



Examen Systèmes Distribués

Master SDIA 2023 / 2024

Prepared By:
Mahfoud Sanaa

Table des matières

I.	Introduction.....	4
II.	Travail demandé	4
1.	On établie une architecture technique du projet	4
2.	On créer un projet Maven incluant les micro-services suivants : resources-service, reservation-service, gateway-service, discovery-service, config-service et angular-front-app :	5
3.	On Développe et on teste les micro-services discovery-service et gateway-service et config-service.....	5
4.	On développe et on teste le micro-service resources-service (Entities, DAO, service, DTO, Mapper, RestController)	9
5.	On développe et on teste le micro-service reservation-service (Entities, DAO, service, DTO, Mapper, RestController, Client Rest Open Feign).....	14
6.	On développe un simple frontend web Angular pour l'application :	20
7.	On sécurise l'application avec une authentification Keycloak :.....	32
8.	Déploiement de l'application avec Docker et Docker compose.....	37
III.	Conclusion	42

Examen Systèmes Distribués

On souhaite créer une application basée sur une architecture micro-service qui permet de gérer des réservations concernant des ressources. Chaque réservation concerne une seule ressource. Une ressource est définie par son id, son nom, son type (MATERIEL_INFO, MATERIEL_AUDIO_VUSUEL). Une réservation est définie par son id, son nom, son contexte, , sa date, sa durée. Chaque réservation est effectuée par une personne. Une personne est définie par son id, son nom, son email et sa fonction.

L’application doit permettre de gérer les ressources et les réservations. Pour faire plus simple, cette application se composera de deux micro-services fonctionnels :

- Un Micro-service qui permet de gérer des « Resources-Service ».
- Un Micro-service qui permet de gérer les réservations effectuées par des personnes.

Les micro-services technique à mettre en place sont :

- Le service Gateway basé sur Spring cloud Gateway
- Le service Discovery base sur Eureka Server ou Consul Discovery (au choix)
- Le service de configuration basé sur Spring cloud config ou Consul Config (au choix)

Pour l’application, nous avons besoin de développer une frontend web, basé sur Angular Framework.

La sécurité de l’application est basée sur Oauth2 et OIDC avec Keycloak comme Provider

Pour les micro-services, il faut générer la documentation des web services Restfull en utilisant la spécification OpenAPIDoc (Swagger). Prévoir aussi des circuit breakers basés sur Resilience4J comme solution de fault Tolerance.

Travail demandé :

Rendre un rapport PDF et le code source des projets au format zip, répondant aux questions suivantes

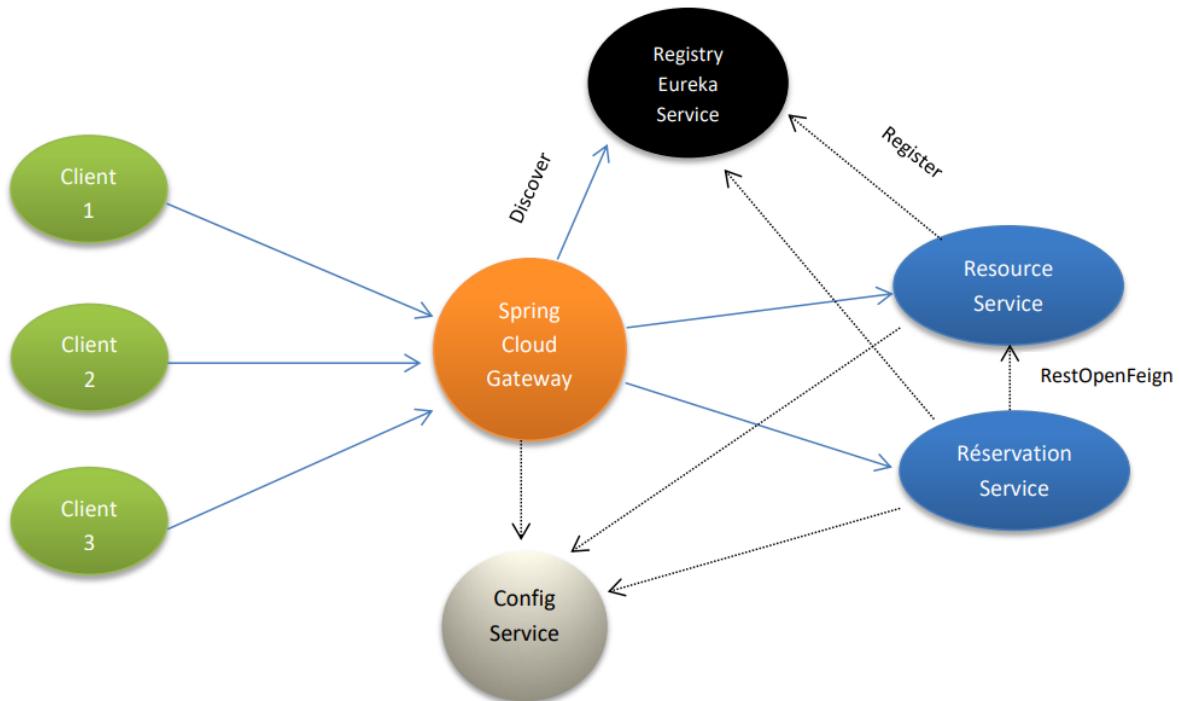
1. Établir une architecture technique du projet
2. Créer un Projet Maven incluant les micro-services suivants : resources-service, reservation-service, gateway-service, discovery-service, config-service et angular-front-app
3. Développer et tester les micro-services discovery-service et gateway-service et config-service
4. Développer et tester le micro-service resources-service (Entities, DAO, service, DTO, Mapper, RestController)
5. Développer et tester le micro-service reservation-service (Entities, DAO, service, DTO, Mapper, RestController, Client Rest Open Feign)
6. Développer un simple frontend web pour l’application
7. Sécuriser l’application avec une authentification Keycloak
8. Déployer l’application avec Docker et Docker compose

I. Introduction

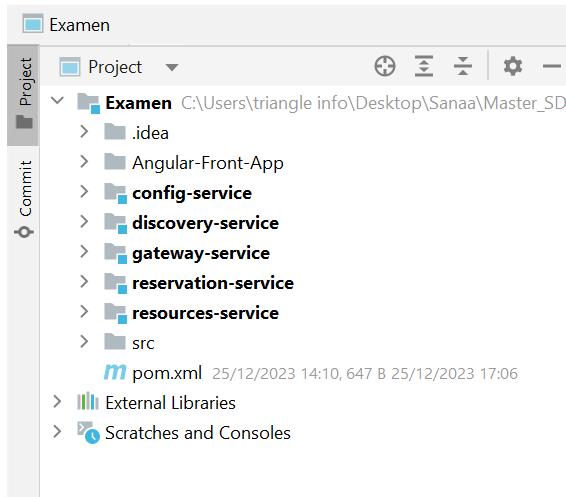
Dans le cadre de ce projet ambitieux, nous envisageons la création d'une application de gestion de réservations basée sur une architecture micro-service, offrant une approche modulaire et évolutive. Les deux piliers fonctionnels de cette application, le service de gestion des ressources (resources-service) et le service de gestion des réservations (reservation-service), seront soutenus par des composants clés tels que le service de passerelle (gateway-service), le service de découverte (discovery-service), et le service de configuration (config-service). En parallèle, nous intégrerons un frontend web dynamique, développé avec le framework Angular, pour permettre aux utilisateurs d'interagir de manière conviviale avec les fonctionnalités offertes par les micro-services. La sécurité de l'application reposera sur OAuth2 et OIDC, avec Keycloak comme fournisseur d'identité, tandis que la tolérance aux pannes sera assurée par des circuit breakers basés sur Resilience4J. En présentant cette architecture technique robuste, nous prévoyons de répondre de manière exhaustive aux exigences de gestion des ressources et des réservations tout en assurant la sécurité, la documentation approfondie et la résilience du système.

II. Travail demandé

1. On établie une architecture technique du projet



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



3. On Développe et on teste les micro-services discovery-service et gateway-service et config-service

Discovery service

- ✓ Application.properties

```
server.port=8761
eureka.client.fetch-registry=false
eureka.client.register-with-eureka=false
```

- ✓ DiscoveryServiceApplication

```
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(DiscoveryServiceApplication.class, args);
    }
}
```

The screenshot shows the Spring Eureka dashboard at localhost:8761. The top navigation bar includes links for "Interface graphique...", "Méthode d'extraction...", "Examinons un peu...", "New chat", "(2) WhatsApp", "4A_CoursImage.pdf", and "Recherche d'images...". The dashboard header features the "spring Eureka" logo and "HOME LAST 1000 SINCE STARTUP". The "System Status" section displays environment details (Environment: test, Data center: default), current time (2023-12-25T16:31:25 +0100), uptime (00:03), lease expiration settings (Lease expiration enabled: false), and renew thresholds (Renews threshold: 6, Renews (last min): 3). The "DS Replicas" section lists instances registered with Eureka, with a link to "Activer Windows".

