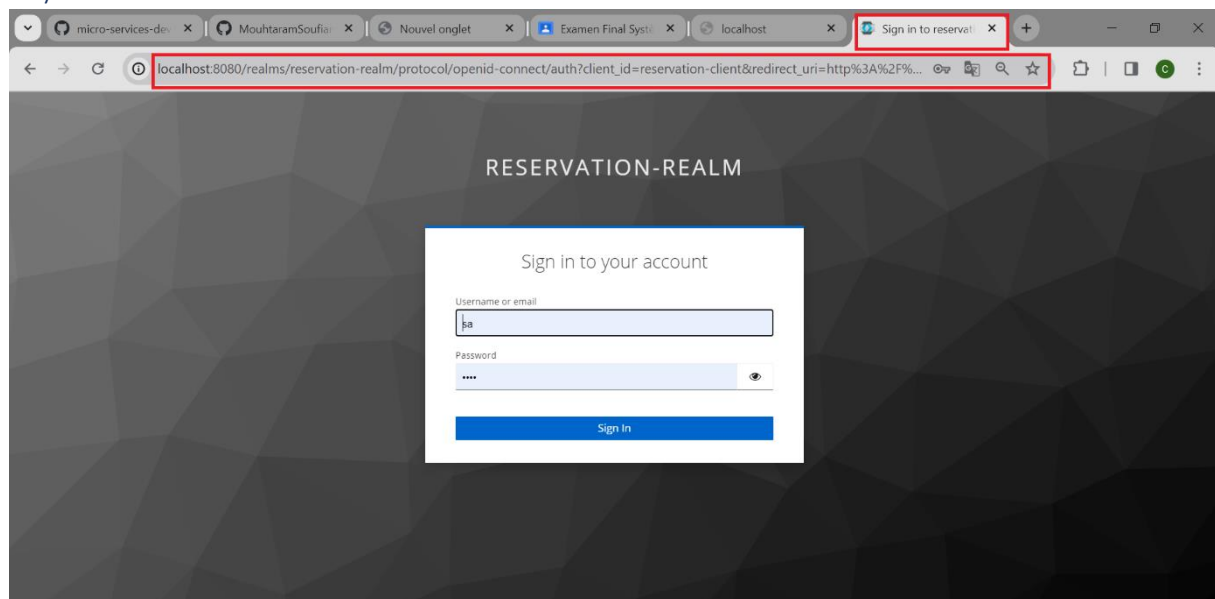
 return http


```

        //cors(Customizer.withDefaults())
        //authorizeHttpRequests(ar->ar.requestMatchers("/resources/**").permitAll())
        .authorizeHttpRequests(ar -> ar.anyRequest().authenticated())
        .oauth2ResourceServer(o2 -> o2.jwt(jw-
>jw.jwtAuthenticationConverter(jwtAuthConverter)))
        .headers(h->h.frameOptions(fo->fo.disable()))
        .csrf(csrf->csrf.ignoringRequestMatchers("/h2-console/**"))
        .build();
    }
    //cors problem
    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(Arrays.asList("*"));
        configuration.setAllowedMethods(Arrays.asList("*"));
        configuration.setAllowedHeaders(Arrays.asList("*"));
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source; }
}

```

Keycloak



Gestion des ressources

localhost:4200/resources

Navbar Home Resources Persons Reservations Disabled user1

All Resources

| Id | Name | Type | Actions |
|----|---------------------|-----------------------|---|
| 1 | Ordinateur Portable | MATERIEL_INFO | Edit Delete |
| 2 | Projecteur | MATERIEL_AUDIO_VISUEL | Edit Delete |
| 3 | Microphone | MATERIEL_INFO | Edit Delete |
| 4 | Tablette | MATERIEL_INFO | Edit Delete |
| 5 | Enceinte | MATERIEL_AUDIO_VISUEL | Edit Delete |
| 6 | Appareil Photo | MATERIEL_AUDIO_VISUEL | Edit Delete |

[Add Resource](#)

localhost:4200/resources/create

Navbar Home Resources Persons Reservations Disabled

Create Resource

Name

Type

Select a type

[Submit](#)

localhost:4200/resources/edite/1

Navbar Home Resources Persons Reservations Disabled

Update Resource

Name

Ordinateur Portable

Type

Ordinateur Portabled

MATERIEL_INFO

[Submit](#)

