ECOLE NORMALE SUPÉRIURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Master Distributed Systemes and Artificial Intelligence

Distributed Systemes

Rapport Examen Final

Réalisé par :
Soufiane MOUHTARAM

Professor :
Pr. YOUSSFI Moahemd

1er janvier 2024

TABLE DE CONTENTES

INTRODUCTION

Ce rapport décrit la conception et la mise en œuvre d'une application de gestion de ressources et de réservations basée sur les microservices. L'application est sécurisée avec Keycloak et déployée avec Docker.

FIGURE 1 – Frameworks

Les outils de development

FIGURE 2 – Tools

ARCHITECTURE

Architecture

L'application est composée de deux microservices :

Le microservice ressource est responsable de la gestion des ressources de l'application. Il fournit une API qui permet aux autres microservices d'accéder aux ressources, telles que les données, les fichiers et les services externes.

Le microservice reservation est responsable de la gestion des réservations de l'application. Il fournit une API qui permet aux utilisateurs de réserver des ressources.

Sécurité

L'application est sécurisée avec Keycloak. Keycloak est une solution d'authentification et d'autorisation open source. Elle permet de sécuriser les applications microservices en garantissant que seuls les utilisateurs autorisés peuvent accéder aux ressources.

Les utilisateurs doivent s'authentifier auprès de Keycloak avant de pouvoir accéder à l'application. Keycloak fournit une variété de méthodes d'authentification, telles que l'authentification par mot de passe, l'authentification par certificat et l'authentification sociale.

Déploiement

L'application est déployée avec Docker. Docker est une plateforme de conteneurisation qui permet de déployer des applications microservices de manière simple et efficace. Les microservices sont empaquetés dans des conteneurs, qui sont des images légères et portables.

Les conteneurs sont déployés sur un cluster Kubernetes. Kubernetes est un système de gestion de conteneurs open source qui permet de gérer et de mettre à l'échelle des applications composées de microservices.

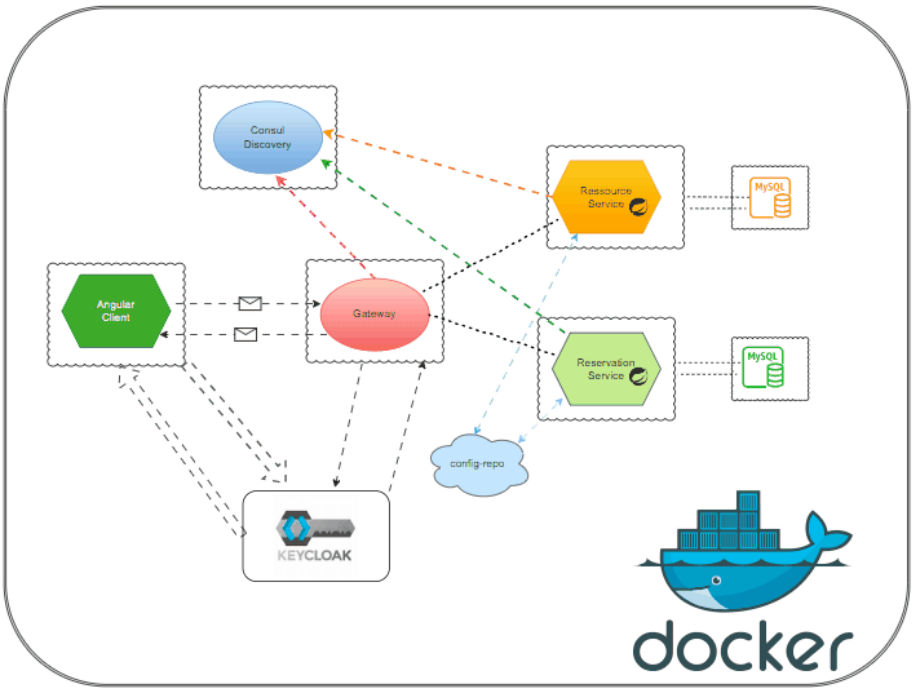

FIGURE 3 – Architecture de projet

ConsulDiscovery Consul Discovery :

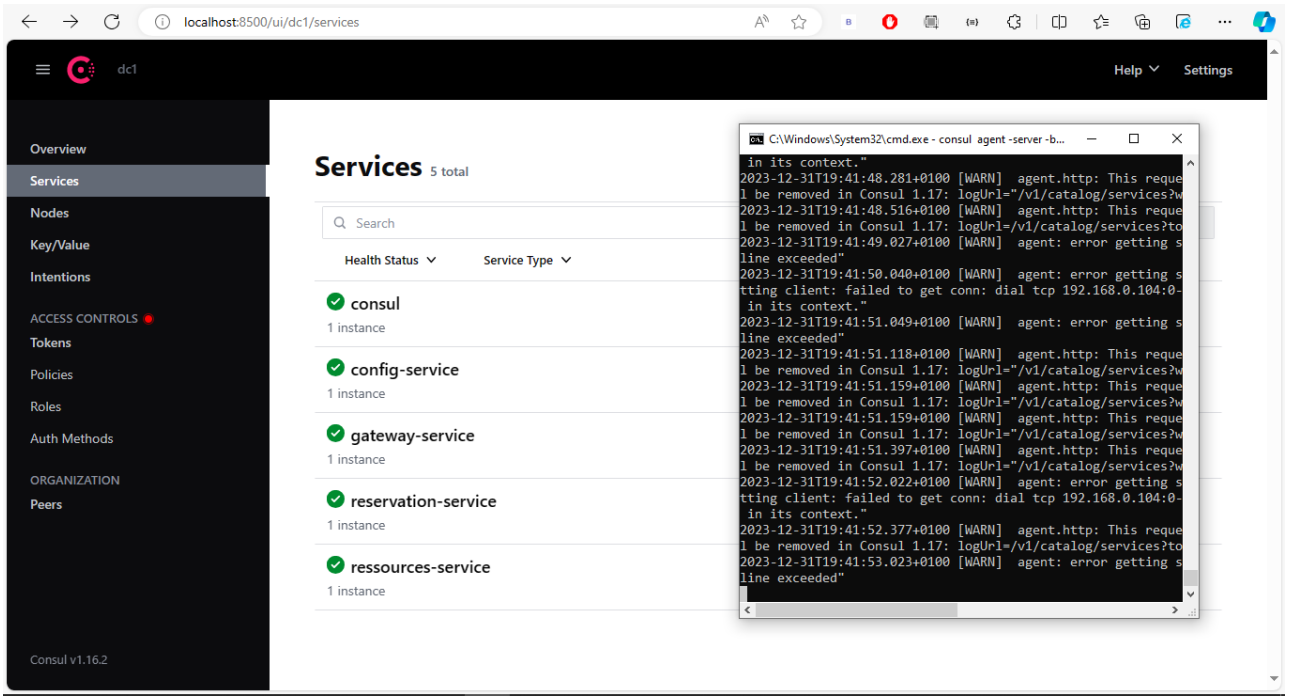

FIGURE 4 – Consul Discovery

KEYCLOAK

Keycloak Keycloak interface :

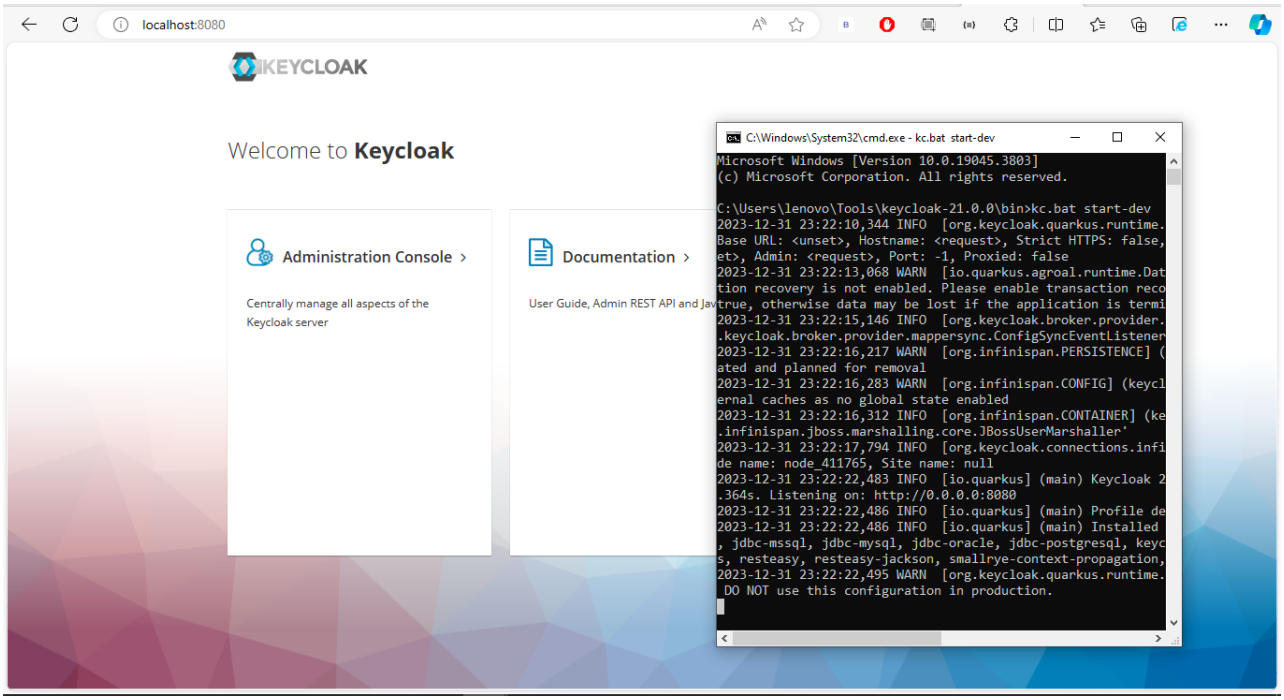

FIGURE 5 – Keycloak

SWAGGER

SwaggerDocumentation : Microservice reservation-service :

