TP : Sécuriser une application avec Spring Security et JWT

Architecture des composants d’entreprise

Table des matières

I. Objectif du TP

L'objectif de cet atelier est de vous montrer comment intégrer Spring Security dans une application développée avec Spring Boot pour implémenter les deux services de base : **l'authentification** et **l'autorisation** en utilisation le Framework JJWT comme implémentation de la norme JWT (Json Web Token).

II. Prérequis

- IntelliJ IDEA ;
- JDK version 17 ;
- Une connexion Internet pour permettre à Maven de télécharger les librairies.
- Le TP n°7 relatif au développement d'une application multi connecteur. Le code source de cet atelier est disponible sur GITHUB : https://github.com/abbouformations/bank-service-multi-connecteur.git.

NB : *Ce TP a été réalisé avec IntelliJ IDEA 2023.2.3 (Ultimate Edition).*

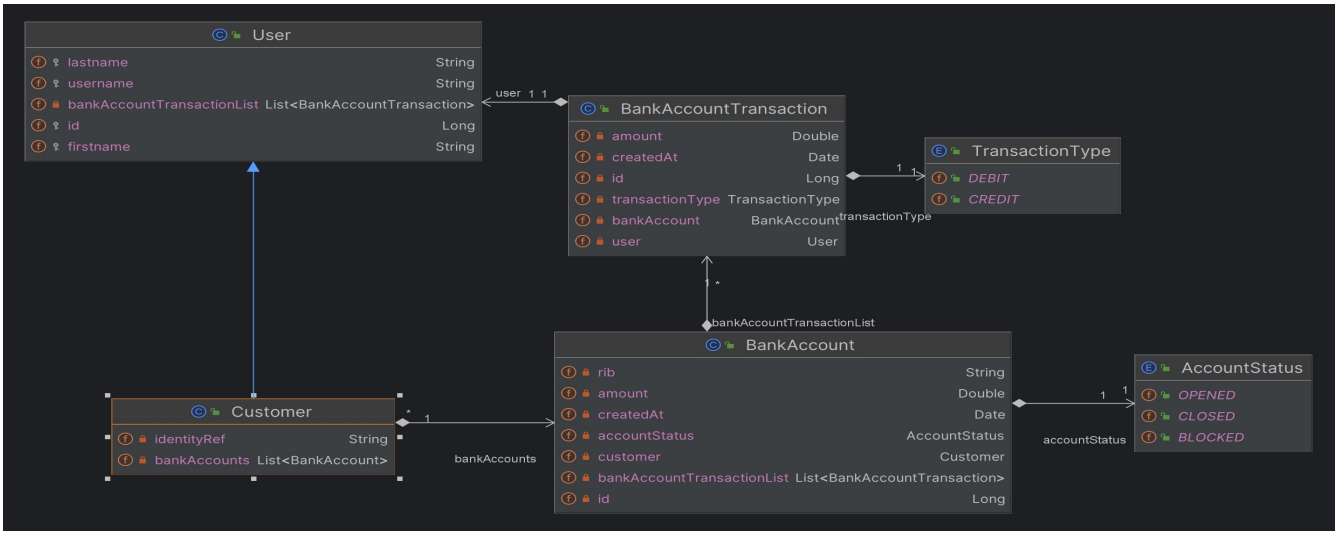
III. Développement de l'application

a. Le diagramme de classe

Nous allons prendre le même diagramme de classe que nous avons implémenté au niveau de l'atelier n°7. Pour rappel, l'application développée offre les services suivants :

- Consulter la liste des clients de la banque.
- Consulter un client par son numéro d'identité.
- Modifier un client par son numéro d'identité.
- Supprimer un client par son numéro d'identité.
- Consulter la liste des comptes bancaires.
- Consulter un compte bancaire par son RIB.
- Effecteur des virements d'un compte vers un autre compte.

Le diagramme de classe est le suivant :



- ✓ Un client peut avoir un ou plieurs comptes bancaires.
- ✓ Sur un compte bancaire, le client peut effectuer une ou plusieurs transactions.
- ✓ Un client est un utilisateur.

- ✓ Un utilisateur (exemple : agent guichet) peut effectuer une ou plusieurs transactions.
- ✓ Un compte bancaire peut avoir les statuts suivants : OPENED, CLOSED ou bien BLOCKED.
- ✓ Une transaction est de deux types : DEBIT ou CREDIT.
- ✓ Aucune transaction ne peut être effectuée sur un compte « CLOSED » ou bien « BLOCKED ».
- ✓ Pour effectuer un virement, le solde du compte bancaire de l'émetteur doit être supérieur au montant du virement.
- ✓ L'identité du client est unique.
- ✓ Le nom d'utilisateur (*username*) est unique.

b. Les autorisations à implémenter

- Le tableau ci-dessous définit les autorisations que nous allons implémenter dans cet atelier :

❖ **Les permissions :**

Le lien	La permission
/api/rest/customer/agent_guichet/all	GET_ALL_CUSTUMERS
/api/rest/customer/identity/*	GET_CUSTOMER_BY_IDENTITY
/api/rest/customer/agent_guichet/create	CREATE_CUSTOMER
/api/rest/customer/agent_guichet/update/*	UPDATE_CUSTOMER
/api/rest/customer/agent_guichet/delete/*	DELETE_CUSTOMER
/api/rest/bank/all	GET_ALL_BANK_ACCOUNT
/api/rest/bank?rib=[your_RIB]	GET_BANK_ACCOUNT_BY_RIB
/api/rest/bank/create	CREATE_BANK_ACCOUNT
/api/rest/transaction/create	ADD_WIRED_TRANSFER
/api/rest/transaction?*	GET_TRANSACTIONS

❖ **Les rôles :**

Rôle	Permission
ROLE_AGENT_GUICHET	GET_ALL_CUSTUMERS, GET_CUSTOMER_BY_IDENTITY CREATE_CUSTOMER, UPDATE_CUSTOMER DELETE_CUSTOMER, GET_ALL_BANK_ACCOUNT GET_BANK_ACCOUNT_BY_RIB, CREATE_BANK_ACCOUNT
ROLE_AGENT_GUICHET_GET	GET_ALL_CUSTUMERS, GET_CUSTOMER_BY_IDENTITY GET_ALL_BANK_ACCOUNT, GET_BANK_ACCOUNT_BY_RIB
ROLE_CLIENT	GET_CUSTOMER_BY_IDENTITY, GET_BANK_ACCOUNT_BY_RIB ADD_WIRED_TRANSFER, GET_TRANSACTIONS

Pour cet atelier, nous allons créer les utilisateurs avec les rôles suivants :

Utilisateur (username)	**Permission**
agentguichet	ROLE_AGENT_GUICHET
agentguichet2	ROLE_AGENT_GUICHET_GET
client	ROLE_CLIENT
admin	[ROLE_AGENT_GUICHET, ROLE_CLIENT]