Gestion des persons

localhost:4200/persons/create

Navbar Home Resources Persons Reservations Disabled

Add Resource

Name
ckeikh

Type
cheikh@gmail.com

Type
manangement

Submit

localhost:4200/persons

Navbar Home Resources Persons Reservations Disabled

All Persons

| Id | Name | Email | Function | Actions |
|----|--------|------------------|-------------|---|
| 1 | ckeikh | cheikh@gmail.com | manangement | Edit Delete Reservation |

[Add Person](#)

localhost:4200/persons/edite/1

Navbar Home Resources Persons Reservations Disabled

Update Person

Name
ckeikh

Type
cheikh@gmail.com

Type
function

Submit

Gestion des réservations

localhost:4200/reservations/create/1

Navbar Home Resources Persons Reservations Disabled

Add Resource

Name

Context

Duration

ffxyssfds
blabla

ResourceId

PersonId

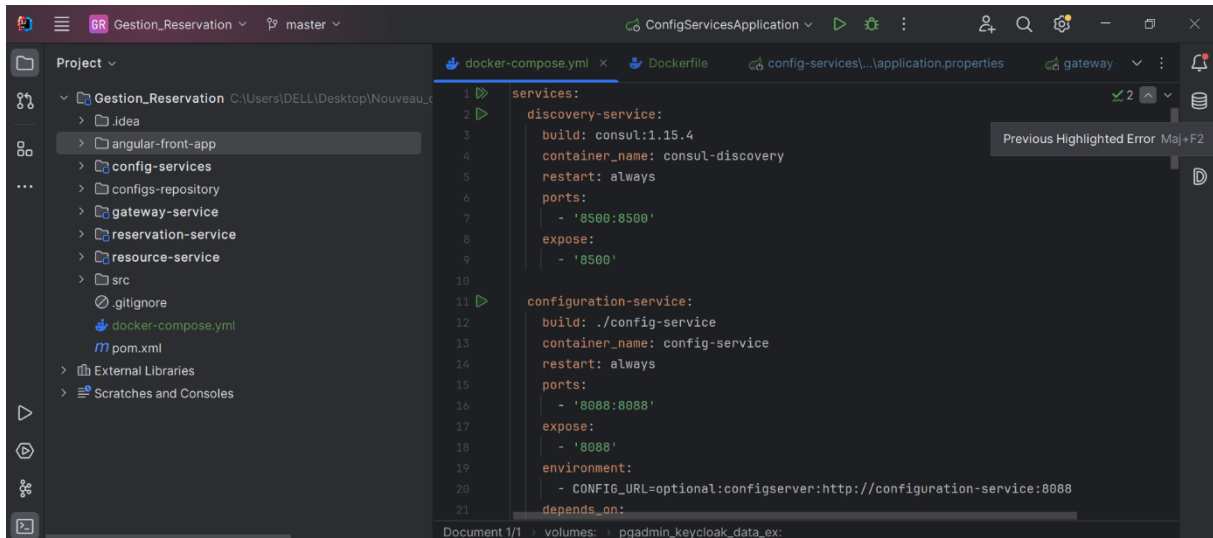
localhost:4200/reservations

Navbar Home Resources Persons Reservations Disabled • Cheikh Beidj

All Reservations

| Id | Name | Context | Date | Duration | Actions |
|----|------------|---------------|-------------------------------|----------|---|
| 1 | Meeting | Education | 2023-12-30T22:36:06.135+00:00 | 1 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 2 | Conference | Technology | 2023-12-30T22:36:06.141+00:00 | 6 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 3 | Workshop | Healthcare | 2023-12-30T22:36:06.146+00:00 | 2 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 4 | Training | Entertainment | 2023-12-30T22:36:06.149+00:00 | 3 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 5 | Meeting | Education | 2023-12-30T22:36:06.150+00:00 | 3 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 6 | Conference | Technology | 2023-12-30T22:36:06.154+00:00 | 10 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 7 | Workshop | Healthcare | 2023-12-30T22:36:06.158+00:00 | 7 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 8 | Training | Entertainment | 2023-12-30T22:36:06.161+00:00 | 9 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 9 | Meeting | Education | 2023-12-30T22:36:06.164+00:00 | 1 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 10 | Conference | Technology | 2023-12-30T22:36:06.168+00:00 | 10 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |
| 11 | Workshop | Healthcare | 2023-12-30T22:36:06.171+00:00 | 10 | <input type="button" value="Edit"/> <input type="button" value="Delete"/> |