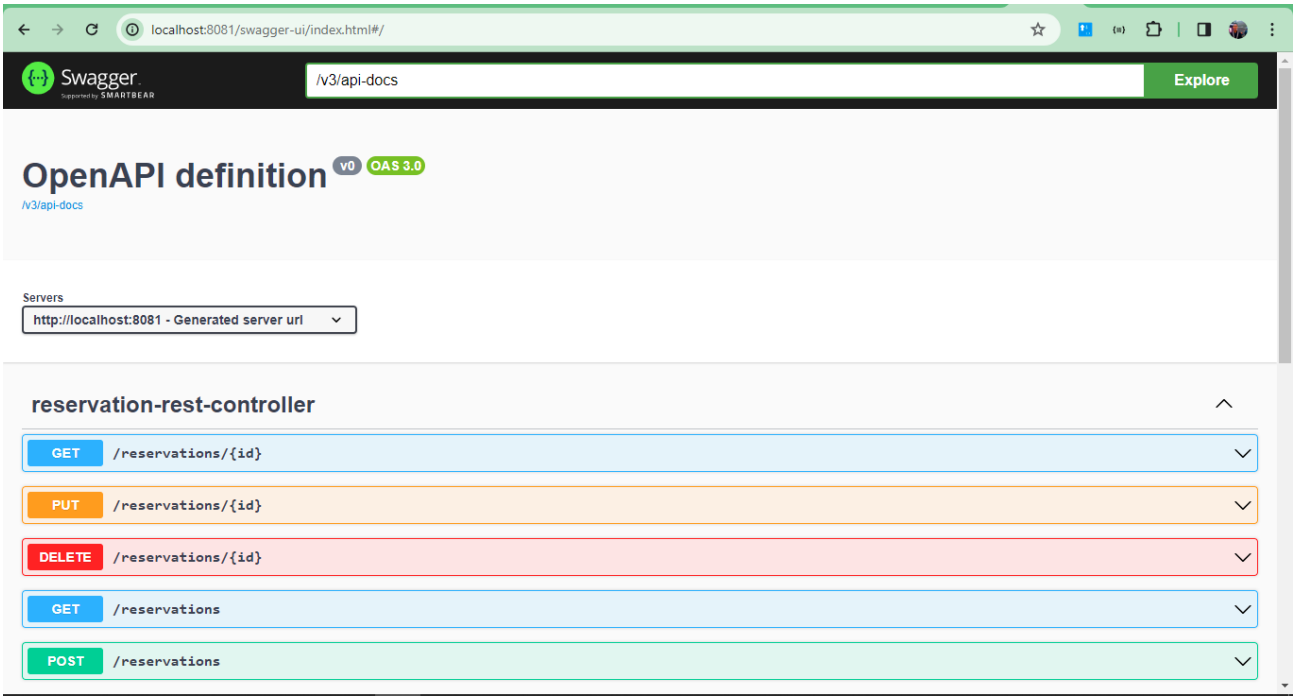

FIGURE 6 – Swagger microservice reservation-service

SwaggerDocumentation : Microservice ressource-service :

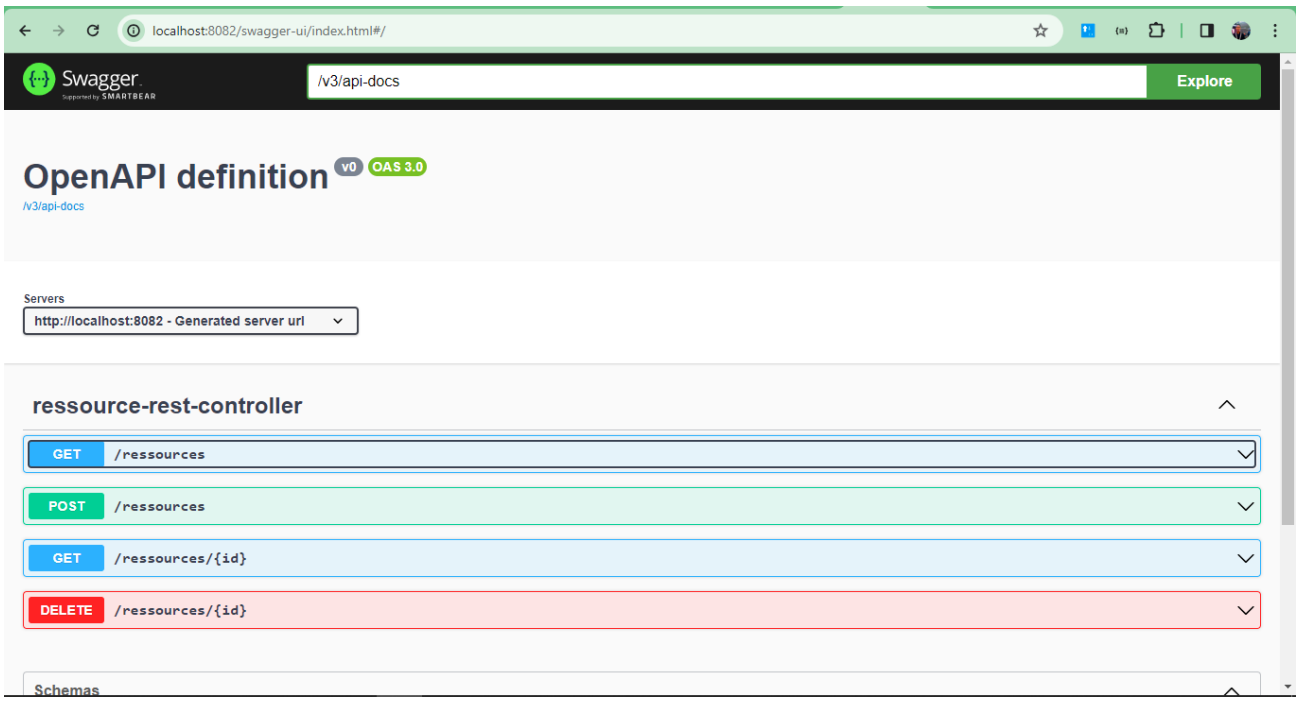

FIGURE 7 – Swagger microservice ressource-service

LES TESTS A TRAVERS POSTMAN

Envoyer un requette GET a `'http://localhost:9999/reservation-service/reservations'` avec le acces-token pour avoir la listes des reservations.