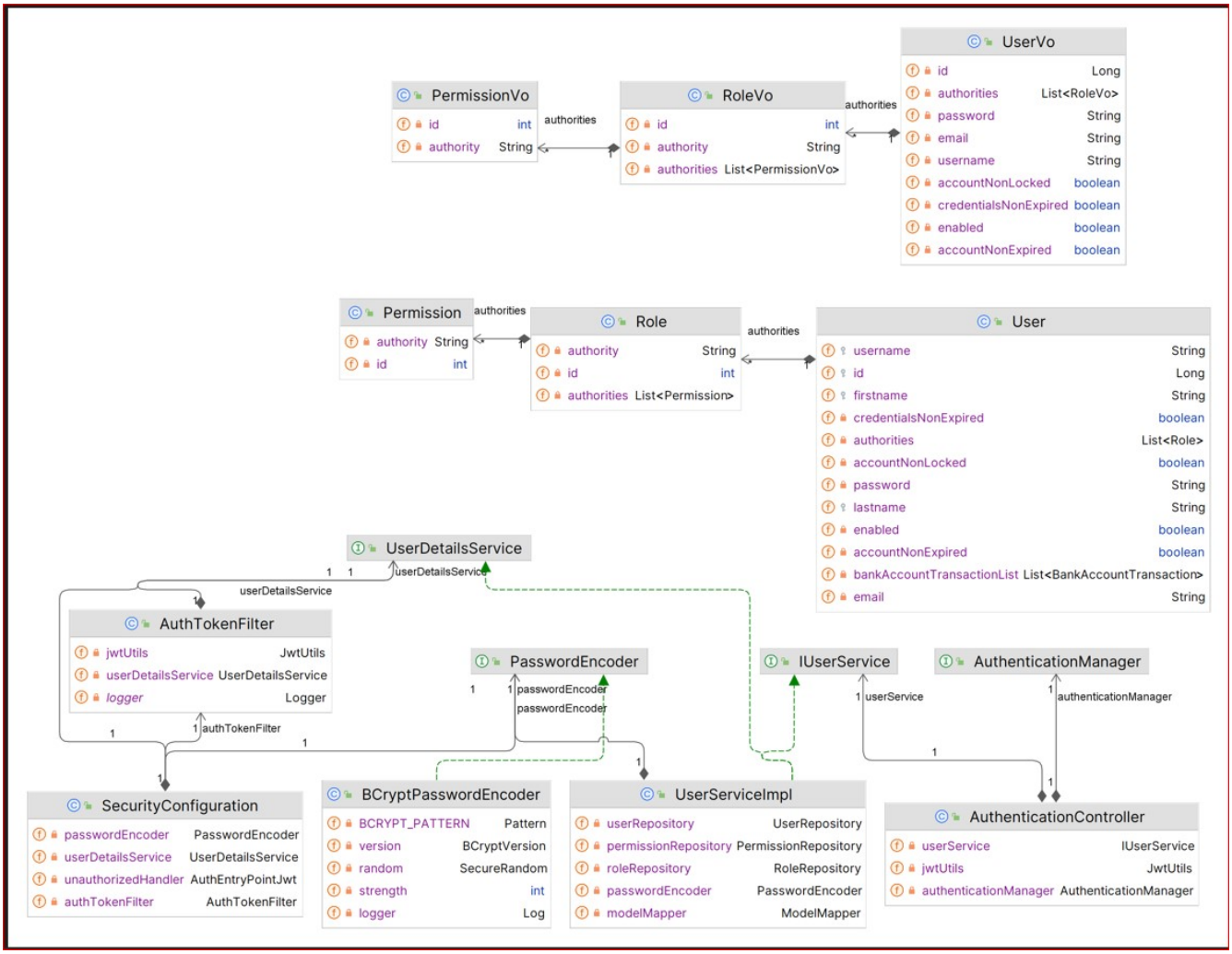
c. Clone de l'application

- Ouvrir un terminal.
- Se positionner par exemple dans le dossier c:\workspace\tp9 et lancer la commande suivante :

git clone https://github.com/abbouformations/bank-service-multi-connecteur.git

- Vérifier que le projet nommé bank-service-multi-connecteur a été bien crée.
- Renommer le projet par exemple ***bank-service-multi-connecteur-jwt***.
- Modifier également le nom de l'ArtifactID dans pom.xml et ouvrir le projet avec Intellig.

d. Le diagramme de classe concernant l'authentification et l'autorisation



Explications :

- La classe ***UserVo*** implémente l'interface ***UserDetails*** de Spring Security. En effet, l'objet représentant l'utilisateur connecté doit être de type ***UserDetails***. Au niveau de cette interface, Spring Security définit les méthodes suivantes :

getUsername()	Pour récupérer username (le login).
getPassword()	Pour récupérer le mot de passe.
isAccountNonExpired()	Pour vérifier si le compte n'est pas expiré.
isAccountNonLocked()	Pour vérifier si le compte n'est pas bloqué.
idCredentialsNonExpired()	Pour vérifier si le mot de passe n'est pas expiré.
isEnabled()	Pour vérifier si l'utilisateur est activé.
getAuthorities()	Pour récupérer les autorisations de l'utilisateur. Une autorisation est un objet de type GrantedAuthority. Au niveau de cette interface, l'autorité est la chaine de caractère ***authority***.

- La classe ***RoleVo*** implémente l'interface ***GrantedAutority*** de Spring Security. Le droit d'accès est représenté par un objet de type ***GrantedAutority***.
- Les deux services à savoir l'authentification et l'autorisation sont implémentés au niveau de la classe ***SecurityConfiguration***.
- Les mots de passes sont cryptés et décryptés moyennant la classe ***BCryptPasswordEncoder*** fournie par Spring Security. Nous pouvons toujours implémenter notre propre algorithme de cryptage en implémentant simplement l'interface ***PasswordEncoder*** de Spring Security.
- La classe ***AuthTokenFilter*** est le filtre qui a pour rôle : le traitement de chaque requête, la vérification de l'entête et la validation du Token d'accès. Ce filtre est exécuté avant le filtre ***UsernamePasswordAuthenticationFilter*** de Spring Security. Cette configuration est faite au niveau de la méthode ***filterChain()*** de la classe ***SecurityConfiguration :***

```
http.addFilterBefore(authTokenFilter, UsernamePasswordAuthenticationFilter.class);
```

- Le service qui vérifie la validité d'un compte utilisateur est implémenté par la classe ***UserServiceImpl***. En effet, cette dernière implémente l'interface ***UserDetailsService*** de Spring Security. Cette interface définie la méthode ***loadUserByUsername(..).***
- Les CORS (*Cross-origin resource sharing*) sont définies moyennant le Bean ***CorsConfigurationSource***. Ce dernier a été instancié au niveau de la classe ***SecurityConfiguration***. Au niveau de la méthode ***corsConfigurationSource(),*** nous avons autorisé au lien http://localhost:3000 d'accéder au WS de notre application backend. Nous avons également défini les méthodes http autorisées ainsi que les entêtes autorisés au niveau des requêtes http.

e. Le fichier pom.xml

Ajouter les deux dépendances suivantes au niveau du fichier pom.xml :

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt</artifactId>
   <version>0.9.1</version>
</dependency>
```

Explications :

- **JJWT** est une implémentation de la norme JWT.
- **spring-boot-starter-security** est le starter de Spring Security qui permet à Spring Boot de configurer automatiquement le Framework Spring Security.

f. Le fichier application.properties

```
# private key
privite_key=@zeRtY193!
#1*24*60*60*1000; 1 jours
expiration_delay=86400000
```

Explications :

- Au niveau de ce fichier, nous avons configuré la clé de signature pour la création du Token JWT ainsi que le délai de validité du Token. Dans cet atelier, le délai de validité du Token est d'une journée.

g. Les classes modèles

- Modifier la classe User comme suit :

```
package ma.formations.service.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.List;

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Data
@Inheritance(strategy = InheritanceType.JOINED)
public class User {
   @Id
   @GeneratedValue
   protected Long id;
   protected String username;
   protected String firstname;
   protected String lastname;
   private String password;
   @OneToMany(mappedBy = "user")
   private List<BankAccountTransaction> bankAccountTransactionList;
   @ManyToMany(cascade = CascadeType.ALL)
   @JoinTable(name = "USER_ROLE")

   private List<Role> authorities = new ArrayList<Role>();
   private boolean enabled;
   private boolean accountNonExpired;
   private boolean credentialsNonExpired;
   private boolean accountNonLocked;
   private String email;

   public User(String username) {
      this.username = username;
```

```
    }
}
```

Explications :

- Au niveau de cette classe, nous avons ajouté les mêmes attributs définis dans l'interface ***UserDetails***, à savoir : ***enabled***, ***accountNonExpired***, ***credentialsNonExpired***, ***accountNonLocked*** et ***Authorities.***
  Ces données seront donc stockées dans la base de données. Vous pouvez par la suite implémenter vos propres règles de gestion concernant :
  - ✓ L'activation ou la désactivation de l'utilisateur ;
  - ✓ L'expiration du compte ;
  - ✓ L'expiration du mot de passe ;
  - ✓ Le verrouillage du compte.
- Créer la classe ***Role*** suivante :

```
package ma.formations.service.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.List;

@Data
@Entity
@NoArgsConstructor
@Builder
@AllArgsConstructor
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    @Column(unique = true)
    private String authority;
    @ManyToMany(cascade = CascadeType.ALL)
    private List<Permission> authorities = new ArrayList<>();
}
```

Explications :

- Remarquer que le nom de l'attribut qui représente le droit (ou l'autorité) est le même que celui défini dans la classe ***RoleVo*** et ceci afin que le ModelMapper puisse effectuer la conversion VO<->BO automatiquement.

- Créer la classe ***Permission*** suivante :

```
package ma.formations.service.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Entity
@NoArgsConstructor
@Builder
@AllArgsConstructor
public class Permission {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private int id;
   @Column(unique = true)
   private String authority;
}
```

h. Les classes DTO

- Modifier la classe ***UserVo*** comme suit :