Gateway service

✓ application.properties

```
spring.application.name=gateway-service
server.port=9999
eureka.instance.prefer-ip-address=true
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

✓ application.yml

```
spring:
  cloud:
    gateway:
      globalcors:
        corsConfigurations:
          '//**':
            allowedOrigins: "*"
            allowedHeaders: "*"
            allowedMethods:
              - GET
              - POST
              - PUT
              - DELETE
```

✓ GatewayServiceApplication

```
@SpringBootApplication
public class GatewayServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(GatewayServiceApplication.class, args);
    }
    @Bean
    DiscoveryClientRouteDefinitionLocator locator(ReactiveDiscoveryClient
rdc, DiscoveryLocatorProperties dlp){
        return new DiscoveryClientRouteDefinitionLocator(rdc,dlp);
    }
}
```

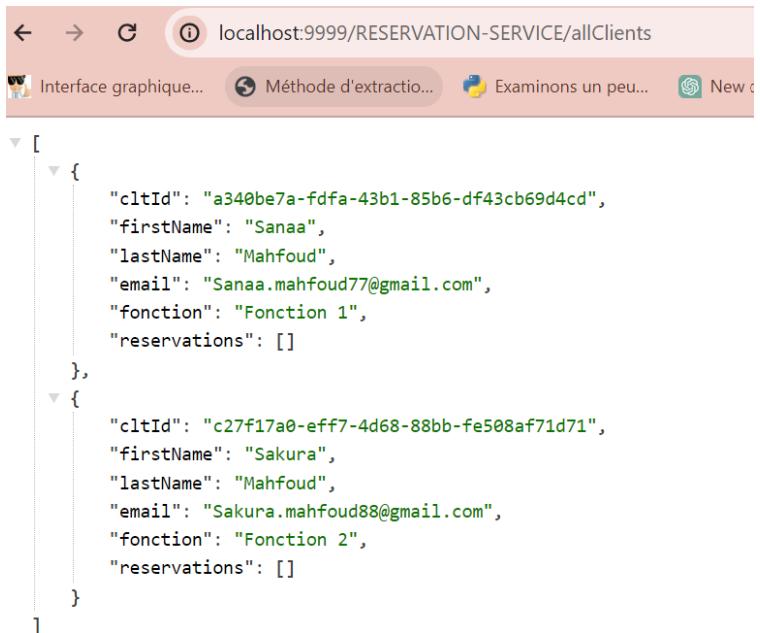
➤ On affiche les réservations



The screenshot shows a browser window with the URL `localhost:9999/RESERVATION-SERVICE/reservations`. The page displays a JSON array of four reservation objects. Each object has properties: `resId`, `resNom`, `resContexte`, `createRes`, `resDuree`, `client`, `clientId`, `resource`, and `resourceId`. The data is as follows:

```
[{"resId": 1, "resNom": "Reservation 1", "resContexte": "Context 1", "createRes": "2023-12-27", "resDuree": 12, "client": null, "clientId": "a340be7a-fdfa-43b1-85b6-df43cb69d4cd", "resource": null, "resourceId": 1}, {"resId": 2, "resNom": "Reservation 2", "resContexte": "Context 2", "createRes": "2023-12-27", "resDuree": 4, "client": null, "clientId": "c27f17a0-eff7-4d68-88bb-fe508af71d71", "resource": null, "resourceId": 2}, {"resId": 3, "resNom": "Reservation 3", "resContexte": "Context 3", "createRes": "2023-12-27", "resDuree": 4, "client": null, "clientId": "a340be7a-fdfa-43b1-85b6-df43cb69d4cd", "resource": null, "resourceId": 2}, {"resId": 4, "resNom": "Reservation 4", "resContexte": "Context 4", "createRes": "2023-12-27", "resDuree": 4, "client": null, "clientId": "a340be7a-fdfa-43b1-85b6-df43cb69d4cd", "resource": null, "resourceId": 1}]
```

➤ On affiche les clients



A screenshot of a browser window displaying a JSON response from the URL `localhost:9999/RESERVATION-SERVICE/allClients`. The JSON data represents two client objects:

```
[{"cltId": "a340be7a-fdfa-43b1-85b6-df43cb69d4cd", "firstName": "Sanaa", "lastName": "Mahfoud", "email": "Sanaa.mahfoud77@gmail.com", "fonction": "Fonction 1", "reservations": []}, {"cltId": "c27f17a0-eff7-4d68-88bb-fe508af71d71", "firstName": "Sakura", "lastName": "Mahfoud", "email": "Sakura.mahfoud88@gmail.com", "fonction": "Fonction 2", "reservations": []}]
```

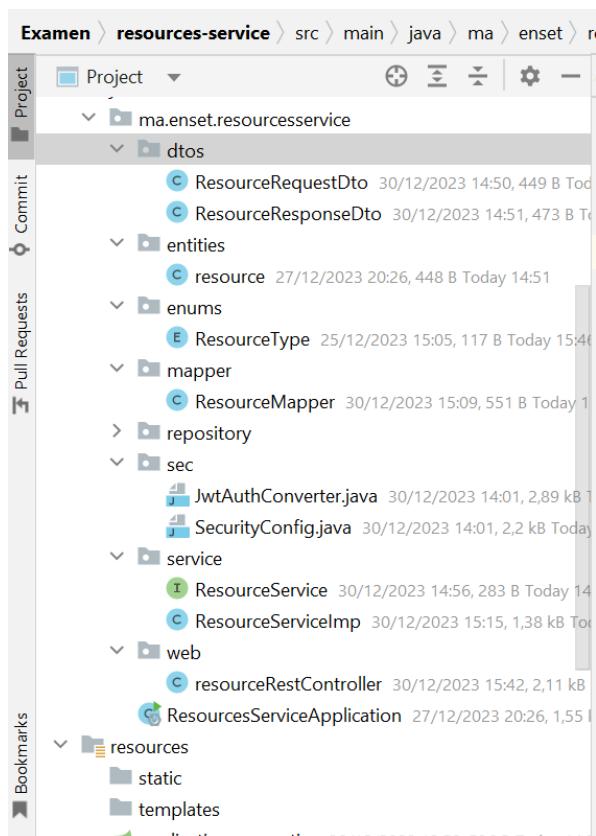
➤ On affiche les resources



A screenshot of a browser window displaying a JSON response from the URL `localhost:9999/RESOURCE-SERVICE/resources`. The JSON data represents two resource objects:

```
[{"resId": 1, "resNom": "resource 1", "type": "MATERIEL_INFO"}, {"resId": 2, "resNom": "resource 2", "type": "MATERIEL_AUDIO_VISUEL"}]
```

- On développe et on teste le micro-service resources-service (Entities, DAO, service, DTO, Mapper, RestController)
- On créer un projet spring contiens toutes le classes nécessaire



✓ application.properties

```
spring.application.name=resource-service
server.port=8081
spring.datasource.url=jdbc:h2:mem:resource-db
spring.h2.console.enabled=true
spring.cloud.discovery.enabled=true
spring.cloud.config.enabled=false
#eureka.instance.prefer-ip-address=true
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

✓ ResourcesServiceApplication

```
@SpringBootApplication
@EnableFeignClients
public class ResourcesServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(ResourcesServiceApplication.class, args);
    }

    @Bean
    CommandLineRunner commandLineRunner(resourceRepository
resourceRepository) {
        return args -> {
            List<resource> reourceList = List.of(
                resource.builder()
                    // .resId(UUID.randomUUID().toString())
                    .resNom("resource 1")
                    .type(ResourceType.MATERIEL_INFO)
                    .build(),
                resource.builder()
                    // .resId(UUID.randomUUID().toString())
                    .resNom("resource 2")
                    .type(ResourceType.MATERIEL_AUDIO_VSUEL)
                    .build()

            );
            resourceRepository.saveAll(reourceList) ;
        };
    }
}
```

✓ On génère la documentation des web services Restfull en utilisant la spécification OpenAPIDoc (Swagger).

The screenshot shows the Swagger UI interface. At the top, there's a navigation bar with the Swagger logo, the URL '/v3/api-docs', and a green 'Explore' button. Below the navigation, the title 'OpenAPI definition' is displayed, along with a '1.0' version indicator and an 'OAS 3.0' badge. A link to '/v3/api-docs' is also present. The main content area is titled 'profile-controller'. It shows two API endpoints: a 'GET /profile' endpoint and a 'GET /profile/resources' endpoint. Both endpoints have dropdown menus next to them. The 'GET /profile' endpoint has a dropdown menu with the URL 'http://localhost:8081 - Generated server url'. The 'GET /profile/resources' endpoint has a dropdown menu with the text 'Activer Windows'. There are also small '^' and 'v' icons for collapsing and expanding sections.

resource-entity-controller

- GET** /resources
- POST** /resources
- GET** /resources/{id}
- PUT** /resources/{id}
- DELETE** /resources/{id}
- PATCH** /resources/{id}

➤ On test la méthode **GET** /resources

Request URL
http://localhost:8081/resources?page=0&size=20

Server response

Code	Details
200	<p>Response body</p> <pre>[{ "resId": 1, "resNom": "resource 1", "type": "MATERIEL_INFO" }, { "resId": 2, "resNom": "resource 2", "type": "MATERIEL_AUDIO_VISUEL" }]</pre> <p>Copy Download</p> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Sat, 30 Dec 2023 13:15:27 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Activer Windows
Accédez aux paramètres pour activer Windows

➤ On test la méthode **GET** /resources/{id}

Request URL
http://localhost:8081/resources/1

Server response

Code	Details
200	<p>Response body</p> <pre>{ "resId": 1, "resNom": "resource 1", "type": "MATERIEL_INFO" }</pre> <p>Copy Download</p> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Sat, 30 Dec 2023 13:17:09 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

➤ On test la méthode **POST** /resources

POST /resources

create-resource

Parameters

No parameters

Request body **required**

application/json

```
{
  "resNom": "Resource Name",
  "type": "MATERIEL_INFO"
}
```

Curl

```
curl -X 'POST' \
'http://localhost:8081/resources' \
-H 'accept: application/hal+json' \
-H 'Content-Type: application/json' \
-d '{
  "resNom": "Resource Name",
  "type": "MATERIEL_INFO"
}'
```

Request URL

<http://localhost:8081/resources>

Server response

Code Details

200 Response body

```
{
  "resId": 3,
  "resNom": "Resource Name",
  "type": "MATERIEL_INFO"
}
```

Response headers

```
connection: keep-alive
content-type: application/hal+json
date: Sat, 30 Dec 2023 14:25:30 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Activer Windows
Accédez aux paramètres pour activer Windows.

➤ On test la méthode **PUT** /resources/{id}

PUT /resources/{id}

update-resource

Parameters

Name	Description
id * required	string (path)
1	

Request body **required**

application/json

```
{
  "resNom": "Resource Name 1",
  "type": "MATERIEL_AUDIO_VISUEL"
}
```

Activer Windows

Curl

```
curl -X 'PUT' \
'http://localhost:8081/resources/1' \
-H 'accept: application/hal+json' \
-H 'Content-Type: application/json' \
-d '{
  "resName": "Resource Name 1",
  "type": "MATERIEL_AUDIO_VISUEL"
}'
```

Request URL

```
http://localhost:8081/resources/1
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "resId": 1, "resName": "Resource Name 1", "type": "MATERIEL_AUDIO_VISUEL" }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/hal+json date: Sat, 30 Dec 2023 14:46:47 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Activer Windows
Accédez aux paramètres pour activer Windows.