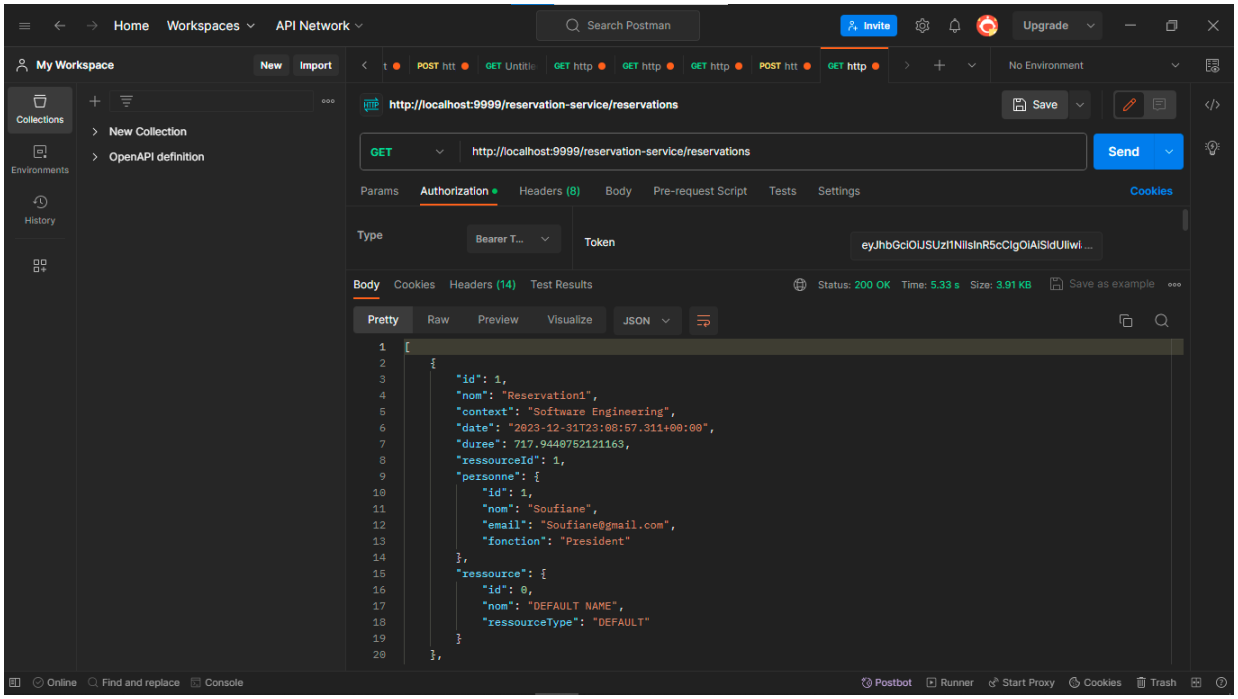

FIGURE 8 – ressource-service

SCREEN D'ÉCRANS DE VS CODE

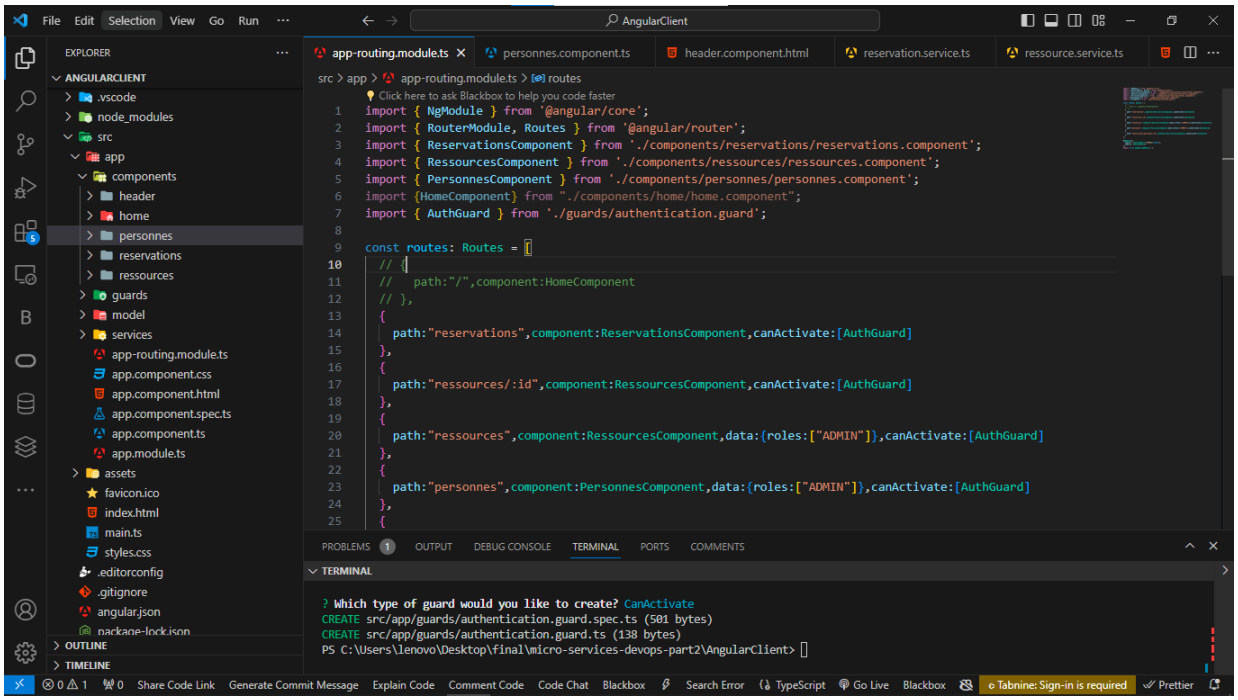

FIGURE 9 – l'architecture de project Angular

DIGRAMME DE CLASSE DE PROJET

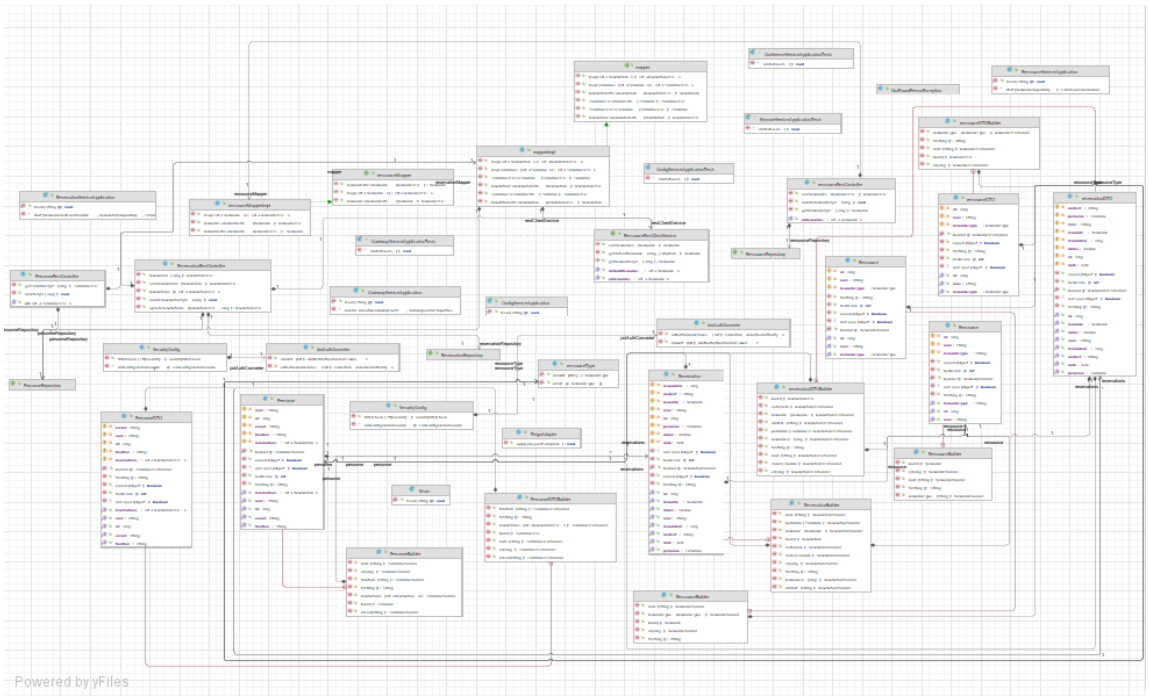

FIGURE 10 – Diagramme de classe

STRUCTURE DE MICRO-SERVICES

La structure de chaque micro-service

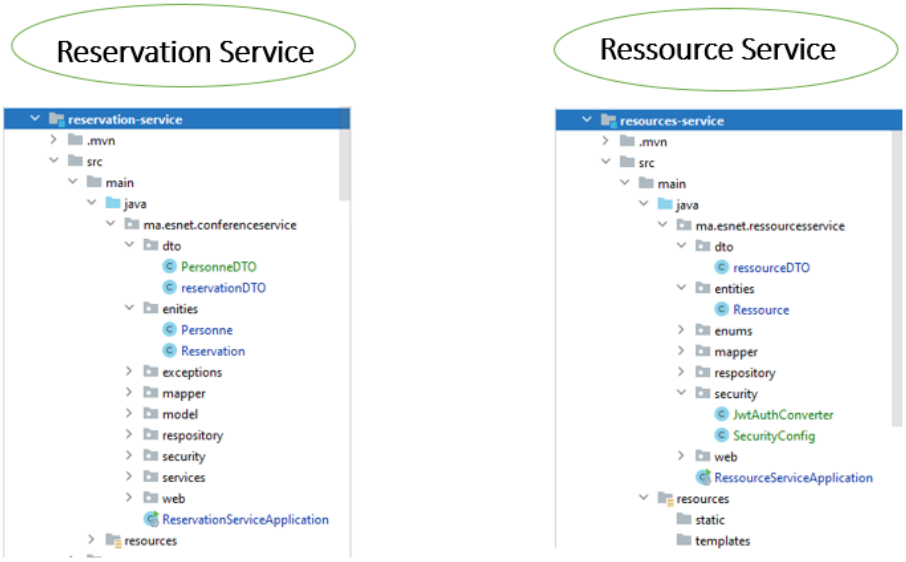

FIGURE 11 – Structure complete

Les entities et DTOs de Ressource Service

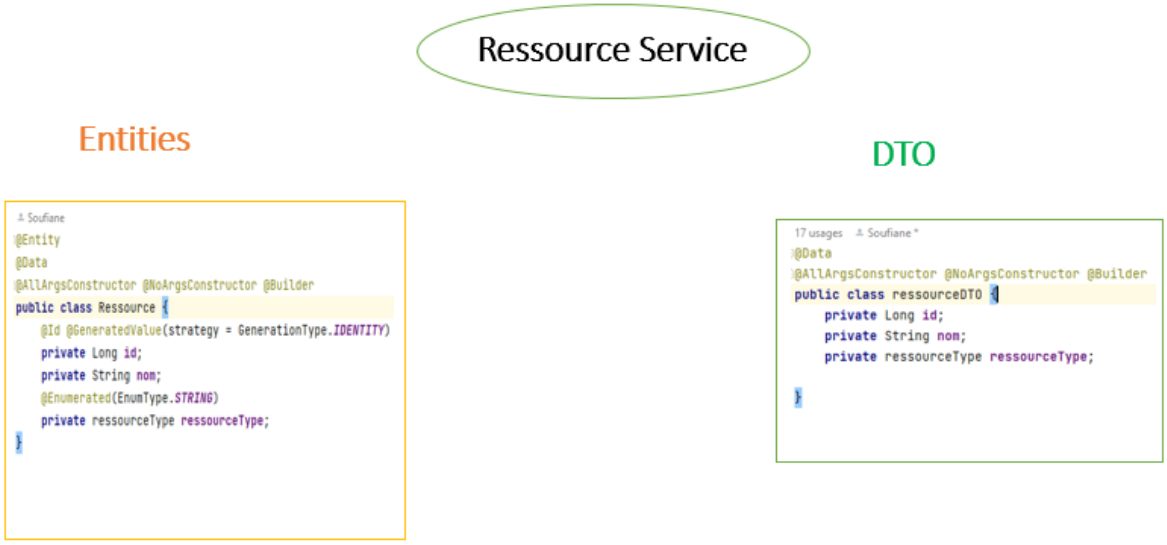

FIGURE 12 – Entities et DTOs

Les entities et DTOs de Reservation Service

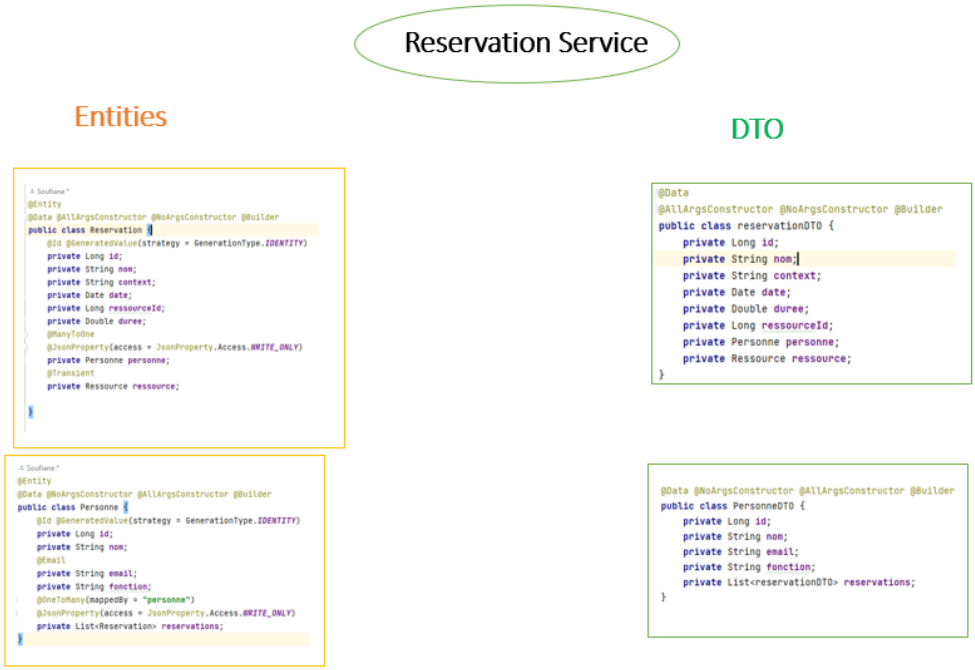

FIGURE 13 – Entities et DTOs

OPEN FEIGN ET CIRCUIT BREAKER

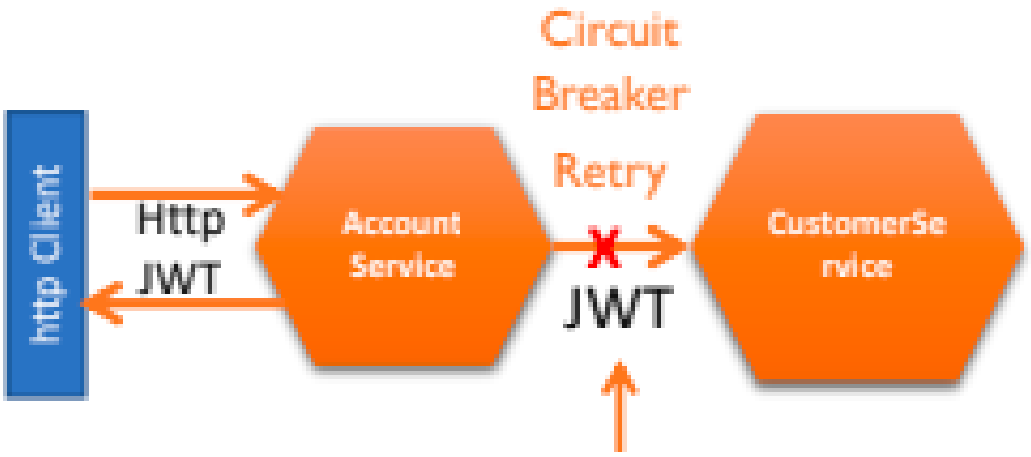

FIGURE 14 – Circuit breaker

Pour communiquer la service reservation avec ressource reservation qunad elle a besoin de donnes de ressource d' un reservation et la gestion de tolerance en panne en cas ou l autre service est n'a pas disponsible.

```
7 usages  Soufiane *
@FeignClient(name = "ressources-service",configuration = FeignAdapter.class)

public interface RessourceRestClientService {
        2 usages  Soufiane *
        @GetMapping("/ressources/{id}")
        @CircuitBreaker(name = "ressourcesService",fallbackMethod = "getDefaultResource")
         Ressource getRessourceById(@PathVariable Long id);
        new *
        @GetMapping("/ressources")
        @Retry(name = "retryAllResources",fallbackMethod = "getDefaultResources")
         List<Ressource> getAllressources();
        1 usage  new *
        @PostMapping("/ressources")
         Ressource addRessources(@RequestBody Ressource ressource);
        new *
        default Ressource getDefaultResource(Long id,Exception e){
                return Ressource.builder()
                        .ressourceType("DEFAULT")
                        .nom("DEFAULT NAME")
                        .id(0L)
                        .build();
        }
        new *
        default List<Ressource> getDefaultResources() { return List.of(); }

}
```

FIGURE 15 – Open Feign et Circuit Breaker

SECURITÉ DES MICROSERVICES

Configuration de la Securité

```
@Configuration
@EnableMethodSecurity(prePostEnabled = true)
@EnableWebSecurity
public class SecurityConfig {
    private JwtAuthConverter jwtAuthConverter;

    public SecurityConfig(JwtAuthConverter jwtAuthConverter) {
        this.jwtAuthConverter = jwtAuthConverter;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        return http
                .csrf(csrf->csrf.disable())
                .authorizeHttpRequests(ar -> ar.requestMatchers("/actuator/**",