```
package ma.formations.dtos.user;

import jakarta.validation.constraints.NotEmpty;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.ArrayList;
import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class UserVo implements UserDetails {
   private Long id;
   @NotEmpty
   private String username;
   @NotEmpty
   private String password;
   private boolean accountNonExpired;
   private boolean accountNonLocked;
   private boolean credentialsNonExpired;
   private boolean enabled;
   private List<RoleVo> authorities = new ArrayList<RoleVo>();
   private String email;
}
```

Explications :

- Remarquer que cette classe implémente l'interface ***UserDetails*** de ***Spring Security***.

- Créer la classe **RoleVo** comme suit :

```
package ma.formations.dtos.user;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.security.core.GrantedAuthority;

import java.util.ArrayList;
import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class RoleVo implements GrantedAuthority {
   private int id;
   //this field contains the role user, for example : ROLE_ADMIN
   private String authority;
   private List<PermissionVo> authorities = new ArrayList<>();
}
```

Explications :

- Remarquer que cette classe implémente l'interface ***GrantedAuthority*** de ***Spring Security***.

- Créer la classe ***PermissionVo*** comme suit :

```
package ma.formations.dtos.user;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class PermissionVo {
   private int id;
   //this field contains the authority, for example : GET_ALL_CUSTUMERS
   private String authority;
}
```

- Créer la classe ***TokenVo*** comme suit :

```
package ma.formations.dtos;

import lombok.*;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class TokenVo implements Serializable {
   private String username;
   private String jwtToken;
   private List<String> roles = new ArrayList<>();
}
```

Explications :

- Cette classe est utilisée pour transmettre les données au client via Rest.

- Créer le record ***UserRequest*** suivant :

```
package ma.formations.dtos.user;

public record UserRequest(String username,String password) {
}
```

Explications :

- Il s'agit d'une classe de type Record (nouveau type introduit dans JDK 16) dont les champs sont immutables.
- Cette classe sert juste à encapsuler les données envoyées par la couche front-end.

- Créer le record ***CreateUserRequest*** suivant :

```
package ma.formations.dtos.user;

public record CreateUserRequest(String username, String password, String email) {
}
```

i. Implémenter l'interface UserDetailsService

- Créer l'interface ***IUserService*** :

```
package ma.formations.service;


import ma.formations.dtos.user.PermissionVo;
import ma.formations.dtos.user.RoleVo;
import ma.formations.dtos.user.UserVo;
```

```
public interface IUserService {
   void save(UserVo user);
   void save(RoleVo role);
   void save(PermissionVo vo);
   RoleVo getRoleByName(String role);
   PermissionVo getPermissionByName(String authority);

}
```

- Créer la classe ***UserServiceImpl*** :

```
package ma.formations.service;

import lombok.AllArgsConstructor;
import ma.formations.dao.PermissionRepository;
import ma.formations.dao.RoleRepository;
import ma.formations.dao.UserRepository;
import ma.formations.dtos.user.PermissionVo;
import ma.formations.dtos.user.RoleVo;
import ma.formations.dtos.user.UserVo;
import ma.formations.service.model.Permission;
import ma.formations.service.model.Role;
import ma.formations.service.model.User;
import org.modelmapper.ModelMapper;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

@Service
@Transactional
@AllArgsConstructor
public class UserServiceImpl implements IUserService, UserDetailsService {
   private UserRepository userRepository;
   private RoleRepository roleRepository;
   private PasswordEncoder passwordEncoder;
   private ModelMapper modelMapper;
   private PermissionRepository permissionRepository;

   @Override
   public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
      UserVo userVo = modelMapper.map(userRepository.findByUsername(username), UserVo.class);
      List<RoleVo> permissions = new ArrayList<>();
      userVo.getAuthorities().forEach(
            roleVo -> roleVo.getAuthorities().forEach(
                  permission -> permissions.add(
                        RoleVo.builder().authority(permission.getAuthority()).build())));
      userVo.getAuthorities().addAll(permissions);
      return userVo;
```

```
    }

    @Override
    public void save(UserVo userVo) {
        User user = modelMapper.map(userVo, User.class);
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        user.setAuthorities(user.getAuthorities().stream().map(bo ->
                    roleRepository.findByAuthority(bo.getAuthority()).get()).
                collect(Collectors.toList()));
        userRepository.save(user);
    }

    @Override
    public void save(RoleVo roleVo) {
        Role role = modelMapper.map(roleVo, Role.class);
        role.setAuthorities(role.getAuthorities().stream().map(bo ->
                    permissionRepository.findByAuthority(bo.getAuthority()).get()).
                collect(Collectors.toList()));
        roleRepository.save(role);
    }

    @Override
    public void save(PermissionVo vo) {
        permissionRepository.save(modelMapper.map(vo, Permission.class));
    }

    @Override
    public RoleVo getRoleByName(String authority) {
        return modelMapper.map(roleRepository.findByAuthority(authority).get(), RoleVo.class);
    }

    @Override
    public PermissionVo getPermissionByName(String authority) {
        return modelMapper.map(permissionRepository.findByAuthority(authority), PermissionVo.class);
    }
}
```

j. La classe utilitaire de JJWT

- Créer la classe ***JwtUtils*** suivante :

```
package ma.formations.jwt;

import io.jsonwebtoken.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.util.Date;
import java.util.HashMap;
```

```java
import java.util.Map;
import java.util.stream.Collectors;

@Component
public class JwtUtils {
   private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);

   @Value("${privite_key}")
   private String jwtSecret;

   @Value("${expiration_delay}")
   private int delaiExpiration;

   public String generateJwtToken(Authentication authentication) {

      UserDetails userPrincipal = (UserDetails) authentication.getPrincipal();
      Map<String, Object> credentials = new HashMap<>();
      credentials.put("roles", userPrincipal.getAuthorities().stream().
            map(GrantedAuthority::getAuthority).
            collect(Collectors.toList()));
      credentials.put("sub", userPrincipal.getUsername());
      return Jwts.builder().setClaims(credentials).setIssuedAt(new Date())
            .setExpiration(new Date((new Date()).getTime() + delaiExpiration))
            .signWith(SignatureAlgorithm.HS512, jwtSecret).compact();
   }

   public String getUserNameFromJwtToken(String token) {
      return Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();
   }

   public boolean validateJwtToken(String authToken) {
      try {
         Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
         return true;
      } catch (SignatureException e) {
         logger.error("Invalid JWT signature: {}", e.getMessage());
      } catch (MalformedJwtException e) {
         logger.error("Invalid JWT token: {}", e.getMessage());
      } catch (ExpiredJwtException e) {
         logger.error("JWT token is expired: {}", e.getMessage());
      } catch (UnsupportedJwtException e) {
         logger.error("JWT token is unsupported: {}", e.getMessage());
      } catch (IllegalArgumentException e) {
         logger.error("JWT claims string is empty: {}", e.getMessage());
      }
      return false;
   }
}
```

k. La classe Filter

- Créer la classe ***AuthTokenFilter*** suivante :

```java
package ma.formations.jwt;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.AllArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@AllArgsConstructor
public class AuthTokenFilter extends OncePerRequestFilter {
   private static final Logger logger = LoggerFactory.getLogger(AuthTokenFilter.class);
   private JwtUtils jwtUtils;
   private UserDetailsService userDetailsService;

   @Override
   protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
         throws ServletException, IOException {
      try {
         String jwt = parseJwt(request);
         if (jwt != null && jwtUtils.validateJwtToken(jwt)) {
            String username = jwtUtils.getUserNameFromJwtToken(jwt);
            UserDetails userDetails = userDetailsService.loadUserByUsername(username);
            UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(
                  userDetails, null, userDetails.getAuthorities());
            authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authentication);
         }
      } catch (Exception e) {
         logger.error("Cannot set user authentication: {}", e);
      }
      filterChain.doFilter(request, response);
   }

   private String parseJwt(HttpServletRequest request) {
      String headerAuth = request.getHeader("Authorization");
      if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {
         return headerAuth.substring(7);
      }
      return null;
   }
}
```

l. La classe AuthenticationEntryPoint

Créer la classe **AuthenticationEntryPoint** suivante :

```
package ma.formations.jwt;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import java.io.IOException;

@Component
public class AuthEntryPointJwt implements AuthenticationEntryPoint {

   private static final Logger logger = LoggerFactory.getLogger(AuthEntryPointJwt.class);