- On test la méthode **DELETE** /resources/{id}

DELETE /resources/{id}

delete-resource

Parameters

Name	Description
id * required	string (path)

Execute Clear

Responses

Activer Windows

Curl

```
curl -X 'DELETE' \
'http://localhost:8081/resources/3' \
-H 'accept: */*'
```

Request URL

```
http://localhost:8081/resources/3
```

Server response

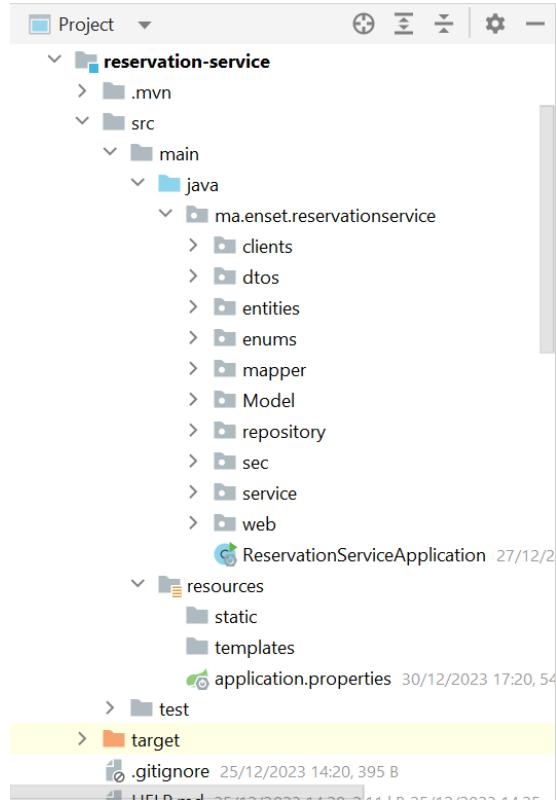
Code	Details
200	<p>Response headers</p> <pre>connection: keep-alive content-length: 0 date: Sat, 30 Dec 2023 14:56:19 GMT keep-alive: timeout=60</pre>

Responses

Code	Description	Links
204	No Content	No links
404	Not Found	No links

5. On développe et on teste le micro-service reservation-service (Entities, DAO, service, DTO, Mapper, RestController, Client Rest Open Feign)

- ✓ On crée un projet spring contenant toutes les classes nécessaires



- ✓ application.properties

```
spring.application.name=reservation-service
server.port=8082
spring.datasource.url=jdbc:h2:mem:reservation-db
spring.h2.console.enabled=true
spring.cloud.discovery.enabled=true
spring.cloud.config.enabled=false
eureka.instance.prefer-ip-address=true
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

✓ ReservationServiceApplication

```
@SpringBootApplication
@EnableFeignClients
public class ReservationServiceApplication {

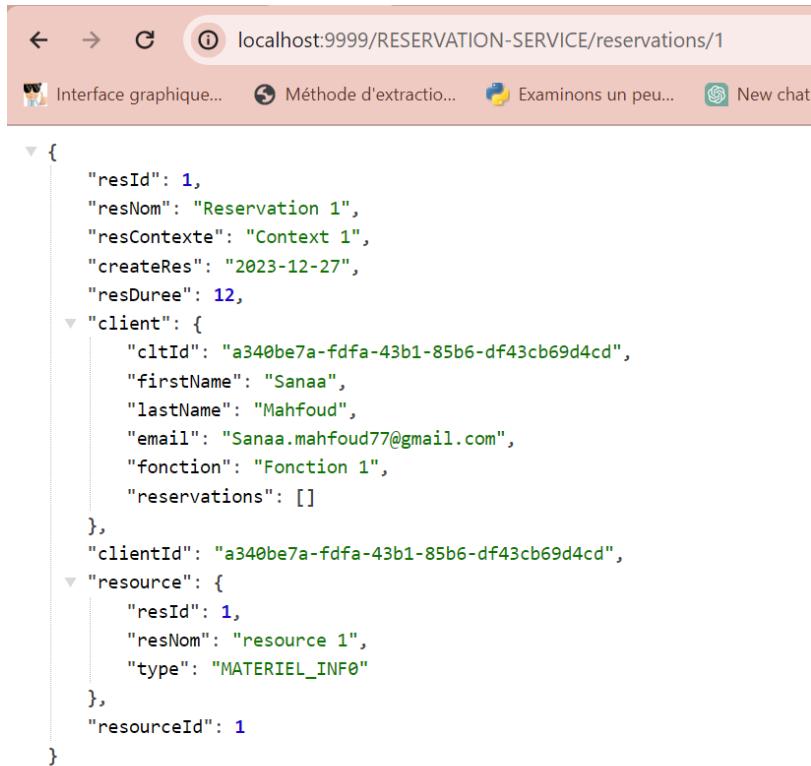
    public static void main(String[] args) {
        SpringApplication.run(ReservationServiceApplication.class, args);
    }

    @Bean
    CommandLineRunner commandLineRunner(reservationRepository reservationRepository,
clientRepository clientRepository){
        return args -> {
            client
client1=client.builder().cltId(UUID.randomUUID().toString()).firstName("Sanaa").lastName("Mahfoud").email("Sanaa.mahfoud77@gmail.com").fonction("Fonction 1").reservations(null).build();
            client
client2=client.builder().cltId(UUID.randomUUID().toString()).firstName("Sakura").lastName("Mahfoud").email("Sakura.mahfoud88@gmail.com").fonction("Fonction 2").build();
            clientRepository.save(client1);
            clientRepository.save(client2);

            List<reservation> reservationList = List.of(
                reservation.builder()
                    //.resId(UUID.randomUUID().toString())
                    .resNom("Reservation 1")
                    .resContexte("Context 1")
                    .createRes(LocalDate.now())
                    .resDuree(12)
                    .clientId(client1.getClId())
                    .client(null)
                    .resourceId(1L)
                    .build(),
                reservation.builder()
                    //.resId(UUID.randomUUID().toString())
                    .resNom("Reservation 2")
                    .resContexte("Context 2")
                    .createRes(LocalDate.now())
                    .resDuree(4)
                    .clientId(client2.getClId())
                    .client(null)
                    .resourceId(2L)
                    .build(),
                reservation.builder()
                    //.resId(UUID.randomUUID().toString())
                    .resNom("Reservation 3")
                    .resContexte("Context 3")
                    .createRes(LocalDate.now())
                    .resDuree(4)
                    .clientId(client1.getClId())
                    .client(null)
                    .resourceId(2L)
                    .build(),
                reservation.builder()
                    //.resId(UUID.randomUUID().toString())
                    .resNom("Reservation 4")
                    .resContexte("Context 4")
                    .createRes(LocalDate.now())
                    .resDuree(4)
                    .clientId(client1.getClId())
                    .client(null)
                    .resourceId(1L)
                    .build()

            );
            reservationRepository.saveAll(reservationList) ;};}}}
```

- ✓ On utilise OpenFeign pour afficher la resource d'une réservation



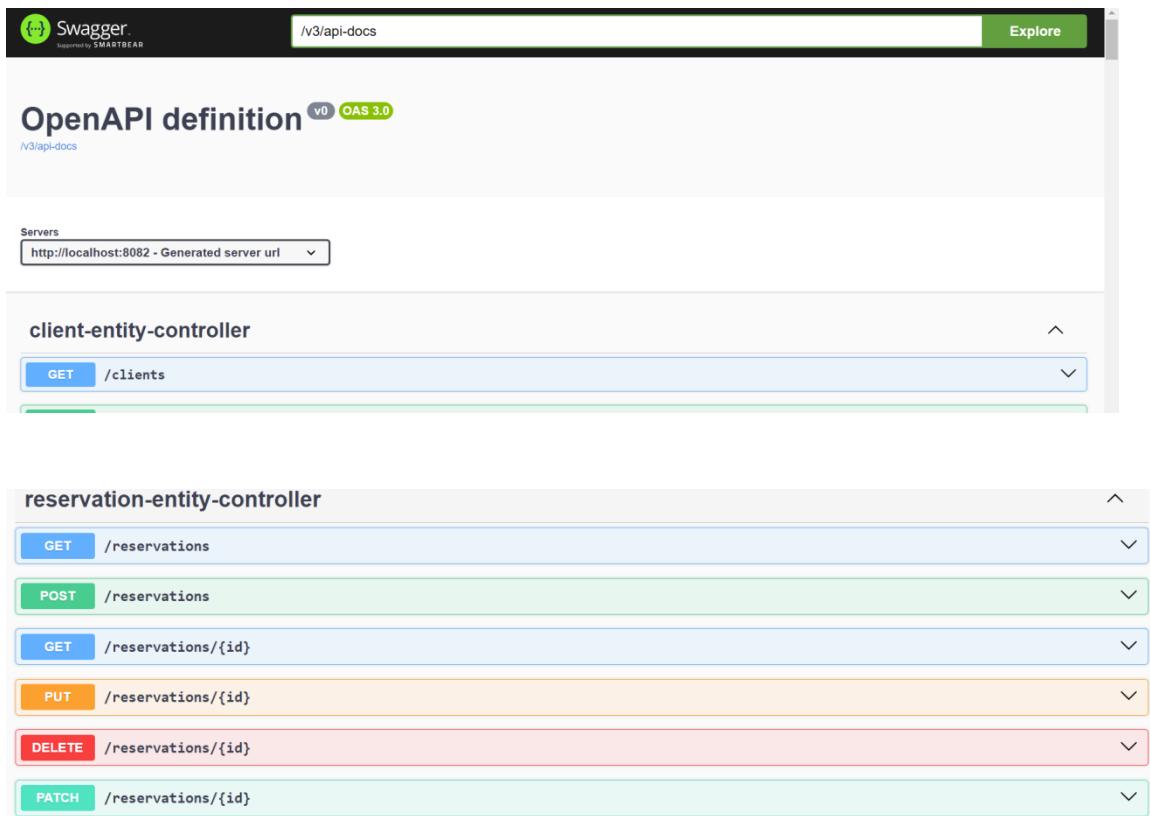
The screenshot shows a browser window with the URL `localhost:9999/RESERVATION-SERVICE/reservations/1`. The page displays a JSON object representing a reservation. The JSON structure is as follows:

```

{
  "resId": 1,
  "resNom": "Reservation 1",
  "resContexte": "Context 1",
  "createRes": "2023-12-27",
  "resDuree": 12,
  "client": {
    "cltId": "a340be7a-fdfa-43b1-85b6-df43cb69d4cd",
    "firstName": "Sanaa",
    "lastName": "Mahfoud",
    "email": "Sanaa.mahfoud77@gmail.com",
    "fonction": "Fonction 1",
    "reservations": []
  },
  "clientId": "a340be7a-fdfa-43b1-85b6-df43cb69d4cd",
  "resource": {
    "resId": 1,
    "resNom": "resource 1",
    "type": "MATERIEL_INFO"
  },
  "resourceId": 1
}

```

- ✓ On génère la documentation des web services Restfull en utilisant la spécification OpenAPIDoc (Swagger).



The screenshot shows the Swagger UI interface for a RESTful service. At the top, it says `/v3/api-docs` and has a **Explore** button.

OpenAPI definition v0 OAS 3.0

`/v3/api-docs`

Servers
`http://localhost:8082 - Generated server url`

client-entity-controller

`GET /clients`

reservation-entity-controller

`GET /reservations`

`POST /reservations`

`GET /reservations/{id}`

`PUT /reservations/{id}`

`DELETE /reservations/{id}`

`PATCH /reservations/{id}`

reservation-rest-controller

- GET** /allClients/{id}
- PUT** /allClients/{id}
- DELETE** /allClients/{id}
- GET** /allClients
- POST** /allClients
- GET** /reservations{idClient}/{id}

✓ On test la méthode **GET** /reservations

Server response

Code	Details
200	Response body
	<pre>[{ "resId": 1, "resNom": "Reservation 1", "resContexte": "Context 1", "createRes": "2023-12-30", "resDuree": 12, "client": null, "clientId": "d7f316d1-dcd9-4dae-b1b5-f55b8ded5c36", "resource": null, "resourceId": 1 }, { "resId": 2, "resNom": "Reservation 2", "resContexte": "Context 2", "createRes": "2023-12-30", "resDuree": 4, "client": null, "clientId": "00684c0f-dfeb-42f4-b28c-e31a5c610e76", "resource": null, "resourceId": 2 }, { "resId": 3, "resNom": "Reservation 3", "resContexte": "Context 3" }]</pre>

Activer Windows  Download 

✓ On test la méthode **POST** /reservations

POST /reservations

create-reservation

Parameters

No parameters

Request body **required**

application/json

```
{
  "resNom": "Reservation Name",
  "resContexte": "Context",
  "resDuree": 6,
  "clientId": "d7f316d1-dcd9-4dae-b1b5-f55b8ded5c36",
  "resourceId": 1
}
```

Request URL
http://localhost:8082/reservations

Server response

Code	Details
200 Undocumented	Response body <pre>{ "resId": 5, "resNom": "Reservation Name", "resContexte": "Context", "createRes": "2023-12-30", "resBuree": 6, "client": null, "clientId": "d7f31ed1-dcd9-4dae-b1b5-f55b8ded5c36", "resource": null, "resourceId": 1 }</pre> <div style="text-align: right;"> Download </div> Response headers <pre>connection: keep-alive content-type: application/hal+json date: Sat, 30 Dec 2023 17:20:49 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

- ✓ On test la méthode **DELETE** /reservations/{id}

DELETE /reservations/{id}

delete-reservation

Parameters

Name	Description
id <small>* required</small> string (path)	5

Execute **Clear**

Request URL
http://localhost:8082/reservations/5

Server response

Code	Details
200 Undocumented	Response headers <pre>connection: keep-alive content-length: 0 date: Sat, 30 Dec 2023 17:30:24 GMT keep-alive: timeout=60</pre>
Responses	
204	No Content
404	Not Found
Links	
204	No links
404	No links

- ✓ On test la méthode **GET** /allClients

Request URL
http://localhost:8082/allClients

Server response

Code	Details
200	Response body <pre>[{ "cltid": "d7f31ed1-dcd9-4dae-b1b5-f55b8ded5c36", "firstName": "Sanaa", "lastName": "Mahfoud", "email": "Sanaa.mahfoud77@gmail.com", "fonction": "Fonction 1", "reservations": [] }, { "cltid": "00684cef-df0b-42f4-b28c-e31a5c610e76", "firstName": "Sakura", "lastName": "Mahfoud", "email": "Sakura.mahfoud8@gmail.com", "fonction": "Fonction 2", "reservations": [] }]</pre> <div style="text-align: right;"> Download </div>

- ✓ On test la méthode **POST** /allClients

```
{
  "firstName": "Ahmed",
  "lastName": "Mahfoud",
  "email": "Ahmed@gmail.com",
  "fonction": "Fonction"
}
```

- ✓ On test la méthode **PUT** /allClients/{id}

```
{
  "firstName": "Sanaa",
  "lastName": "Mahfoud",
  "email": "Sakura@gmail.com",
  "fonction": "Fonction"
}
```

Request URL: <http://localhost:8082/allClients/d7f316d1-dcd9-4dae-b1b5-f55b8ded5c36>

Server response:

Code	Details
200	Response body: <pre>{ "clientId": "d7f316d1-dcd9-4dae-b1b5-f55b8ded5c36", "firstName": "Sanaa", "lastName": "Mahfoud", "email": "Sakura@gmail.com", "fonction": "Fonction", "reservations": [] }</pre> <div style="text-align: right;"> Copy Download </div> Response headers: <pre>connection: keep-alive content-type: application/json date: Sat, 30 Dec 2023 17:43:17 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

- ✓ On test la méthode **DELETE** /allClients/{id}

DELETE /allClients/{id}

Parameters:

Name	Description
Id * required	string (path)

Responses:

```

Curl
curl -X 'DELETE' \
'http://localhost:8082/allClients/4b1cc08-996e-4340-819d-cla3af903993' \
-H 'accept: */*'

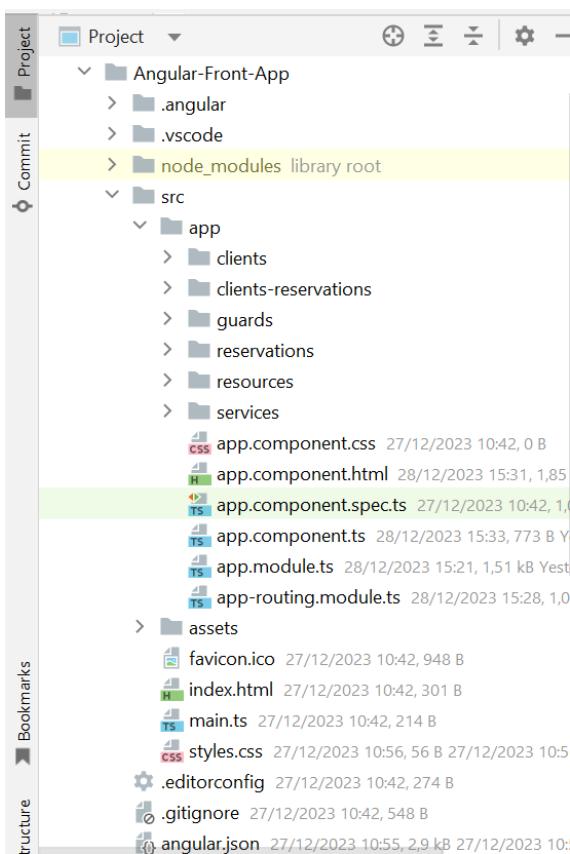
Request URL
http://localhost:8082/allClients/4b1cc08-996e-4340-819d-cla3af903993

Server response
Code Details

200 Response headers
connection: keep-alive
content-length: 0
date: Sat, 30 Dec 2023 17:45:12 GMT
keep-alive: timeout=60

```

6. On développe un simple frontend web Angular pour l'application :



➤ App.components.html

Cette page html représente une barre de navigation (navbar) utilisant Bootstrap, une bibliothèque CSS et JavaScript populaire. La barre de navigation comporte plusieurs éléments, dont un logo "Navbar", un bouton de bascule pour les petits écrans, et une liste d'éléments de navigation tels que "Home", "Resources", "Reservations", et "Clients". Il y a également un élément de menu déroulant à droite avec des options telles que "Login", "Logout", et "Something else here". Le texte affiché dans le menu déroulant est conditionnel et dépend de la variable "profile" dans le code Angular. Si "profile" est défini, il affiche le nom d'utilisateur, sinon il affiche "Not Logged". Enfin, à l'extérieur de la barre de navigation, il y a une balise **<router-outlet>** qui est probablement utilisée dans un cadre Angular pour afficher le contenu de la vue dynamiquement en fonction de l'itinéraire actuel.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" routerLink="/resources">Resources</a>
        </li>
        <li class="nav-item">
          <ngIf="keycloakService.getUserRoles().includes('ADMIN')">
            <a class="nav-link" routerLink="/reservations">Reservations</a>
          </ngIf>
        </li>
        <li class="nav-item" *ngIf="keycloakService.getUserRoles().includes('ADMIN')">
          <a class="nav-link" routerLink="/clients">Clients</a>
        </li>
      </ul>
      <ul class="navbar">
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
            <span *ngIf="profile; else not_logged_in">{{profile.username}}</span>
            <ng-template #not_logged_in>
              Not Logged
            </ng-template>
          </a>
          <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
            <li><a class="dropdown-item" (click)="login()">Login</a></li>
            <li><a class="dropdown-item" (click)="logout()">Logout</a></li>
            <li><hr class="dropdown-divider"></li>
            <li><a class="dropdown-item" href="#">Something else here</a></li>
          </ul>
        </li>
      </ul>
    </div>
  </div>
</nav>
<router-outlet></router-outlet>
```