                        "/v3/api-docs/**",
                        "/swagger-ui/**",
                        "/swagger-ui.html",
                        "health/**",
                        "/h2-console/**").permitAll())
                .authorizeHttpRequests(ar->ar.anyRequest().authenticated())
                .oauth2ResourceServer(o2->o2.jwt(jwt->jwt.
                jwtAuthenticationConverter(jwtAuthConverter)))
                .headers(h->h.frameOptions(fo->fo.disable()))
                .sessionManagement(sm->sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .build();
    }
    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(Arrays.asList("*"));
        configuration.setAllowedHeaders(Arrays.asList("*"));
        configuration.setAllowedMethods(Arrays.asList("*"));
```

```
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
}
```

Classe Jwt Converter

```
@Component
public class JwtAuthConverter implements Converter<Jwt, AbstractAuthenticationToken> {
    private final JwtGrantedAuthoritiesConverter jwtGrantedAuthoritiesConverter=new
        JwtGrantedAuthoritiesConverter();


    @Override
    public AbstractAuthenticationToken convert(Jwt jwt) {
        Collection<GrantedAuthority> authorities = Stream.concat(
                jwtGrantedAuthoritiesConverter.convert(jwt).stream(),
                extractResourceRoles(jwt).stream()
        ).collect(Collectors.toSet());
        return new JwtAuthenticationToken(jwt,
            authorities,jwt.getClaim("preferred_username"));
    }

    private Collection<GrantedAuthority> extractResourceRoles(Jwt jwt) {
        Map<String , Object> realmAccess;
        Collection<String> roles;
        if(jwt.getClaim("realm_access")==null){
            return Set.of();
        }
        realmAccess = jwt.getClaim("realm_access");
        roles = (Collection<String>) realmAccess.get("roles");
        return roles.stream().map(role->new
            SimpleGrantedAuthority(role)).collect(Collectors.toSet());
    }

}
```

Open Feign Adapter quand la reservation envoyer une resquette [GET, POST, PUT, DELETE] a ressource service et a besoin de acces token.

```
@Component
public class FeignAdapter implements RequestInterceptor {

    @Override
    public void apply(RequestTemplate requestTemplate) {
        SecurityContext context = SecurityContextHolder.getContext();
        JwtAuthenticationToken jwtAuthenticationToken = (JwtAuthenticationToken)
            context.getAuthentication();
        String tokenValue = jwtAuthenticationToken.getToken().getTokenValue();
        requestTemplate.header("Authorization","Bearer "+tokenValue);
    }
}
```