   @Override
   public void commence(HttpServletRequest request, HttpServletResponse response,
                        AuthenticationException authException) throws IOException, ServletException {
      logger.error("Unauthorized error: {}", authException.getMessage());
      response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Error: Unauthorized");
   }

}
```

m. La classe de configuration

- Créer la classe **SecurityConfiguration** suivante :

```
package ma.formations.config;

import lombok.AllArgsConstructor;
import ma.formations.jwt.AuthEntryPointJwt;
import ma.formations.jwt.AuthTokenFilter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.Customizer;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
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.config.annotation.web.configuration.WebSecurityCustomizer;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
```

```
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.Arrays;
import java.util.List;

@Configuration
@EnableWebSecurity
@AllArgsConstructor
@EnableMethodSecurity
public class SecurityConfiguration {
   private UserDetailsService userDetailsService;
   private PasswordEncoder passwordEncoder;
   private AuthTokenFilter authTokenFilter;
   private AuthEntryPointJwt unauthorizedHandler;

   @Bean
   public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration)
         throws Exception {
      return authenticationConfiguration.getAuthenticationManager();
   }


   @Bean
   public AuthenticationProvider authenticationProvider() {
      DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
      authenticationProvider.setUserDetailsService(userDetailsService);
      authenticationProvider.setPasswordEncoder(passwordEncoder);
      return authenticationProvider;
   }

   @Bean
   CorsConfigurationSource corsConfigurationSource() {
      CorsConfiguration configuration = new CorsConfiguration();
      configuration.setAllowedOrigins(List.of("http://localhost:3000"));
      configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
      configuration.setAllowedHeaders(Arrays.asList("Authorization", "Requestor-Type", "Content-Type"));
      UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
      source.registerCorsConfiguration("/**", configuration);
      return source;
   }

   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http.cors(Customizer.withDefaults());
      http.csrf(AbstractHttpConfigurer::disable);
      http.authorizeHttpRequests(auth -> {
         auth.requestMatchers(
               new AntPathRequestMatcher("/"),
               new AntPathRequestMatcher("/auth/**"),
               new AntPathRequestMatcher("/h2/**"),
               new AntPathRequestMatcher("/h2/login.do**")
         ).permitAll();

        /*  auth.requestMatchers(new AntPathRequestMatcher("/api/rest/customer/identity/**")).
```

```
                hasAnyAuthority("AGENT_GUICHET", "CLIENT");
            auth.requestMatchers(new AntPathRequestMatcher("/api/rest/customer/agent_guichet/**")).
                hasAuthority("AGENT_GUICHET");*/
            auth.anyRequest().authenticated();
        });
        http.authenticationProvider(authenticationProvider());
        http.exceptionHandling((exception) -> exception.authenticationEntryPoint(unauthorizedHandler).
                accessDeniedPage("/access-denied"));
        http.sessionManagement(sess -> sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
        http.addFilterBefore(authTokenFilter, UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    @Bean
    public WebSecurityCustomizer webSecurityCustomizer() {
        return (web) -> web.ignoring().requestMatchers(
                new AntPathRequestMatcher("/h2/**"),
                new AntPathRequestMatcher("/static/**"),
                new AntPathRequestMatcher("/css/**"),
                new AntPathRequestMatcher("/js/**",
                    "/images/**"));
    }
}
```

Explications :

- Dans la méthode filterChain, remarquer que nous avons commenté les lignes de code pour lesquelles nous avons configurer les accès aux différents endpoints de notre application. En effet, Spring Security propose deux façons pour configurer les autorisations :
  - ✓ Soit avec la configuration dans la méthode filterChain
  - ✓ Soit avec l'utilisation des annotations ***@PreAuthorize***, ***@Secured***, ***@PostAuthorize*** sur les méthodes directement. Dans ce cas, il faut demander à Spring security d'activer la sécurité sur les méthodes en annotant la classe de configuration par : ***@EnableMethodSecurity***.

n. La classe d'authentification

- Créer la classe ***AuthenticationController*** suivante :

```
package ma.formations.presentation.auth;

import lombok.AllArgsConstructor;
import ma.formations.dtos.user.*;
import ma.formations.jwt.JwtUtils;
import ma.formations.service.IUserService;
import ma.formations.service.exception.BusinessException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RestController;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/auth")
@AllArgsConstructor
public class AuthenticationController {
   private final AuthenticationManager authenticationManager;
   private final JwtUtils jwtUtils;
   private IUserService userService;

   @PostMapping("/signin")
   public ResponseEntity<TokenVo> authenticateUser(@RequestBody UserRequest userRequest) {
      try {
         Authentication authentication = authenticationManager
               .authenticate(new UsernamePasswordAuthenticationToken(userRequest.username(),
userRequest.password()));
         SecurityContextHolder.getContext().setAuthentication(authentication);
         String jwt = jwtUtils.generateJwtToken(authentication);
         TokenVo tokenVo = TokenVo.builder().
               jwtToken(jwt).
               username(userRequest.username()).
               roles(authentication.getAuthorities().stream().
                     map(GrantedAuthority::getAuthority).
                     collect(Collectors.toList())).build();
         return ResponseEntity.ok(tokenVo);
      } catch (Exception e) {
         throw new BusinessException("Login ou mot de passe incorrect");
      }
   }

   @PostMapping("/signup")
   public ResponseEntity<String> createUser(@RequestBody CreateUserRequest createUserRequest) {
      userService.save(UserVo.builder().
            username(createUserRequest.username()).
            password(createUserRequest.password()).
            email(createUserRequest.email()).
            enabled(true).
            accountNonExpired(true).
            accountNonLocked(true).
            credentialsNonExpired(true).
            authorities(List.of(RoleVo.builder().authority("ROLE_CLIENT").build())).
            build());

      return new ResponseEntity<>(String.format("User [%s] created with success",
            createUserRequest.username()), HttpStatus.CREATED);
   }
}
```

o. Le contrôleur CustomerRestController

```
package ma.formations.presentation.rest;

import jakarta.validation.Valid;
import ma.formations.dtos.customer.*;
```

```java
import ma.formations.service.ICustomerService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/rest/customer")
@PreAuthorize("hasAnyRole('ADMIN','AGENT_GUICHET','CLIENT')")
public class CustomerRestController {

    private final ICustomerService customerService;

    public CustomerRestController(ICustomerService customerService) {
        this.customerService = customerService;
    }

    @GetMapping("/agent_guichet/all")
    @PreAuthorize("hasAuthority('GET_ALL_CUSTUMERS')")
    List<CustomerDto> customers() {
        return customerService.getAllCustomers();
    }

    @GetMapping("/identity/{identity}")
    @PreAuthorize("hasAuthority('GET_CUSTOMER_BY_IDENTITY')")
    CustomerDto customerByIdentity(@PathVariable(value = "identity") String identity) {
        return customerService.getCustomByIdentity(identity);
    }

    @PostMapping("/agent_guichet/create")
    @PreAuthorize("hasAuthority('CREATE_CUSTOMER')")
    public ResponseEntity<AddCustomerResponse> createCustomer(@RequestBody @Valid AddCustomerRequest dto) {
        return new ResponseEntity<>(customerService.createCustomer(dto), HttpStatus.CREATED);
    }

    @PutMapping("/agent_guichet/update/{identityRef}")
    @PreAuthorize("hasAuthority('UPDATE_CUSTOMER')")
    public ResponseEntity<UpdateCustomerResponse> updateCustomer(@PathVariable String identityRef, @RequestBody
@Valid UpdateCustomerRequest dto) {
        return new ResponseEntity<>(customerService.updateCustomer(identityRef, dto), HttpStatus.OK);
    }

    @DeleteMapping("/agent_guichet/delete/{identityRef}")
    @PreAuthorize("hasAuthority('DELETE_CUSTOMER')")
    public ResponseEntity<String> deleteCustomer(@PathVariable String identityRef) {
        customerService.deleteCustomerByIdentityRef(identityRef);
        return new ResponseEntity<>(String.format("Customer with identity %s is removed", identityRef), HttpStatus.OK);
    }
}
```

Explications :

- Remarquer l'utilisation de l'annotation ***@PreAuthorize*** qui permet de configurer l'autorité que l'utilisateur doit avoir pour avoir l'accès à la méthode.

- L'annotation ***@PreAuthorize*** utilise ***Spring EL*** (Spring Expression Langage).
- Si vous annotez uniquement la classe avec ***@PreAuthorize("hasAnyRole('ADMIN','AGENT_GUICHET')")*** par exemple, l'utilisateur ayant l'un de ces deux rôles aura l'accès à toutes les méthodes de la classe.
- Remarquer que le nom du rôle que nous avons passé dans l'expression de Spring EL est ADMIN et non pas **ROLE_ADMIN**. Par défaut, les rôles dans Spring Security commencent par la chaine **ROLE_**
- Pour que Spring Security active l'utilisation de l'annotation ***@PreAuthorize***, il faut annoter la classe de configuration, à savoir la classe ***SecurityConfiguration***, par : ***@EnableMethodSecurity***

p. Le contrôleur BankAccountRestController