➤ App.componet.ts

Le fichier **app.component.ts** définit le composant racine de l'application. Ce composant gère l'authentification via Keycloak en utilisant le service **KeycloakService**. Il initialise le profil de l'utilisateur lors de l'initialisation du composant, permet à l'utilisateur de se connecter avec la méthode **login**, et de se déconnecter avec la méthode **logout**. La propriété **profile** stocke les informations du profil utilisateur, et le cycle de vie **ngOnInit** est utilisé pour charger le profil si l'utilisateur est déjà connecté. L'application comporte également une interface utilisateur définie dans le fichier de modèle **app.component.html** et des styles dans le fichier **app.component.css**.

```
import { Component } from '@angular/core';
import {KeycloakProfile} from "keycloak-js";
import {KeycloakService} from "keycloak-angular";

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular-Front-App';
  public profile?: KeycloakProfile;
  constructor(public keycloakService:KeycloakService) {}

  ngOnInit() {
    if(this.keycloakService.isLoggedIn()){
      this.keycloakService.loadUserProfile().then(profile=>{
        this.profile=profile;
      });
    }
  }

  async login() {
    await this.keycloakService.login({
      redirectUri: window.location.origin
    });
  }

  logout() {
    this.keycloakService.logout(window.location.origin)
  }
}
```

➤ App.module.ts

App.module.ts représente le module principal (**AppModule**) d'une application Angular intégrant l'authentification via Keycloak. La fonction **initializeKeycloak** est définie pour configurer et initialiser Keycloak avec des paramètres spécifiques. Ce module utilise le décorateur **@NgModule** pour déclarer les composants, importer les modules nécessaires, et fournir la configuration d'initialisation de Keycloak via **APP_INITIALIZER**. Ainsi, avant le démarrage de l'application, Keycloak est correctement configuré et initialisé, assurant une gestion appropriée de l'authentification tout au long du cycle de vie de l'application.

```
function initializeKeycloak(keycloak: KeycloakService) {
  return () =>
    keycloak.init({
      config: {
        url: 'http://localhost:8080',
        realm: 'sdia-realm',
        clientId: 'sdia-angular-client'
      },
      initOptions: {
        onLoad: 'check-sso',
        silentCheckSsoRedirectUri:
          window.location.origin + '/assets/silent-check-sso.html'
      }
    });
}

@NgModule({
  declarations: [
    AppComponent,
    ResourcesComponent,
    ReservationsComponent,
    ClientsComponent,
    ClientsReservationsComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    KeycloakAngularModule
  ],
  providers: [
    {provide : APP_INITIALIZER, deps : [KeycloakService],useFactory:
    initializeKeycloak, multi : true}
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

➤ App-routing.module.ts

Le module de routage (**AppRoutingModule**) d'une application Angular déclare quatre itinéraires associant des composants spécifiques à des URL. Chaque itinéraire est protégé par le service **AuthGuard**, exigeant une authentification, et détermine l'accès en fonction des rôles utilisateur spécifiés dans les données associées.

```

const routes: Routes = [
  { path : "resources", component : ResourcesComponent, canActivate:[AuthGuard], data : { roles:['USER']} },
  { path : "reservations", component : ReservationsComponent, canActivate:[AuthGuard], data : { roles:['ADMIN','USER']} },
  { path : "clients", component : ClientsComponent, canActivate:[AuthGuard], data : { roles:['ADMIN','USER']} },
  { path : "clients-reservations", component : ClientsReservationsComponent, canActivate:[AuthGuard], data : { roles:['ADMIN','USER']} }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})

```

Resources

➤ Resources.component.html

Le code HTML représente une structure de tableau dans une carte Bootstrap, affichant des informations sur les ressources. Chaque ligne du tableau correspond à une ressource, avec des colonnes pour l'ID, le nom et le type de la ressource. Les données de la table sont générées dynamiquement à partir d'un tableau d'objets "resources" grâce à une boucle *ngFor dans Angular.

```

<div class="p-3">
  <div class="card">
    <div class="card-body">
      <table class="table">
        <thead>
          <tr>
            <th>ID</th><th>Name</th><th>Type</th>
          </tr>
        </thead>
        <tbody>

          </tbody>
          <tr *ngFor="let p of resources">
            <td>{{p.resId}}</td>
            <td>{{p.resNom}}</td>
            <td>{{p.type}}</td>
          </tr>

        </table>
      </div>
    </div>
  </div>

```

➤ Resources.component.ts

Le code Angular dans le fichier **resources.component.ts** définit un composant appelé **ResourcesComponent**. Ce composant utilise le service **HttpClient** pour effectuer une requête GET vers l'URL "http://localhost:9999/RESOURCE-SERVICE/resources" lors de l'initialisation du composant (**ngOnInit**). Les données obtenues sont stockées dans la propriété **resources**, et toute erreur est consignée dans la console. Ce composant est conçu pour afficher des informations sur les ressources dans l'application Angular, et il est lié à un fichier de modèle HTML (**resources.component.html**) qui utilise ces données pour générer une table.

```
@Component({
  selector: 'app-resources',
  templateUrl: './resources.component.html',
  styleUrls: ['./resources.component.css']
})
export class ResourcesComponent {
  resources : any;
  constructor(private http : HttpClient) {
  }
  ngOnInit() {
    this.http.get("http://localhost:9999/RESOURCE-SERVICE/resources")
      .subscribe({
        next : data => {
          this.resources = data;
        },
        error : err => {
          console.log(err);
        }
      })
  }
}
```

Clients

➤ Clients.component.html

Le code HTML représente une structure de tableau dans une carte Bootstrap, affichant des informations sur les clients. Chaque ligne du tableau correspond à un client, avec des colonnes pour l'ID, le prénom, le nom, l'e-mail et la fonction du client. Une colonne supplémentaire contient un bouton "Reservation" qui déclenche la fonction **onButtonClick** en passant l'objet client associé comme paramètre. Cette structure permet d'afficher les détails des clients et d'interagir avec eux, par exemple, pour effectuer des réservations.

```
<div class="p-3">
<div class="card">
  <div class="card-body">
    <table class="table">
      <thead>
        <tr>
          <th>ID</th><th>First Name</th><th>Last
Name</th><th>Email</th><th>Fonction</th><th></th>
        </tr>
      </thead>
      <tbody>

        </tbody>
        <tr *ngFor="let p of clients">
          <td>{{p.cltId}}</td>
          <td>{{p.firstName}}</td>
          <td>{{p.lastName}}</td>
          <td>{{p.email}}</td>
          <td>{{p.fonction}}</td>
          <td><button type="button" class="btn btn-info"
(click)="onButtonClick(p)">Reservation</button></td>
        </tr></table></div></div></div>
```

➤ Clients.component.ts

Le composant Angular **ClientsComponent** est défini pour afficher les informations sur les clients. Il utilise le service **HttpClient** pour effectuer une requête GET vers "http://localhost:9999/RESERVATION-SERVICE/allClients" lors de son initialisation (**ngOnInit**). Les données obtenues sont stockées dans la propriété **clients**. Le composant inclut également une fonction **onButtonClick(p)** qui affiche les informations du client sélectionné dans la console, stocke ces informations dans la propriété **informationsClient**, puis navigue vers une nouvelle route (**/clients-reservations**) en passant les données du client sous forme de chaîne JSON dans l'URL. Ce mécanisme facilite la transition vers une page dédiée aux réservations pour le client sélectionné.

```
@Component({
  selector: 'app-clients',
  templateUrl: './clients.component.html',
  styleUrls: ['./clients.component.css']
})
export class ClientsComponent {
  clients : any;
  informationsClient : any;
  constructor(private http : HttpClient, private router : Router) {
  }
  ngOnInit() {
    this.http.get("http://localhost:9999/RESERVATION-SERVICE/allClients")
      .subscribe({
        next : data => {
          this.clients = data;

        },
        error : err => {
          console.log(err);
        }
      })
  }
  onButtonClick(p: any) {
    console.log('Informations de la ligne:', p);
    this.informationsClient = p;
    this.router.navigate(['/clients-reservations', { data:
      JSON.stringify(this.informationsClient) }]);
  }
}
```

➤ Clients-reservations.component.html

```
<div class="p-3">
  <div class="card">
    <div class="card-body">
      <table class="table">
        <thead>
          <tr>
            <th>ID</th><th>Name</th><th>Contexte</th><th>Creation Date</th><th>Duration</th><th>Client Id</th><th>Resource Id</th>
          </tr>
        </thead>
        <tbody>
        </tbody>
      <tr *ngFor="let p of clientsReservations">
        <td>{{p.resId}}</td>
        <td>{{p.resNom}}</td>
        <td>{{p.resContexte}}</td>
        <td>{{p.createRes}}</td>
        <td>{{p.resDuree}}</td>
        <td>{{p.clientId}}</td>
        <td>{{p.resourceId}}</td>
      </tr>
    </table>
  </div>
</div>
</div>
```

➤ Clients-reservations.component.ts

Le composant Angular **ClientsReservationsComponent** récupère les paramètres de route à l'aide du service **ActivatedRoute**. Il vérifie si le paramètre 'data' existe dans les paramètres, le parse en objets client, puis effectue une requête HTTP pour obtenir les réservations associées à ce client à partir du service "http://localhost:9999/RESERVATION-SERVICE/reservations/idClient/". Les informations des réservations sont ensuite stockées dans la propriété **clientsReservations**. Ce composant est conçu pour afficher les réservations spécifiques à un client après avoir cliqué sur le bouton "Reservation" dans la liste des clients.