UTILISATION DE KEYCLOAK

Configurqtion de Keycloqk

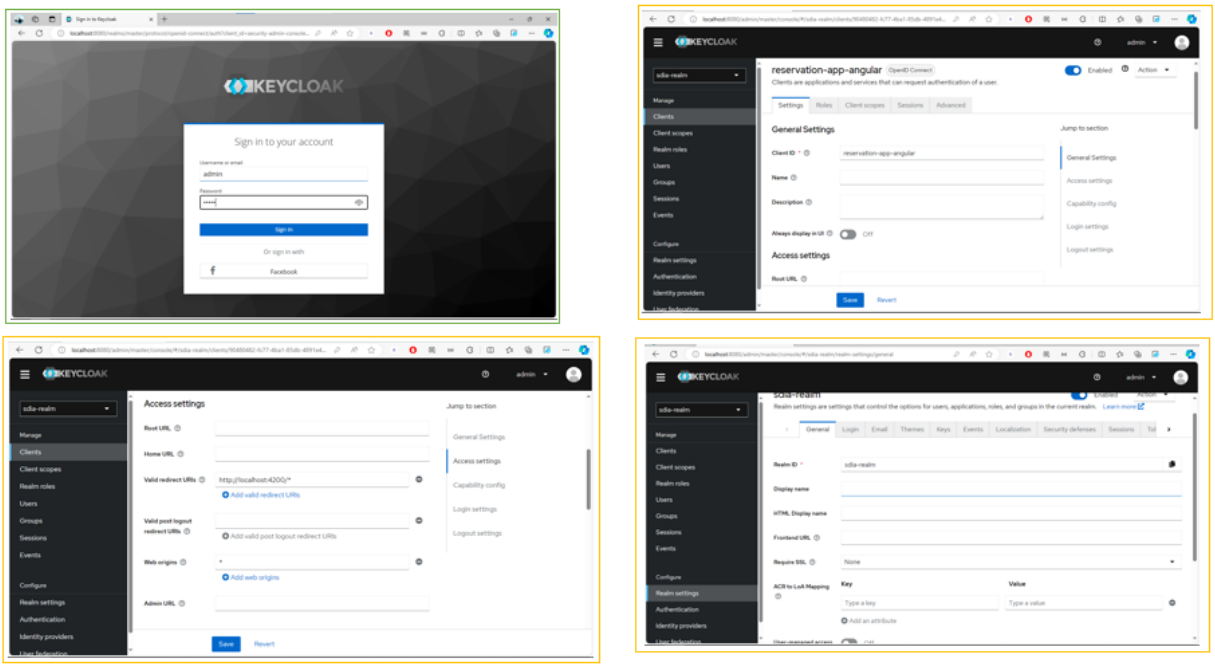

FIGURE 16 – Keycloak

Gestion des utilisateurs dans Keycloqk

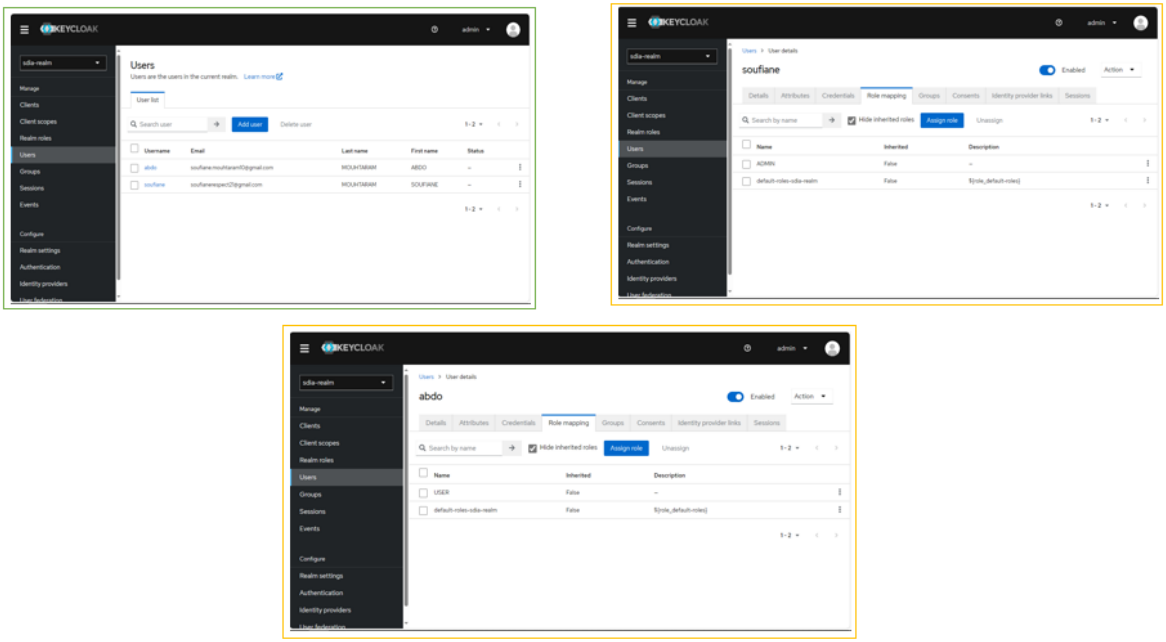

FIGURE 17 – Gestion des utilisateurs

ANGULAR

LapageAccueildenotreFrontendAngular

Faire Authentification pour permettre a acces a les services secures.

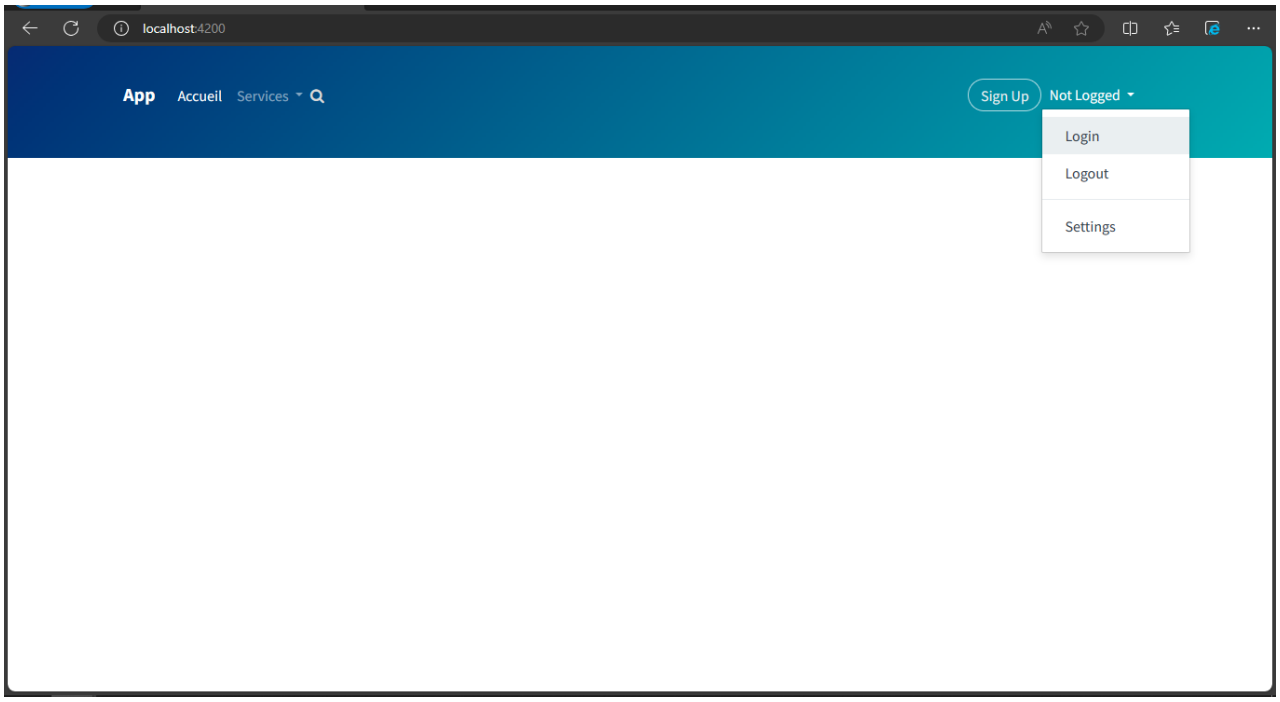

FIGURE 18 – Login

Aprés la clique sur Login il va vous redireger vers Keycloak pour faire l'authentication :