```
package ma.formations.presentation.rest;

import jakarta.validation.Valid;
import ma.formations.dtos.bankaccount.AddBankAccountRequest;
import ma.formations.dtos.bankaccount.AddBankAccountResponse;
import ma.formations.dtos.bankaccount.BankAccountDto;
import ma.formations.service.IBankAccountService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/rest/bank")
//@CrossOrigin(origins = "*", allowedHeaders = "*")
@PreAuthorize("hasAnyRole('ADMIN','AGENT_GUICHET','CLIENT')")
public class BankAccountRestController {
   private final IBankAccountService bankAccountService;

   public BankAccountRestController(IBankAccountService bankAccountService) {
      this.bankAccountService = bankAccountService;
   }

   @GetMapping("/all")
   @PreAuthorize("hasAuthority('GET_ALL_BANK_ACCOUNT')")
   List<BankAccountDto> bankAccounts() {
      return bankAccountService.getAllBankAccounts();
   }

   @GetMapping
   @PreAuthorize("hasAuthority('GET_BANK_ACCOUNT_BY_RIB')")
   BankAccountDto bankAccountByRib(@RequestParam(value = "rib") String rib) {
      return bankAccountService.getBankAccountByRib(rib);
   }


   @PostMapping("/create")
   @PreAuthorize("hasAuthority('CREATE_BANK_ACCOUNT')")
   public ResponseEntity<AddBankAccountResponse> addBankAccount(@Valid @RequestBody AddBankAccountRequest
dto) {
```

```
        return new ResponseEntity<>(bankAccountService.saveBankAccount(dto), HttpStatus.CREATED);
    }
}
```

q. Le contrôleur TransactionRestController

```
package ma.formations.presentation.rest;

import jakarta.validation.Valid;
import lombok.AllArgsConstructor;
import ma.formations.common.CommonTools;
import ma.formations.dtos.transaction.AddWirerTransferRequest;
import ma.formations.dtos.transaction.AddWirerTransferResponse;
import ma.formations.dtos.transaction.GetTransactionListRequest;
import ma.formations.dtos.transaction.TransactionDto;
import ma.formations.service.ITransactionService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

import java.util.List;


@AllArgsConstructor
@RestController
@RequestMapping("/api/rest/transaction")
@PreAuthorize("hasRole('CLIENT')")
//@CrossOrigin(origins = "*", allowedHeaders = "*")
public class TransactionRestController {

    private ITransactionService transactionService;
    private CommonTools commonTools;

    @PostMapping("/create")
    @PreAuthorize("hasAuthority('ADD_WIRED_TRANSFER')")
    public ResponseEntity<AddWirerTransferResponse> addWirerTransfer(@Valid @RequestBody
AddWirerTransferRequest dto) {
        return new ResponseEntity<>(transactionService.wiredTransfer(dto), HttpStatus.CREATED);
    }

    @GetMapping
    @PreAuthorize("hasAuthority('GET_TRANSACTIONS')")
    public List<TransactionDto> getTransactions(GetTransactionListRequest dto) {
        return transactionService.getTransactions(dto);
    }
}
```

r. Les enums

- Créer les deux enum ***Roles*** et ***Permissions*** suivants :

```
package ma.formations.enums;

public enum Roles {
   ROLE_AGENT_GUICHET_GET,
   ROLE_AGENT_GUICHET,
   ROLE_CLIENT,
   ROLE_ADMIN
}
```

```
package ma.formations.enums;

public enum Permisssions {
   GET_ALL_CUSTUMERS,
   GET_CUSTOMER_BY_IDENTITY,
   CREATE_CUSTOMER,
   UPDATE_CUSTOMER,
   DELETE_CUSTOMER,
   GET_ALL_BANK_ACCOUNT,
   GET_BANK_ACCOUNT_BY_RIB,
   CREATE_BANK_ACCOUNT,
   ADD_WIRED_TRANSFER,
   GET_TRANSACTIONS
}
```

s. La classe de démarrage

- Modifier la classe de démarrage comme suit :

```
package ma.formations;

import ma.formations.dtos.bankaccount.AddBankAccountRequest;
import ma.formations.dtos.customer.AddCustomerRequest;
import ma.formations.dtos.transaction.AddWirerTransferRequest;
import ma.formations.dtos.user.PermissionVo;
import ma.formations.dtos.user.RoleVo;
import ma.formations.dtos.user.UserVo;
import ma.formations.enums.Permisssions;
import ma.formations.enums.Roles;
import ma.formations.service.IBankAccountService;
import ma.formations.service.ICustomerService;
import ma.formations.service.ITransactionService;
import ma.formations.service.IUserService;
import net.devh.boot.grpc.server.security.authentication.BasicGrpcAuthenticationReader;
import net.devh.boot.grpc.server.security.authentication.GrpcAuthenticationReader;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

import java.util.Arrays;
import java.util.List;

@SpringBootApplication
public class BankServiceApplication {

   public static void main(String[] args) {

      SpringApplication.run(BankServiceApplication.class, args);
   }

   @Bean
   public PasswordEncoder passwordEncoder() {
      return new BCryptPasswordEncoder();
   }

   @Bean
   public GrpcAuthenticationReader grpcAuthenticationReader() {
      return new BasicGrpcAuthenticationReader();
   }


   @Bean
   CommandLineRunner initDataBase(ICustomerService customerService,
                                  IBankAccountService bankAccountService,
                                  ITransactionService transactionService,
                                  IUserService userService) {

      return args -> {
         customerService.createCustomer(AddCustomerRequest.builder().username("user1").
               identityRef("A100").
               firstname("FIRST_NAME1").
               lastname("LAST_NAME1").
               build());

         bankAccountService.saveBankAccount(AddBankAccountRequest.builder().
               rib("RIB_1").
               amount(1000000d).
               customerIdentityRef("A100").
               build());
         bankAccountService.saveBankAccount(AddBankAccountRequest.builder().
               rib("RIB_11").
               amount(2000000d).
               customerIdentityRef("A100").
               build());