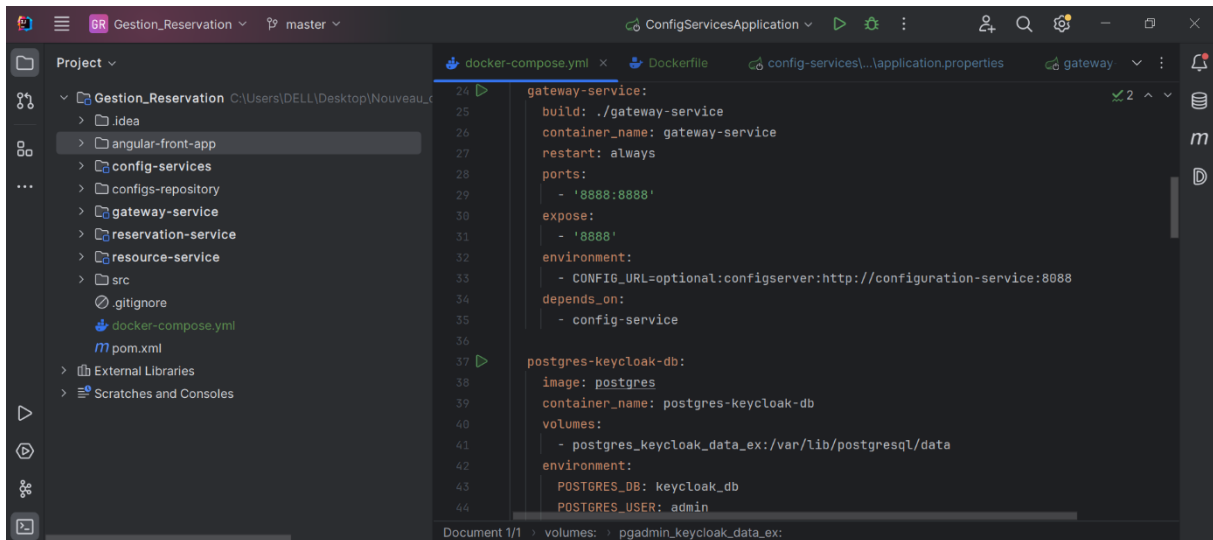
8. Déployer l'application avec Docker et Docker compose



The screenshot shows the IntelliJ IDEA interface with the 'Gestion_Reservation' project. The 'docker-compose.yml' file is open, displaying the following configuration:

```
1 services:
2   discovery-service:
3     build: consul:1.15.4
4     container_name: consul-discovery
5     restart: always
6     ports:
7       - '8500:8500'
8     expose:
9       - '8500'
10
11   configuration-service:
12     build: ./config-service
13     container_name: config-service
14     restart: always
15     ports:
16       - '8088:8088'
17     expose:
18       - '8088'
19     environment:
20       - CONFIG_URL=optional:configserver:http://configuration-service:8088
21     depends_on:
```

The status bar at the bottom indicates 'Document 1/1' and 'volumes: pgadmin_keycloak_data_ex:'.



The screenshot shows the IntelliJ IDEA interface with the 'Gestion_Reservation' project. The 'docker-compose.yml' file is open, displaying the following configuration:

```
24 gateway-service:
25   build: ./gateway-service
26   container_name: gateway-service
27   restart: always
28   ports:
29     - '8888:8888'
30   expose:
31     - '8888'
32   environment:
33     - CONFIG_URL=optional:configserver:http://configuration-service:8088
34   depends_on:
35     - config-service
36
37 postgres-keycloak-db:
38   image: postgres
39   container_name: postgres-keycloak-db
40   volumes:
41     - postgres_keycloak_data_ex:/var/lib/postgresql/data
42   environment:
43     POSTGRES_DB: keycloak_db
44     POSTGRES_USER: admin
```

The status bar at the bottom indicates 'Document 1/1' and 'volumes: pgadmin_keycloak_data_ex:'.