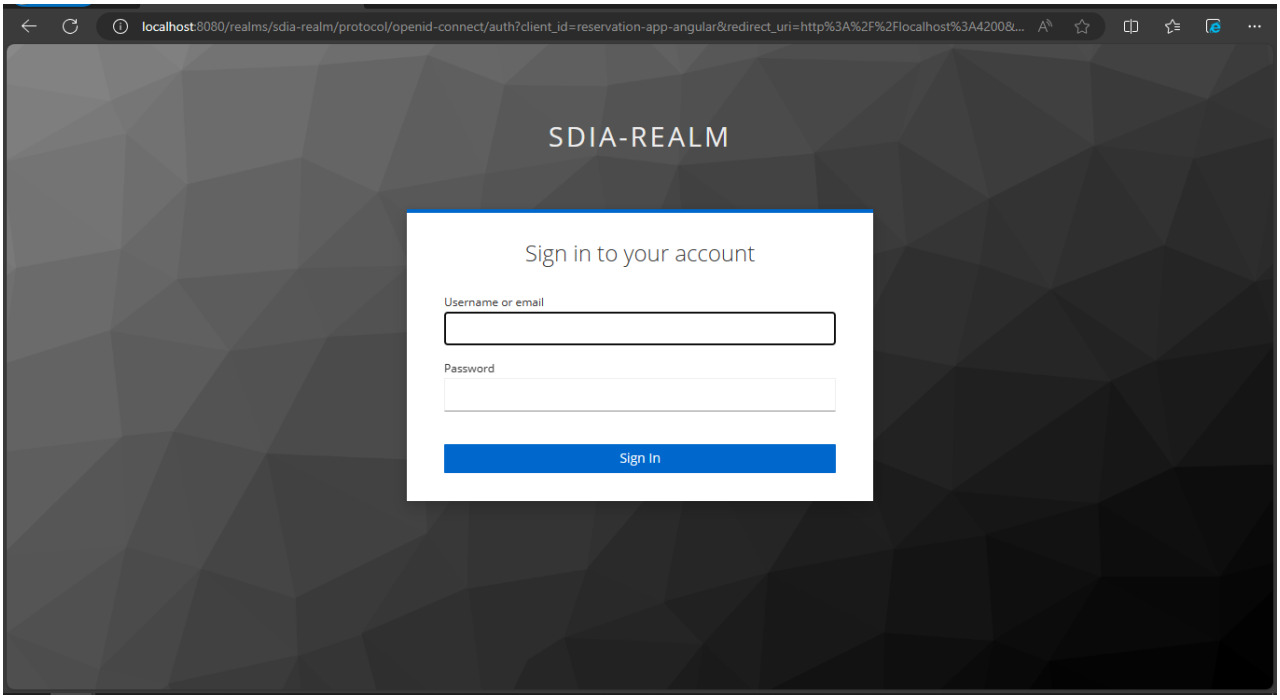

FIGURE 19 – Login User in keycloak

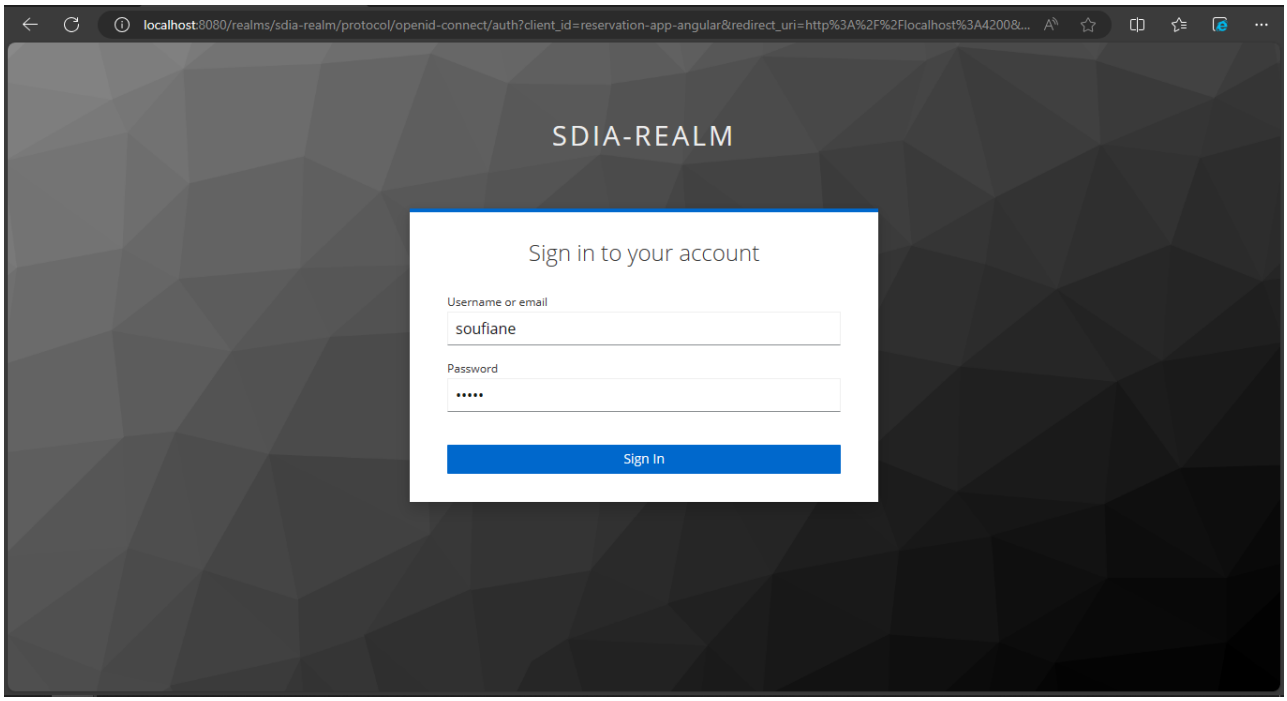

FIGURE 20 – Utilisateur a Role ADMIN

CRUD Angular La page de Utilisateur a role ADMIN il peut faire :

- La suppression.

- La modification.

- L'ajout.

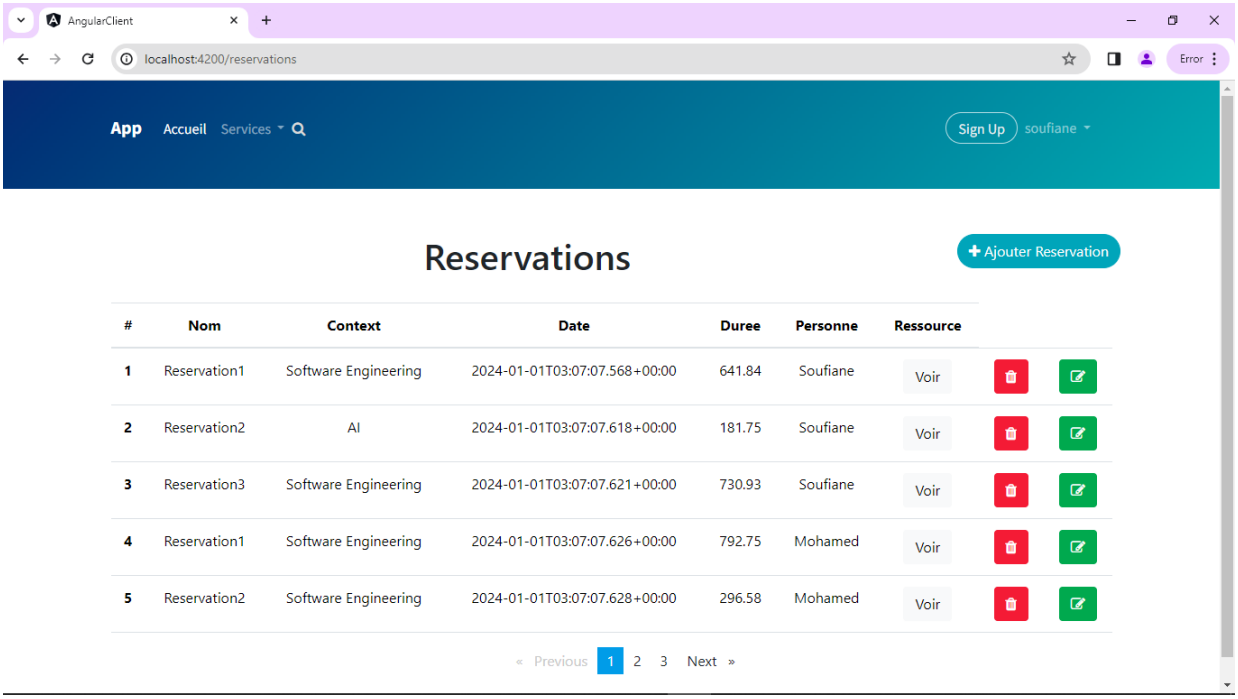

FIGURE 21 – User a Role ADMIN

Modification d' une reservation

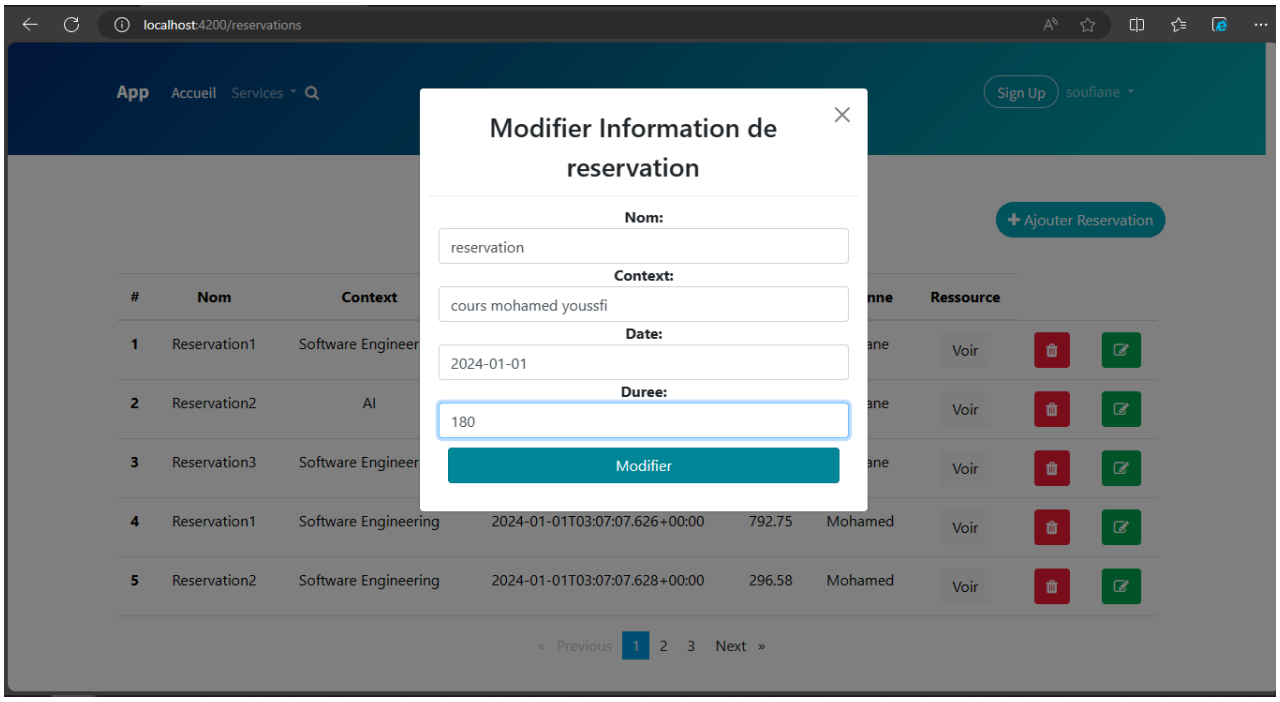

FIGURE 22 – Modification de reservation

Aprés la modification voir la reservation 1

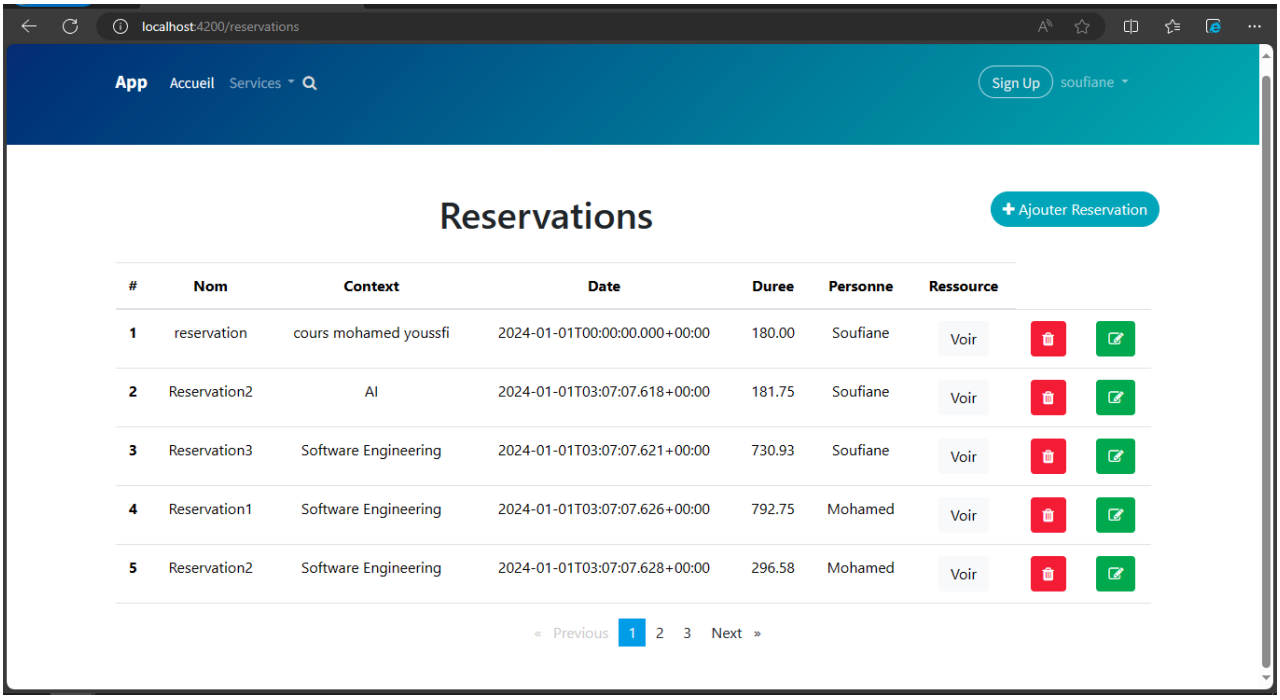

FIGURE 23 – Modification de reservation

Ajout d'une reservation

Ajout d'une reservation avec les donnes d'une ressource et de personne tous ils vont ajouté