```
@Component({
  selector: 'app-clients-reservations',
  templateUrl: './clients-reservations.component.html',
  styleUrls: ['./clients-reservations.component.css']
})
export class ClientsReservationsComponent {
  clientInformations: any;
  clientsReservations: any;

  constructor(private http : HttpClient, private route: ActivatedRoute) {}

  ngOnInit() {
    // Récupérer les paramètres de route
    this.route.params.subscribe(params => {
      // Vérifier si le paramètre 'data' existe dans les paramètres
      if (params['data']) {
        this.clientInformations = JSON.parse(params['data']);
        console.log(this.clientInformations['cltId'])
        this.http.get("http://localhost:9999/RESERVATION-SERVICE/reservations/idClient/"+this.clientInformations['cltId'])
      }
    })
  }
}
```

➤ Reservations.component.ts

```
<div class="p-3">
  <div class="card">
    <div class="card-body">
      <table class="table">
        <thead>
          <tr>
            <th>ID</th><th>Name</th><th>Contexte</th><th>Creation Date</th><th>Duration</th><th>Client Id</th><th>Resource Id</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let p of reservations">
            <td>{{p.resId}}</td>
            <td>{{p.resNom}}</td>
            <td>{{p.resContexte}}</td>
            <td>{{p.createRes}}</td>
            <td>{{p.resDuree}}</td>
            <td>{{p.clientId}}</td>
            <td>{{p.resourceId}}</td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</div>
```

➤ Reservations.component.ts

Le composant contient une propriété **reservations** de type **any** et utilise le service HTTP pour récupérer des données depuis l'URL "http://localhost:9999/RESERVATION-SERVICE/reservations" lors de l'initialisation du composant via la méthode **ngOnInit()**. Les données récupérées sont assignées à la propriété **reservations**. En cas d'erreur, un message d'erreur est affiché dans la console.

```
@Component({
  selector: 'app-reservations',
  templateUrl: './reservations.component.html',
  styleUrls: ['./reservations.component.css']
})
export class ReservationsComponent {
  reservations : any;
  constructor(private http : HttpClient) {
  }
  ngOnInit() {
    this.http.get("http://localhost:9999/RESERVATION-SERVICE/reservations")
      .subscribe({
        next : data => {
          this.reservations = data;
        },
        error : err => {
          console.log(err);
        }
      })
  }
}
```

➤ Auth.guard.ts

Ce service est annoté avec `@Injectable` et est fourni au niveau racine de l'application (`providedIn: 'root'`). Il étend `KeycloakAuthGuard` du module `keycloak-angular` et implémente la méthode `isAccessAllowed` pour contrôler l'accès aux routes en fonction des rôles utilisateur. Le service utilise le service `KeycloakService` pour gérer l'authentification avec Keycloak. Si l'utilisateur n'est pas authentifié, il est redirigé vers la page de connexion. Les rôles requis sont extraits des données de la route, et l'accès est autorisé si l'utilisateur possède tous les rôles requis.

```
import { Injectable } from '@angular/core';
import {
  ActivatedRouteSnapshot,
  Router,
  RouterStateSnapshot
} from '@angular/router';
import { KeycloakAuthGuard, KeycloakService } from 'keycloak-angular';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard extends KeycloakAuthGuard {
  constructor(
    protected override readonly router: Router,
    protected readonly keycloak: KeycloakService
  ) {
    super(router, keycloak);
  }

  public async isAccessAllowed(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ) {
    // Force the user to log in if currently unauthenticated.
    if (!this.authenticated) {
      await this.keycloak.login({
        redirectUri: window.location.origin
      });
    }

    // Get the roles required from the route.
    const requiredRoles = route.data['roles'];

    // Allow the user to proceed if no additional roles are required to
    // access the route.
    if (!(requiredRoles instanceof Array) || requiredRoles.length === 0) {
      return true;
    }