         customerService.createCustomer(AddCustomerRequest.builder().
               username("user2").
               identityRef("A200").
               firstname("FIRST_NAME2").
               lastname("LAST_NAME2").
               build());
```

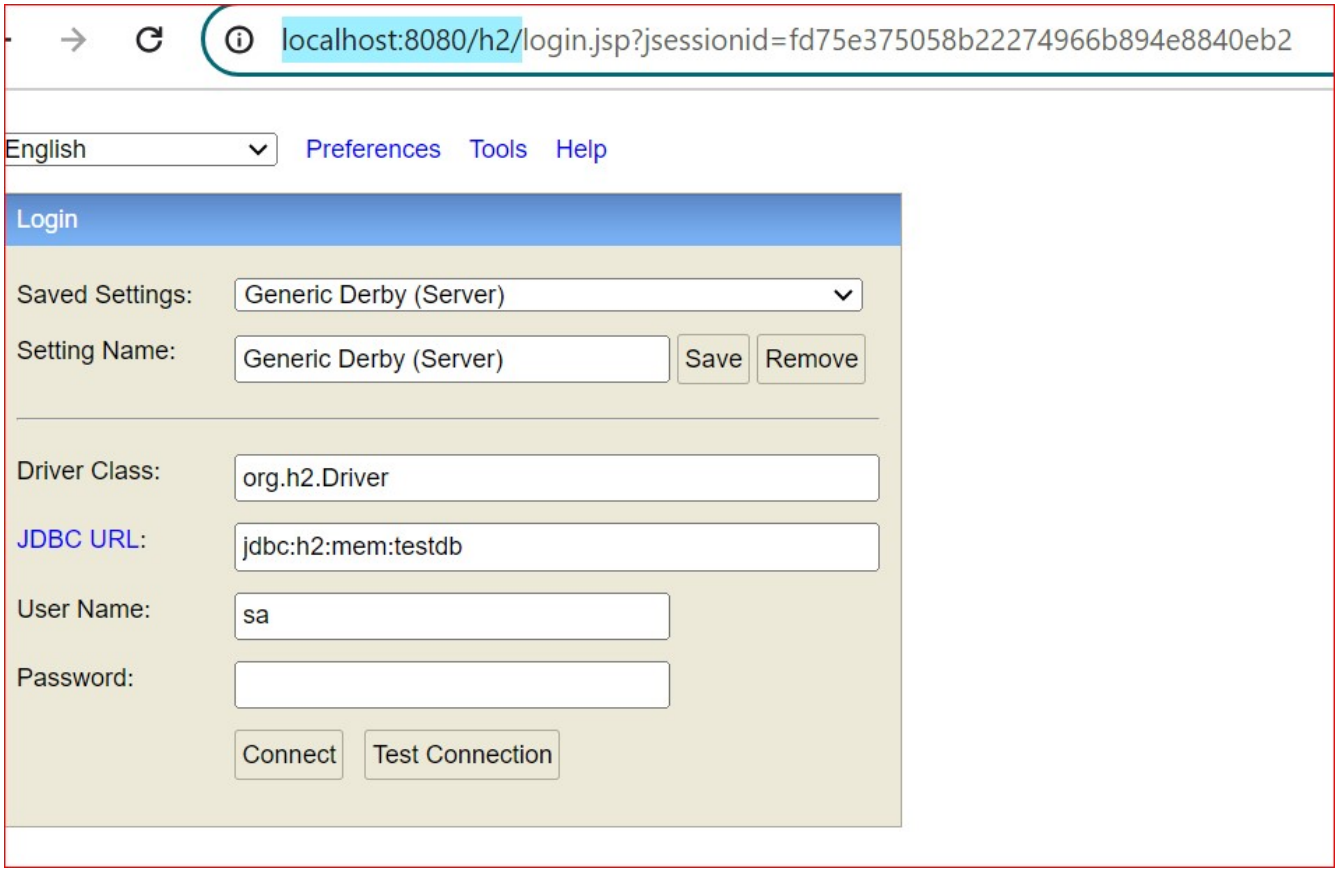
```
bankAccountService.saveBankAccount(AddBankAccountRequest.builder().
        rib("RIB_2").
        amount(2000000d).
        customerIdentityRef("A200").
        build());


customerService.createCustomer(AddCustomerRequest.builder().
        username("user3").
        identityRef("A900").
        firstname("FIRST_NAME9").
        lastname("LAST_NAME9").
        build());

bankAccountService.saveBankAccount(AddBankAccountRequest.builder().
        rib("RIB_9").
        amount(-25000d).
        customerIdentityRef("A900").
        build());


customerService.createCustomer(AddCustomerRequest.builder().
        username("user4").
        identityRef("A800").
        firstname("FIRST_NAME8").
        lastname("LAST_NAME8").
        build());

bankAccountService.saveBankAccount(AddBankAccountRequest.builder().
        rib("RIB_8").
        amount(0.0).
        customerIdentityRef("A800").
        build());

transactionService.wiredTransfer(AddWirerTransferRequest.builder().
        ribFrom("RIB_1").
        ribTo("RIB_2").
        amount(10000.0).
        username("user1").
        build());

transactionService.wiredTransfer(AddWirerTransferRequest.builder().
        ribFrom("RIB_1").
        ribTo("RIB_9").
        amount(20000.0).
        username("user1").
        build());

transactionService.wiredTransfer(AddWirerTransferRequest.builder().
        ribFrom("RIB_1").
        ribTo("RIB_8").
        amount(500.0).
        username("user1").
        build());
```

```
transactionService.wiredTransfer(AddWirerTransferRequest.builder().
        ribFrom("RIB_2").
        ribTo("RIB_11").
        amount(300.0).
        username("user2").
        build());

//Add all permissions
Arrays.stream(Permisssions.values()).toList().forEach(permisssions ->
        userService.save(PermissionVo.builder().authority(permisssions.name()).build()));

//Agent guichet pour toutes les opérations CRUD.
RoleVo roleAgentGuichet = RoleVo.builder().
        authority(Roles.ROLE_AGENT_GUICHET.name()).
        authorities(List.of(
                userService.getPermissionByName(Permisssions.GET_ALL_CUSTUMERS.name()),
                userService.getPermissionByName(Permisssions.GET_CUSTOMER_BY_IDENTITY.name()),
                userService.getPermissionByName(Permisssions.CREATE_CUSTOMER.name()),
                userService.getPermissionByName(Permisssions.UPDATE_CUSTOMER.name()),
                userService.getPermissionByName(Permisssions.DELETE_CUSTOMER.name()),
                userService.getPermissionByName(Permisssions.GET_ALL_BANK_ACCOUNT.name()),
                userService.getPermissionByName(Permisssions.GET_BANK_ACCOUNT_BY_RIB.name()),
                userService.getPermissionByName(Permisssions.CREATE_BANK_ACCOUNT.name()))).
        build();

//Agent guichet pour lecture seule.
RoleVo roleAgentGuichetGet = RoleVo.builder().
        authority(Roles.ROLE_AGENT_GUICHET_GET.name()).
        authorities(List.of(
                userService.getPermissionByName(Permisssions.GET_ALL_CUSTUMERS.name()),
                userService.getPermissionByName(Permisssions.GET_CUSTOMER_BY_IDENTITY.name()),
                userService.getPermissionByName(Permisssions.GET_ALL_BANK_ACCOUNT.name()),
                userService.getPermissionByName(Permisssions.GET_BANK_ACCOUNT_BY_RIB.name()))).
        build();

RoleVo roleClient = RoleVo.builder().
        authority(Roles.ROLE_CLIENT.name()).
        authorities(List.of(
                userService.getPermissionByName(Permisssions.GET_CUSTOMER_BY_IDENTITY.name()),
                userService.getPermissionByName(Permisssions.GET_BANK_ACCOUNT_BY_RIB.name()),
                userService.getPermissionByName(Permisssions.ADD_WIRED_TRANSFER.name()),
                userService.getPermissionByName(Permisssions.GET_TRANSACTIONS.name())
        )).build();

userService.save(roleAgentGuichet);
userService.save(roleAgentGuichetGet);
userService.save(roleClient);

UserVo agentGuichet = UserVo.builder().
        username("agentguichet").
        password("agentguichet").
        authorities(List.of(roleAgentGuichet)).
        accountNonExpired(true).
        accountNonLocked(true).
        credentialsNonExpired(true).
        enabled(true).
        build();
```

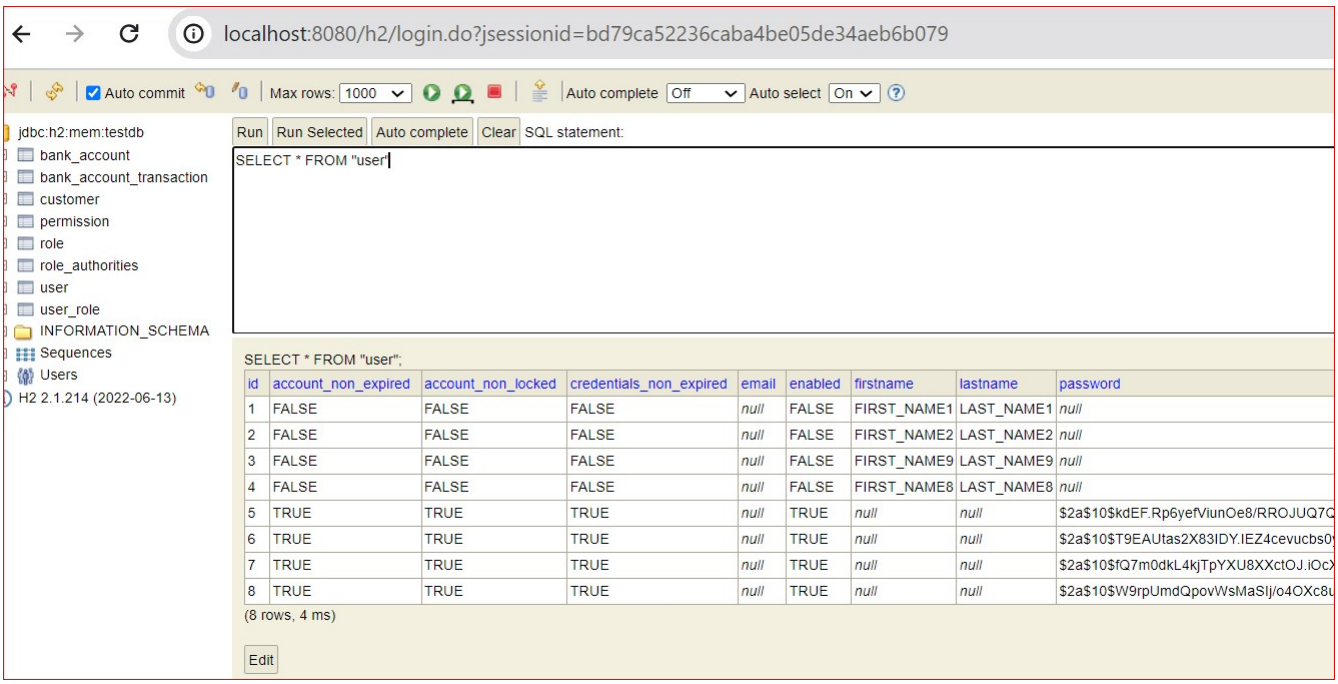
```
        UserVo agentGuichet2 = UserVo.builder().
                username("agentguichet2").
                password("agentguichet2").
                authorities(List.of(roleAgentGuichetGet)).
                accountNonExpired(true).
                accountNonLocked(true).
                credentialsNonExpired(true).
                enabled(true).
                build();