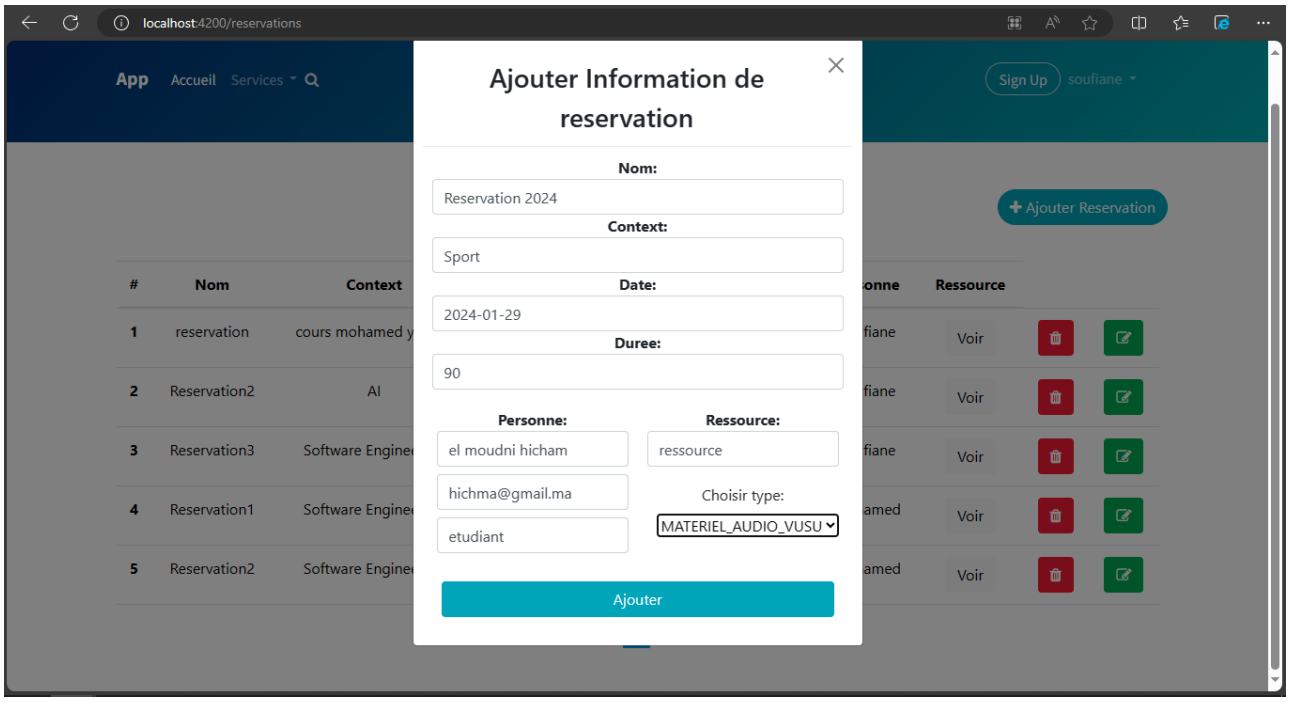

FIGURE 24 – Ajout d'une reservation

Aprés l'ajout d'une reservation

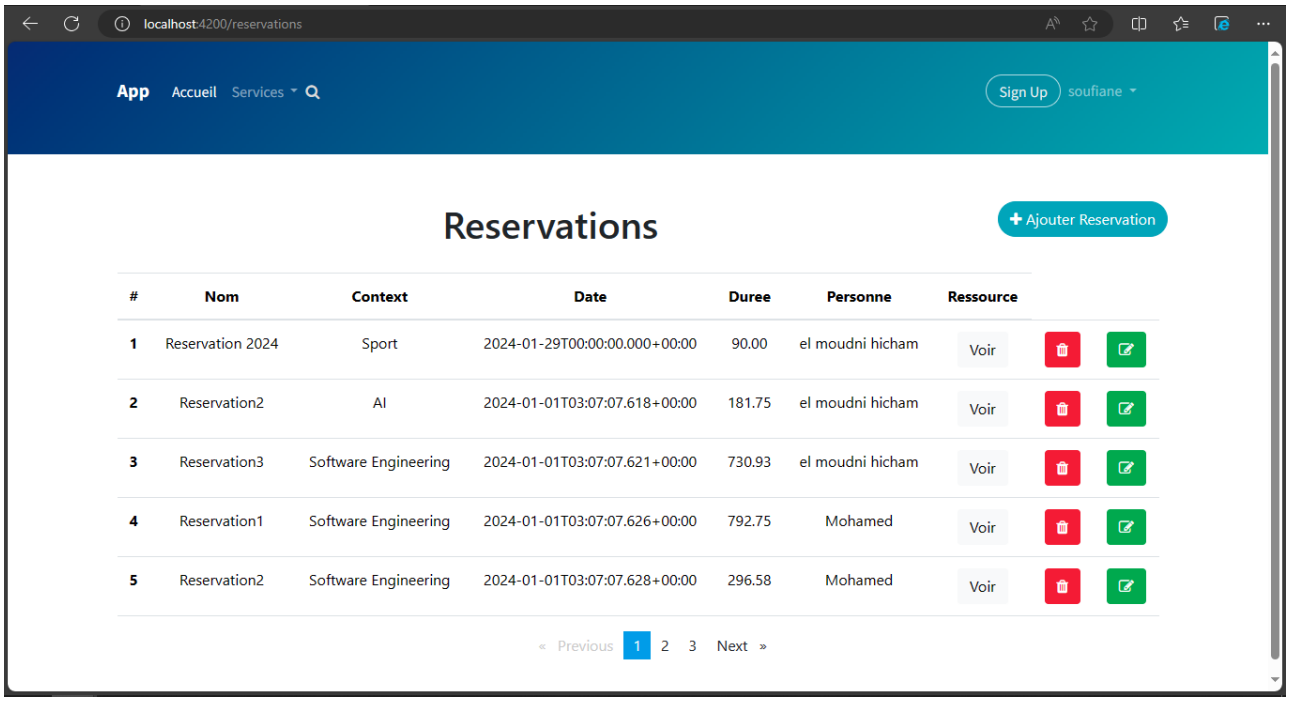

FIGURE 25 – Ajout d'une reservation

La suppression d'une reservation

la suppression de la reservation qu'a id == 1

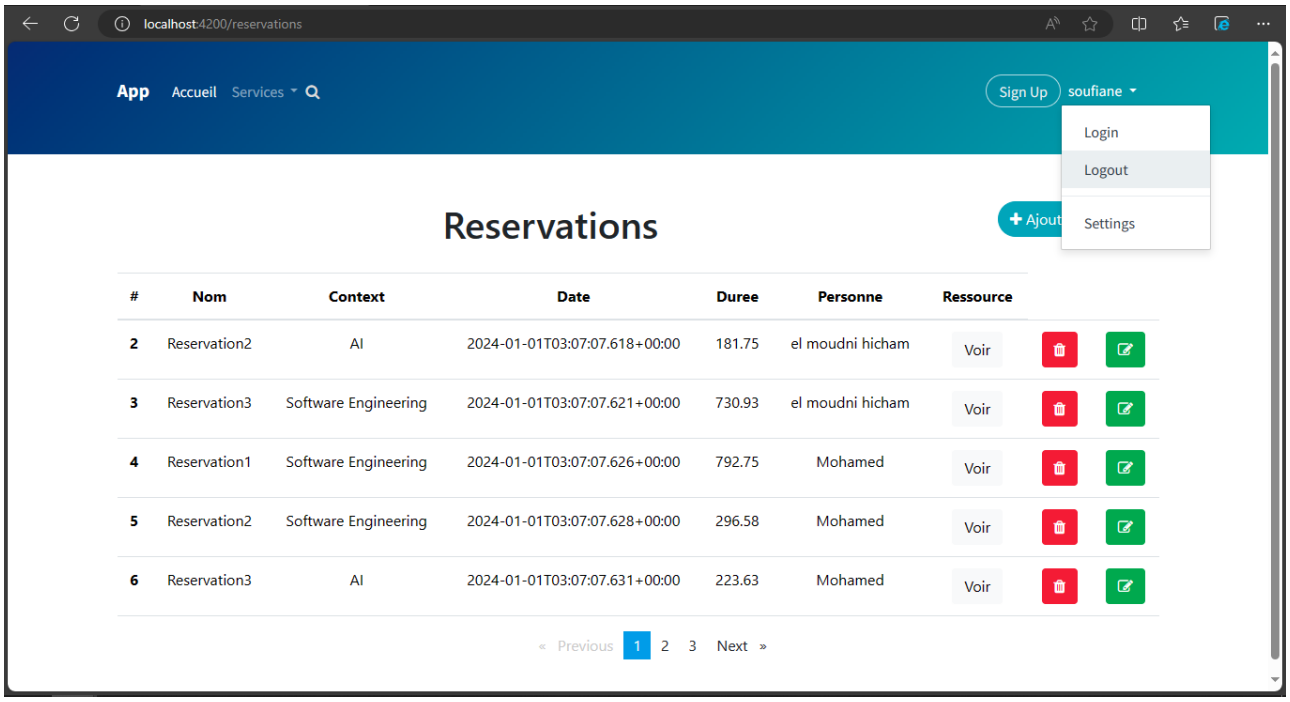