    // Allow the user to proceed if all the required roles are present.
    return requiredRoles.every((role) => this.roles.includes(role));
  }
}
```

➤ On affiche l'interafce Angular

The screenshot shows a web browser window with the URL `localhost:4200` in the address bar. The page title is "Navbar". The navigation bar at the top includes links for "Home", "Resources", "Reservations", and "Clients". On the right side of the header, there is a dropdown menu showing "Not Logged". The main content area is currently empty, displaying a light gray background.

➤ On affiche les Resources :

The screenshot shows a web browser window with the URL `localhost:4200/resources` in the address bar. The page title is "Navbar". The navigation bar at the top includes links for "Home", "Resources", "Reservations", and "Clients". On the right side of the header, there is a dropdown menu showing "Username". The main content area displays a table with two rows of data:

ID	Name	Type
1	resource 1	MATERIEL_INFO
2	resource 2	MATERIEL_AUDIO_VUSUEL

➤ On affiche les clients

The screenshot shows a web browser window with the URL `localhost:4200/clients` in the address bar. The page title is "Navbar". The navigation bar at the top includes links for "Home", "Resources", "Reservations", and "Clients". On the right side of the header, there is a dropdown menu showing "Username". The main content area displays a table with two rows of data:

ID	First Name	Last Name	Email	Fonction	Action
a340be7a-fdfa-43b1-85b6-df43cb69d4cd	Sanaa	Mahfoud	Sanaa.mahfoud77@gmail.com	Fonction 1	Reservation
c27f17a0-eff7-4d68-88bb-fe508af71d71	Sakura	Mahfoud	Sakura.mahfoud88@gmail.com	Fonction 2	Reservation

- On clique sur Reservation pour afficher les réservations de client dans la ligne sélectionnée :

The screenshot shows a web browser window with the URL `localhost:4200/clients-reservations;data=%7B"clientId":"a340be7a-fdfa-43b1-85b6-df43cb69d4cd","firstName":"Sanaa","lastName":"Mahfoud",...`. The page displays a table of reservations for a client with ID `a340be7a-fdfa-43b1-85b6-df43cb69d4cd`. The table has columns: ID, Name, Context, Creation Date, Duration, Client Id, and Resource Id. There are three rows of data.

ID	Name	Context	Creation Date	Duration	Client Id	Resource Id
1	Reservation 1	Context 1	2023-12-27	12	a340be7a-fdfa-43b1-85b6-df43cb69d4cd	1
3	Reservation 3	Context 3	2023-12-27	4	a340be7a-fdfa-43b1-85b6-df43cb69d4cd	2
4	Reservation 4	Context 4	2023-12-27	4	a340be7a-fdfa-43b1-85b6-df43cb69d4cd	1

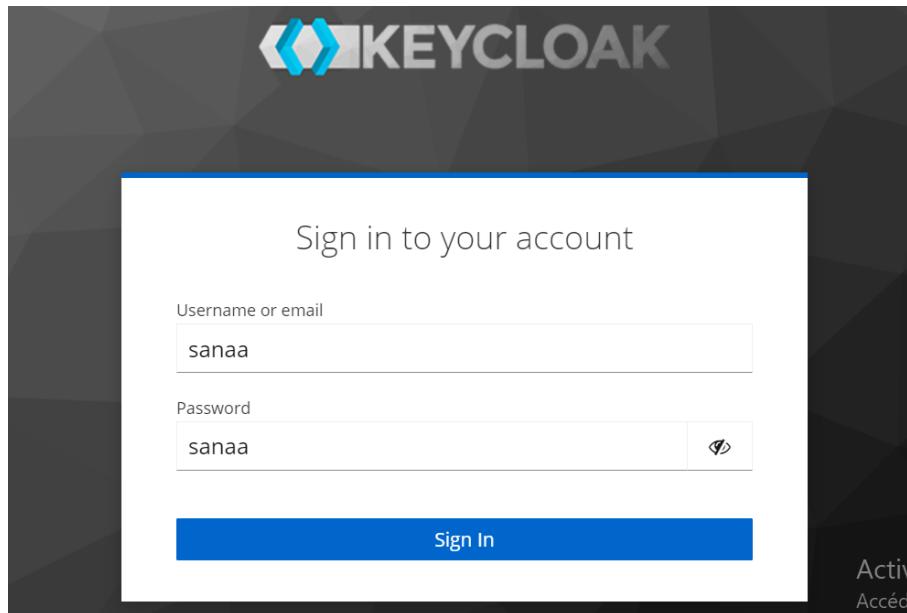
- On affiche toutes les réservations existe :

The screenshot shows a web browser window with the URL `localhost:4200/reservations`. The page displays a table of all existing reservations. The table has columns: ID, Name, Context, Creation Date, Duration, Client Id, and Resource Id. There are four rows of data.

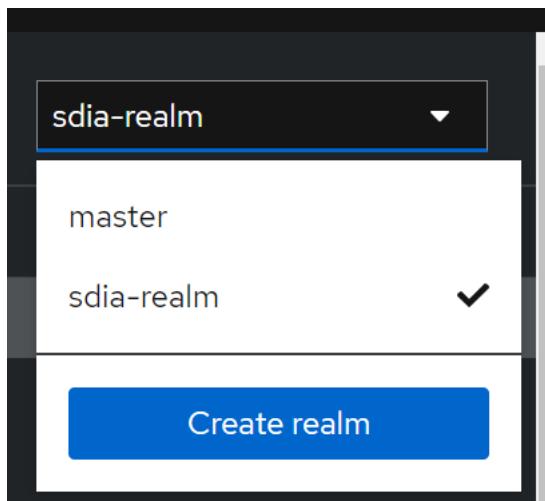
ID	Name	Context	Creation Date	Duration	Client Id	Resource Id
1	Reservation 1	Context 1	2023-12-27	12	a340be7a-fdfa-43b1-85b6-df43cb69d4cd	1
2	Reservation 2	Context 2	2023-12-27	4	c27f17a0-eff7-4d68-88bb-fe508af71d71	2
3	Reservation 3	Context 3	2023-12-27	4	a340be7a-fdfa-43b1-85b6-df43cb69d4cd	2
4	Reservation 4	Context 4	2023-12-27	4	a340be7a-fdfa-43b1-85b6-df43cb69d4cd	1

7. On sécurise l'application avec une authentification Keycloak :

- On crée un admin avec **username** sanaa et **password** sanaa :



- On crée un Realm sdia-realm



- On crée un client `sdia-angular-client`

Clients > Client details

sdia-angular-client OpenID Connect

Enabled Action

General settings

Client ID * `sdia-angular-client`

Name

Description

Jump to section

- General settings
- Access settings
- Capability config

- On crée les roles USER et ADMIN

Role name	Composite	Description
ADMIN	False	–
USER	False	–
default-roles-sdia-realm	True	`\${role_default-roles}`
offline_access	False	`\${role_offline-access}`
uma_authorization	False	`\${role_uma_authorization}`

- On crée les utilisateurs user1 avec password =1234 et user2 avec password=1234

Username	Email	Last name	First name	Status
user1	user1@gmail.com	mahfoud	sanaa	–
user2	user2@gmail.com	mahfoud	sakura	–

- On affect le role USER à user1 et le role USER & ADMIN à user2

The screenshot shows the Keycloak interface for managing users. The left sidebar is titled 'sdia-realm' and includes 'Manage', 'Clients', 'Client scopes', 'Realm roles', 'Users' (which is selected), 'Groups', 'Sessions', and 'Events'. The main panel is titled 'User details' for 'user1'. It has tabs for 'Details', 'Attributes', 'Credentials', 'Role mapping' (which is selected), 'Groups', 'Consents', 'Identity provider links', and 'Sessions'. Under 'Role mapping', there is a search bar, a checkbox for 'Hide inherited roles' (which is checked), and a 'Assign role' button. Below these are two rows of role mappings:

Name	Inherited	Description
USER	False	-
default-roles-sdia-realm	False	\${role_default-roles}

This screenshot shows the Keycloak interface for managing users. The left sidebar is titled 'sdia-realm' and includes 'Manage', 'Clients', 'Client scopes', 'Realm roles', 'Users' (selected), 'Groups', 'Sessions', and 'Events'. The main panel is titled 'User details' for 'user2'. It has tabs for 'Details', 'Attributes', 'Credentials', 'Role mapping' (selected), 'Groups', 'Consents', 'Identity provider links', and 'Sessions'. Under 'Role mapping', there is a search bar, a checkbox for 'Hide inherited roles' (checked), and a 'Unassign' button. Below these are three rows of role mappings:

Name	Inherited	Description
USER	False	-
ADMIN	False	-
default-roles-sdia-realm	False	\${role_default-roles}

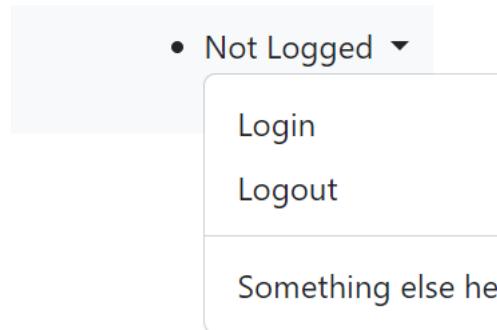
- On ajoute les roles à JWT

The screenshot shows the Keycloak interface for managing clients. The left sidebar is titled 'sdia-realm' and includes 'Manage', 'Clients' (selected), 'Client scopes', 'Realm roles', 'Users', 'Groups', and 'Sessions'. The main panel is titled 'Clients > Client details > Dedicated scopes' for 'sdia-angular-client'. It includes a 'Mappers' tab (selected) and a 'Scope' tab. There is a search bar for mappers and a 'Add mapper' button. Below is a table of mappers:

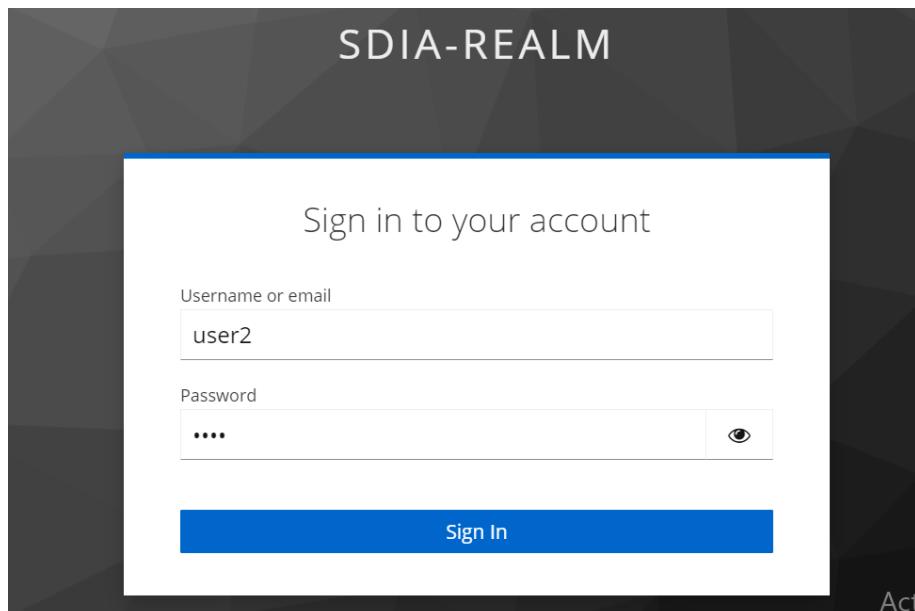
Name	Category	Type	Priority
realm roles	Token mapper	User Realm Role	40

- Comme j'ai déjà mentionné, j'ai ajouté les informations nécessaires de Keycloak dans l'application Angular, puis on affiche les résultats :

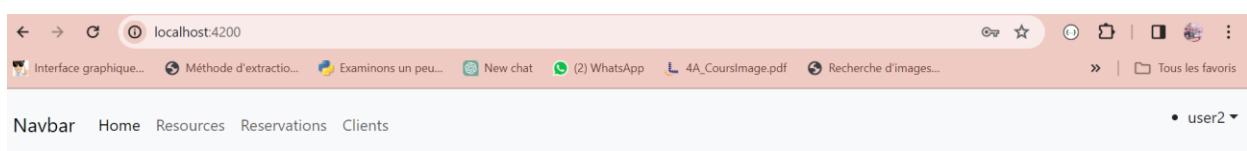
- On fait Logout puis Login :



- On connect avec le user2 qui est ADMIN et USER



- Voilà l'interface affiché :



- Tant que user2 est ADMIN & USER il peut afficher les ressources et les réservations et les clients et peut aussi afficher les réservations de chaque client :

ID	Name	Type
1	resource 1	MATERIEL_INFO
2	resource 2	MATERIEL_AUDIO_VSUEL

ID	Name	Contexte	Creation Date	Duration	Client Id	Resource Id
1	Reservation 1	Context 1	2023-12-28	12	d0f25f14-7687-4422-ba6b-3a692d4c4e20	1
2	Reservation 2	Context 2	2023-12-28	4	2472cf07-b50d-4c93-b573-a542cae0506b	2
3	Reservation 3	Context 3	2023-12-28	4	d0f25f14-7687-4422-ba6b-3a692d4c4e20	2
4	Reservation 4	Context 4	2023-12-28	4	d0f25f14-7687-4422-ba6b-3a692d4c4e20	1

ID	First Name	Last Name	Email	Fonction
d0f25f14-7687-4422-ba6b-3a692d4c4e20	Sanaa	Mahfoud	Sanaa.mahfoud77@gmail.com	Fonction 1
2472cf07-b50d-4c93-b573-a542cae0506b	Sakura	Mahfoud	Sakura.mahfoud88@gmail.com	Fonction 2

. The page displays a table with three rows of reservation data, corresponding to the clients listed above."/>

ID	Name	Contexte	Creation Date	Duration	Client Id	Resource Id
1	Reservation 1	Context 1	2023-12-28	12	d0f25f14-7687-4422-ba6b-3a692d4c4e20	1
3	Reservation 3	Context 3	2023-12-28	4	d0f25f14-7687-4422-ba6b-3a692d4c4e20	2
4	Reservation 4	Context 4	2023-12-28	4	d0f25f14-7687-4422-ba6b-3a692d4c4e20	1

- Tant que user 1 est USER alors il peut afficher seulement les ressources.

ID	Name	Type
1	resource 1	MATERIEL_INFO
2	resource 2	MATERIEL_AUDIO_VUSUEL

8. Déploiement de l'application avec Docker et Docker compose

- Le fichier docker compose :

Ce fichier de configuration Docker Compose définit plusieurs services et conteneurs pour une application distribuée. Il crée des services pour une base de données MariaDB, des services PHPMyAdmin pour gérer ces bases de données, des services Spring Boot pour la gestion des ressources et des réservations, ainsi que des services Keycloak pour la gestion de l'authentification. De plus, il configure un service Angular pour l'interface utilisateur et utilise des volumes Docker pour stocker les données persistantes des bases de données. Les services sont orchestrés à l'aide d'un service de découverte Eureka, et le fichier inclut également des services PostgreSQL, pgAdmin4 et Keycloak, chacun avec ses dépendances et configurations spécifiques.

- Le fichier dockerfile :

Ce Dockerfile crée une image contenant une application Java, en copiant le fichier JAR généré dans le conteneur et en définissant l'exécution de ce JAR comme point d'entrée.