        UserVo client = UserVo.builder().
                username("client").
                password("client").
                authorities(List.of(roleClient)).
                accountNonExpired(true).
                accountNonLocked(true).
                credentialsNonExpired(true).
                enabled(true).
                build();

        UserVo admin = UserVo.builder().
                username("admin").
                password("admin").
                authorities(List.of(roleAgentGuichet, roleClient)).
                accountNonExpired(true).
                accountNonLocked(true).
                credentialsNonExpired(true).
                enabled(true).
                build();
        userService.save(agentGuichet);
        userService.save(agentGuichet2);
        userService.save(client);
        userService.save(admin);
    };
  }
}
```

IV. Tester l'application avec Postman

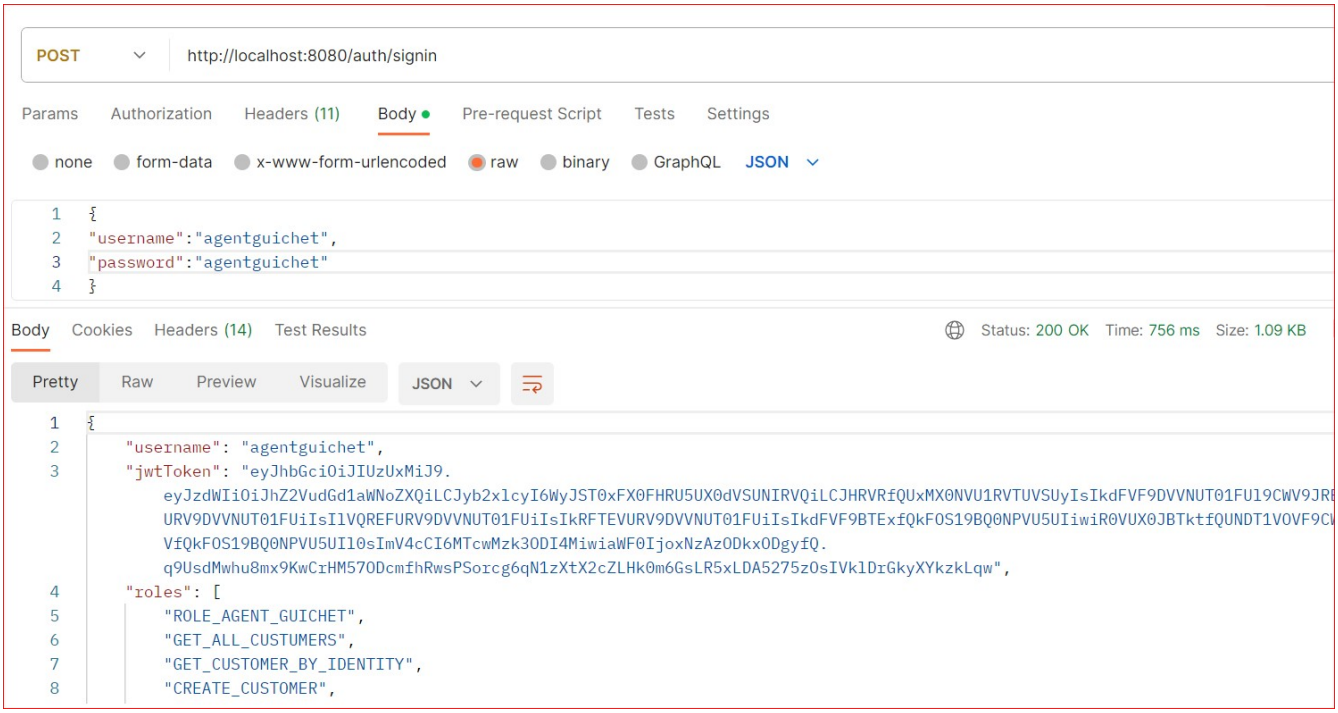
- Démarrer la méthode main de la classe de démarrage et vérifier que les tables ont été créées et initialisées en lançant le lien http://localhost:8080/h2/ :



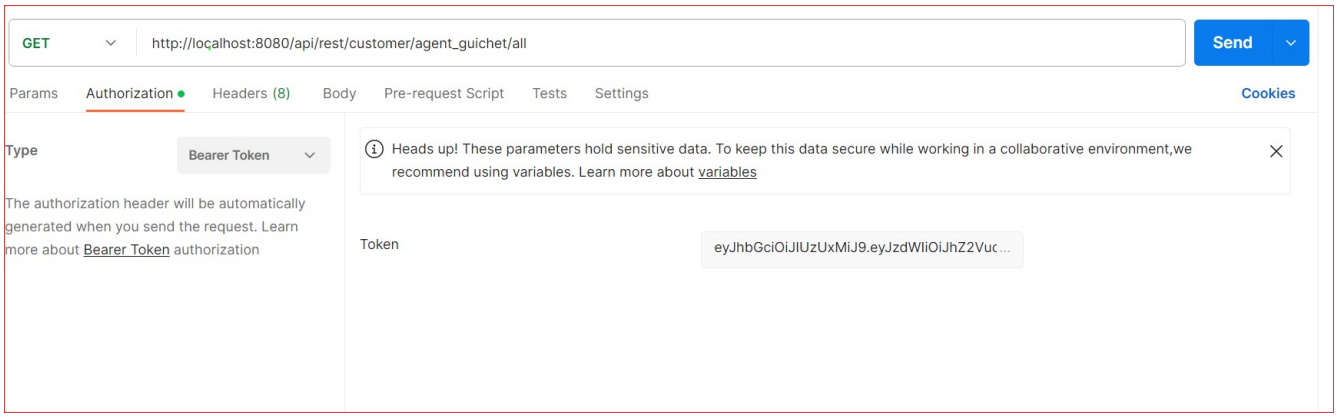
- Cliquer sur Connect et vérifier les tables créées :