FIGURE 26 – suppression d'une reservation

Authentication avec utilisateur a role USER

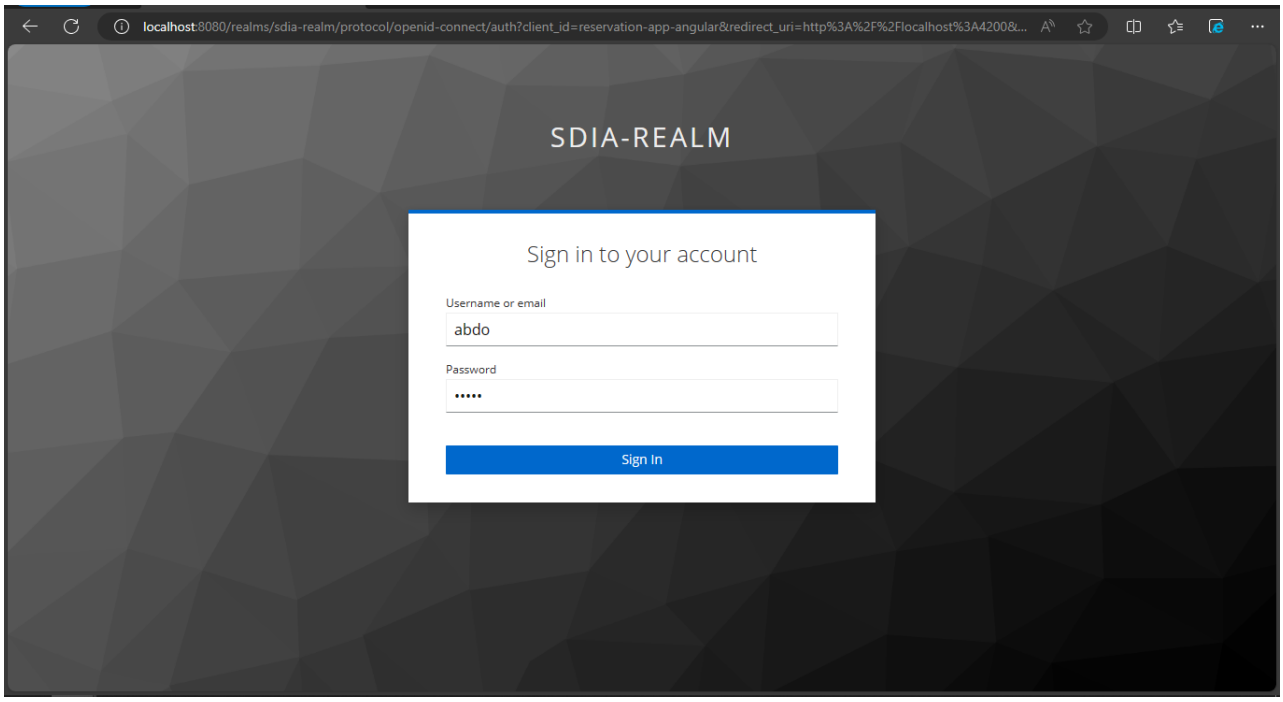

FIGURE 27 – Utilisateur a Role USER

Reservation Page

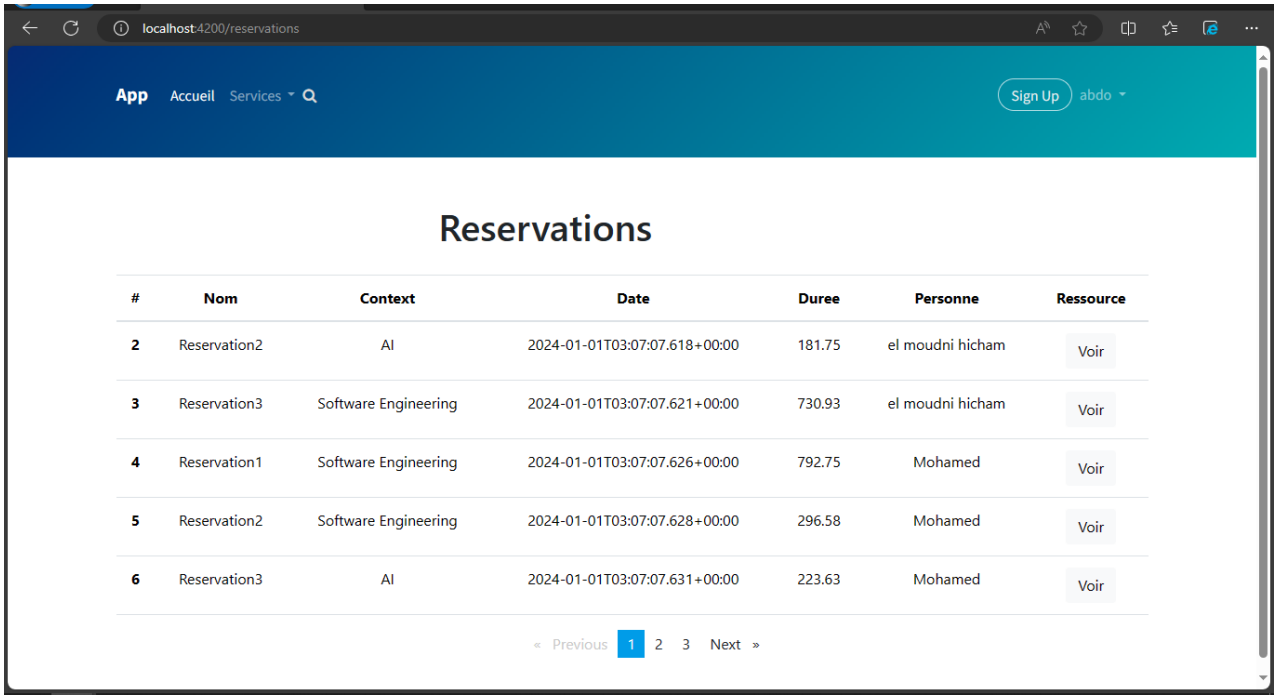

FIGURE 28 – Utilisateur a Role USER

Ressource Page

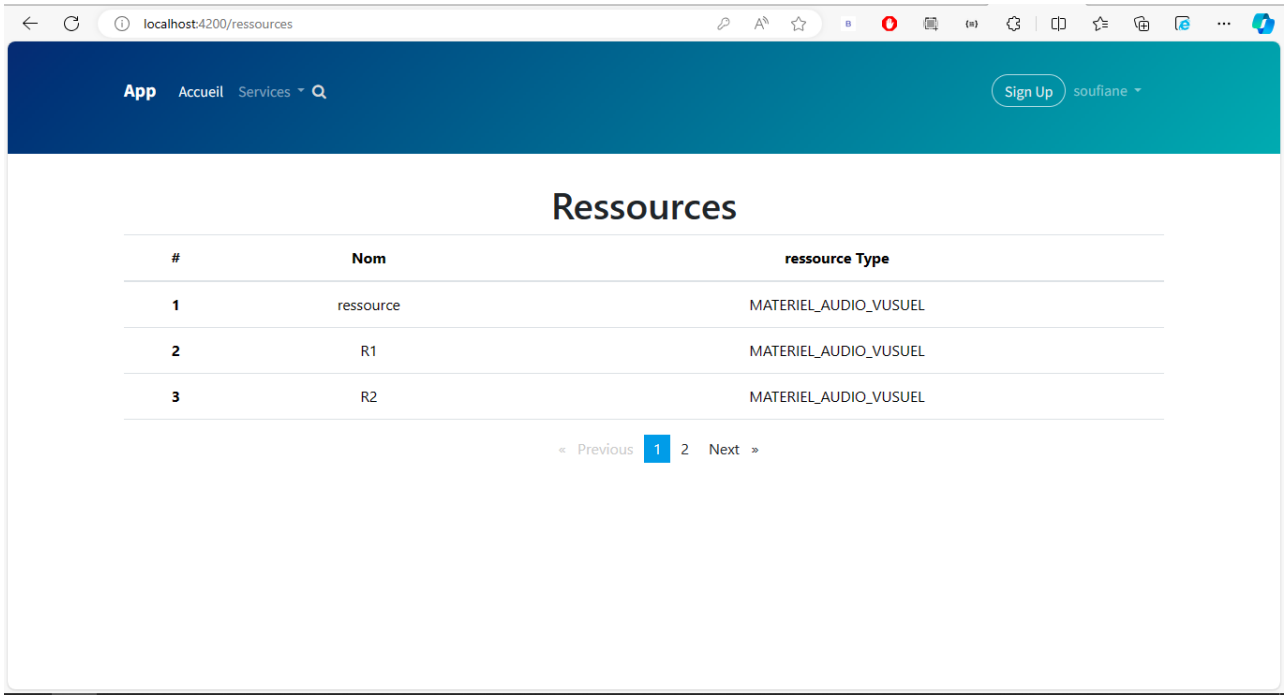

FIGURE 29 – Utilisateur a Role USER