```
FROM openjdk:17-oracle
VOLUME /tmp
COPY target/*.jar app.jar
ENTRYPOINT ["java","-jar", "app.jar"]
```

- On crée le fichier jar de chaque application avec la commande suivante :

mvn clean package –DskipTests

- On démarre les conteneurs avec la commande : docker-compose up -d –build

```
Terminal: Local × Local (2) × Local (3) × Local (4) × Local (5) × Local (6) × Local (7) ×
  ✓ Container enset-gateway-service      Started
  ✓ Container postgres-service          Healthy
  ✓ Container mysql_db                 Healthy
  ✓ Container enset-discovery-service   Started
  ✓ Container enset-reservations-service Started
  ✓ Container enset-resources-service   Started
  ✓ Container phpmyadmin               Started
  ✓ Container examen-keycloak-1        Started
```

```
PS C:\Users\triangle info\Desktop\Sanaa\Master_SDIA\S3\SystemDistr2\Examen> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
ea31a560fa0d quay.io/keycloak/keycloak:latest "/opt/keycloak/bin/k..." 4 minutes ago Up 3 minutes 0.0.0.0:8080->8080/tcp, 8443/tcp
examen-keycloak-1
cfe78807b564 dpage/pgadmin4 "/entrypoint.sh" 4 minutes ago Up 4 minutes 443/tcp, 0.0.0.0:8888->80/tcp
pgadmin4
74a7f003265a postgres "docker-entrypoint.s..." 4 minutes ago Up 4 minutes (healthy) 0.0.0.0:5432->5432/tcp
postgres-service
44ea8fec00f2 examen-enset-discovery-service "java -jar app.jar" 4 minutes ago Up 4 minutes (healthy) 0.0.0.0:8761->8761/tcp
enset-discovery-service
4684e76129f7 mariadb:10.6 "docker-entrypoint.s..." 4 minutes ago Up 4 minutes (healthy) 0.0.0.0:3306->3306/tcp
mysql_db
```

- Voila l'interface de pgadmin :



Register - Server

General Connection Parameters SSH Tunnel Advanced

Name	postgres
Server group	Servers
Background	X
Foreground	X
Connect now?	<input checked="" type="checkbox"/>
Shared?	<input type="checkbox"/>
Shared Username	
Comments	

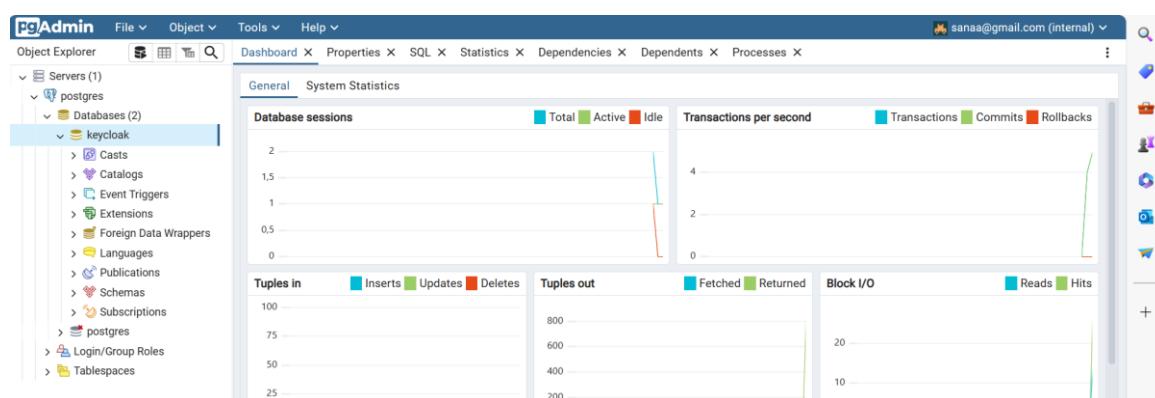
Buttons:

Register - Server

General Connection Parameters SSH Tunnel Advanced

Host name/address	postgres-service
Port	5432
Maintenance database	keycloak
Username	keycloak
Kerberos authentication?	<input type="checkbox"/>
Password
Save password?	<input type="checkbox"/>
Role	
Service	

Buttons:



- On affiche phpmyadmin

Bienvenue dans phpMyAdmin

Langue - Language

Français - French

Connexion

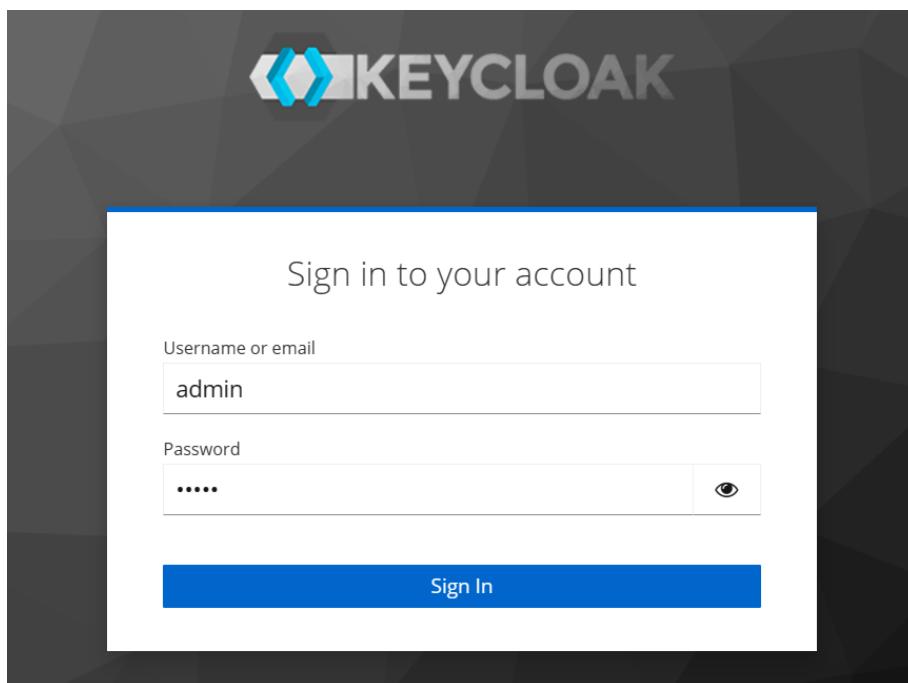
Serveur :

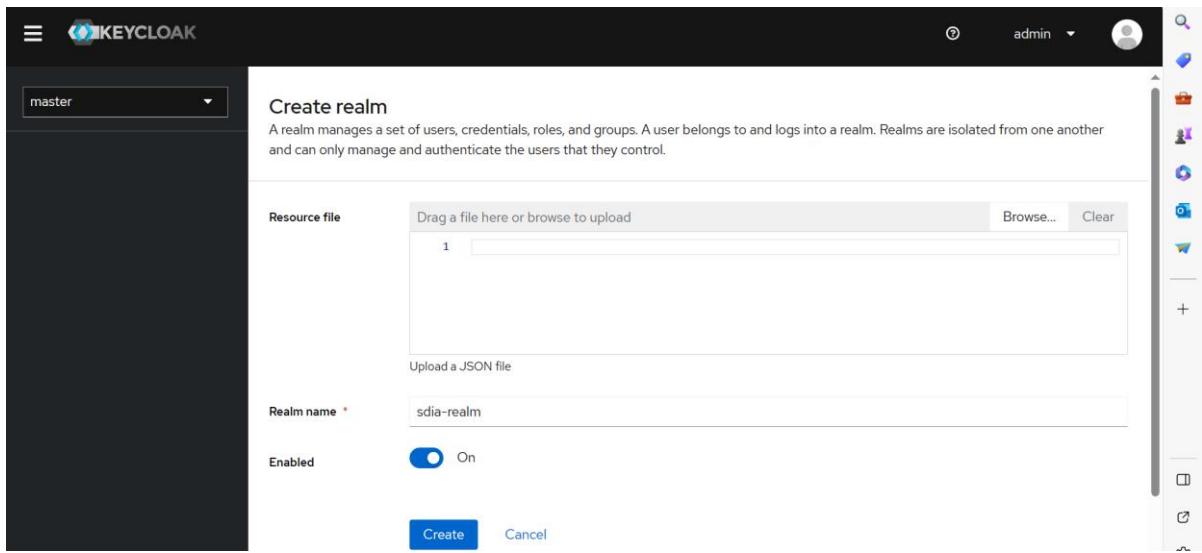
Utilisateur : mahfoud

Mot de passe : ****

Exécuter

- On affiche Keycloak et on crée un realm **sdia-realm**, un client **sdia-angular-client** puis on crée les utilisateurs et les roles et on affecte les roles à les utilisateurs :





III. Conclusion

Le projet visant à créer une application de gestion de réservations basée sur une architecture micro-service a été mené avec succès. L'architecture technique comprend plusieurs micro-services fonctionnels, chacun remplaçant un rôle spécifique dans le système. Le service Gateway, le service Discovery ont été mis en place avec Spring Cloud Gateway et Eureka Server respectivement.

Les micro-services principaux, tels que resources-service et reservation-service, ont été développés et testés de manière approfondie, couvrant l'ensemble du cycle de vie du développement, de la création des entités à la création des DTO, Mappers, services, et enfin les RestControllers. La documentation des web services Restful a été générée en utilisant la spécification OpenAPI Doc (Swagger).

Un frontend web simple basé sur le framework Angular a été développé pour l'application, fournissant une interface utilisateur conviviale pour la gestion des ressources et des réservations. La sécurité de l'application repose sur OAuth2 et OIDC avec Keycloak comme fournisseur d'identité, assurant un environnement sécurisé.

Enfin, le déploiement de l'ensemble de l'application a été simplifié grâce à l'utilisation de Docker et Docker Compose, facilitant la gestion des conteneurs et des dépendances.