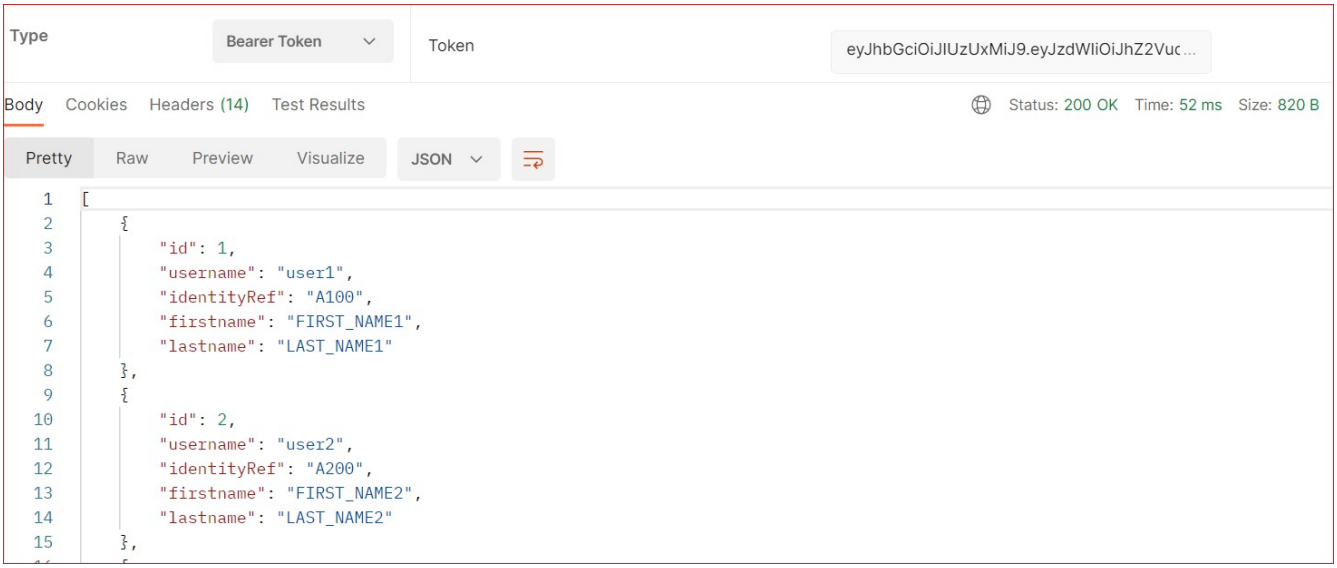
- Tester l'authentification avec le compte utilisateur [usrename=agentguichet, password= agentguichet] comme illustré par l'écran suivant :



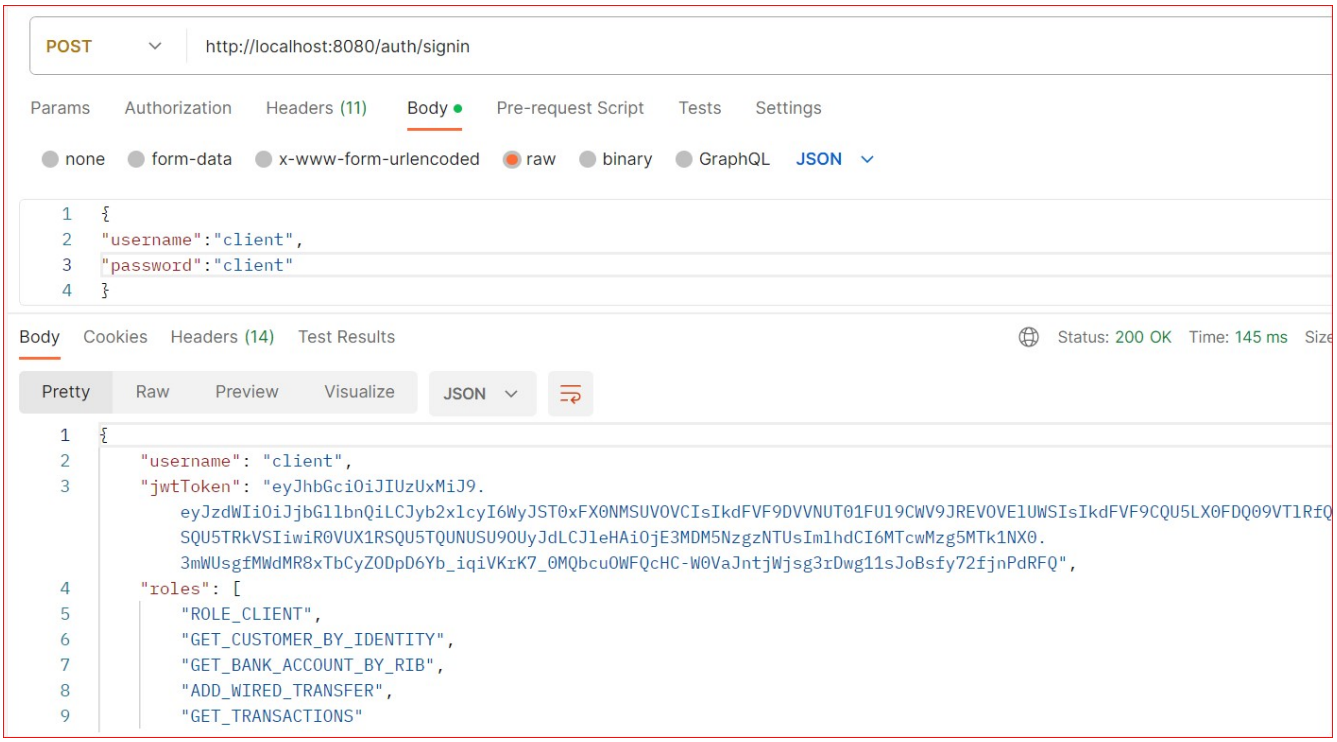
- Pour tester le droit d'accès au lien http://localhost:8080/api/rest/customer/agent_guichet/all, suivre les étapes suivantes :



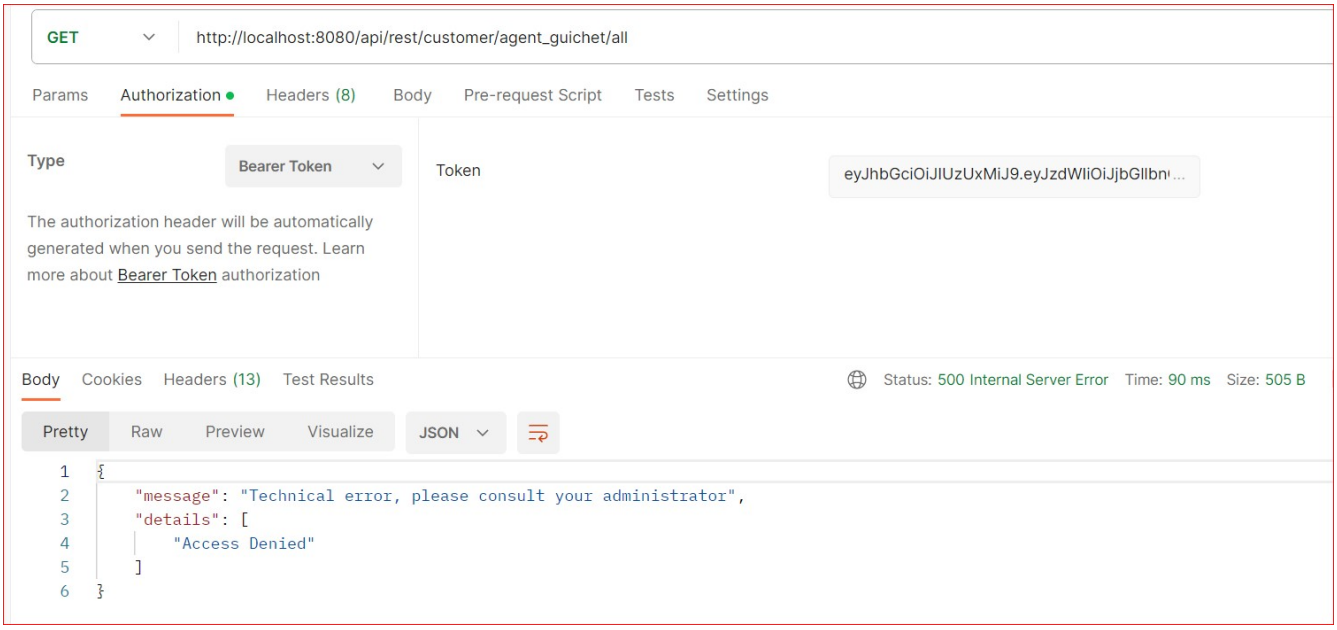
- Préciser GET dans la méthode http.
- Taper l'URL.
- Dans *Authorization*, choiser le type ***Bearer Token*** et entrer le Token généré ci-dessus dans le champ Token.
- Cliquer ensuite sur le bouton Send. Vérifier que le résultat est 200 OK :



- Générer le Token pour l'utilisateur client/client :



- Tester l'accès au lien http://localhost:8080/api/rest/customer/agent_guichet/all avec le Token généré et vérifier que le code http envoyé est 401 (Unauthorized) :



- Faites les autres tests avec les comtes utilisateurs : agentguichet2 ayant le rôle ROLE_AGENT_GUICHET_GET et admin ayant les rôles : ROLE_CLIENT et ROLE_AGENT_GUICHET.

Conclusion

Le code source de cet atelier est disponible sur GITHUB :

https://github.com/abbouformations/bank-service-multi-connecteur-jwt.git