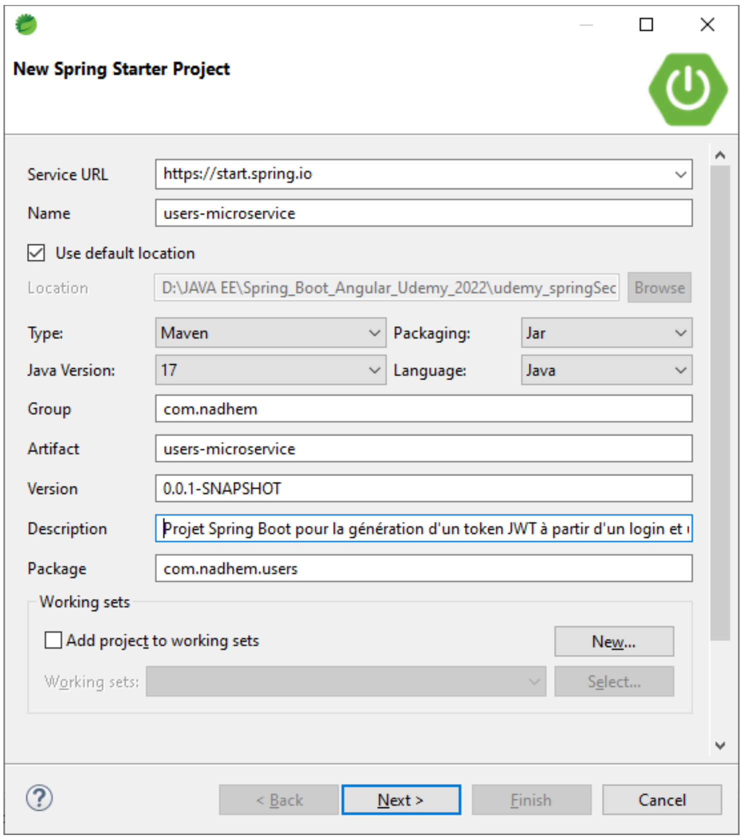
Spring Boot : Génération du JWT Token

Objectifs :

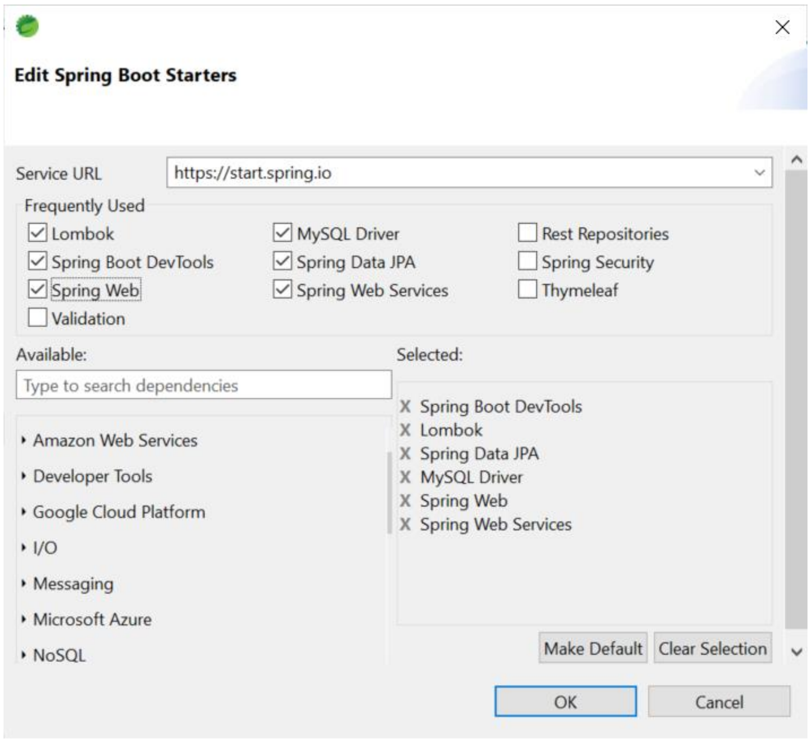
1. Créer le projet et ajouter les dépendances,
2. Editer le fichier *application.properties*,
3. Créer les entités *User* et *Role* et leurs interfaces *Repository*,
4. Ajouter *Spring Security* et *auth0* au projet,
5. Créer la couche service,
6. Créer la classe *SecurityConfig*,
7. Créer la classe *MyUserDetailsService*,
8. Générer le token *JWT* à la suite d'une authentication,
9. Tester la génération du *JWT* avec *POSTMAN*.

Créer le projet et ajouter les dépendances

1. Créer le projet *users-microservice* et ajouter les dépendances



Service URL: https://start.spring.io
Name: users-microservice
☑ Use default location
Location: D:\JAVA EE\Spring_Boot_Angular_Udemy_2022\udemy_springSec
Type: Maven Packaging: Jar
Java Version: 17 Language: Java
Group: com.nadhem
Artifact: users-microservice
Version: 0.0.1-SNAPSHOT
Description: Projet Spring Boot pour la génération d'un token JWT à partir d'un login et
Package: com.nadhem.users
Working sets
☐ Add project to working sets
Working sets:



Edit Spring Boot Starters

Service URL: https://start.spring.io

Frequently Used:
- [✓] Lombok
- [✓] MySQL Driver
- [ ] Rest Repositories
- [✓] Spring Boot DevTools
- [✓] Spring Data JPA
- [ ] Spring Security
- [✓] Spring Web
- [✓] Spring Web Services
- [ ] Thymeleaf
- [ ] Validation

Available: Type to search dependencies
- Amazon Web Services
- Developer Tools
- Google Cloud Platform
- I/O
- Messaging
- Microsoft Azure
- NoSQL

Selected:
- X Spring Boot DevTools
- X Lombok
- X Spring Data JPA
- X MySQL Driver
- X Spring Web
- X Spring Web Services

Editer le fichier application.properties

2. Editer le fichier *application.properties*

```
spring.datasource.url=jdbc:mysql://localhost:3306/users_db?createDatabaseIf
NotExist=true&useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
server.servlet.context-path=/users
server.port=8081
spring.main.allow-circular-references=true
```

Créer les entités User et Rôle et leurs interfaces Repository

3. Créer dans le package entities l'entité User :

```
package com.nadhem.users.entities;
import java.util.List;
import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
```

```java
@Data @NoArgsConstructor @AllArgsConstructor
@Entity
public class User {
 @Id
 @GeneratedValue (strategy=GenerationType.IDENTITY)
 private Long user_id;

@Column(unique=true)
 private String username;
 private String password;
 private Boolean enabled;

   @ManyToMany(cascade=CascadeType.ALL, fetch = FetchType.EAGER)
 @JoinTable(name="user_role",joinColumns = @JoinColumn(name="user_id") ,
                  inverseJoinColumns = @JoinColumn(name="role_id"))
 private List<Role> roles;
}
```

4. Créer dans le package entities l'entité Role :

```java
package com.nadhem.users.entities;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data @NoArgsConstructor @AllArgsConstructor
@Entity
public class Role {
 @Id
 @GeneratedValue (strategy=GenerationType.IDENTITY)
 private Long role_id;
 private String role;
}
```

5. Créer dans le package repos, l'interface UserRepository

```java
package com.nadhem.users.repos;

import org.springframework.data.jpa.repository.JpaRepository;
import com.nadhem.users.entities.User;

public interface UserRepository extends JpaRepository<User, Long> {

         User findByUsername(String username);

}
```

6. Créer dans le package repos, l'interface RoleRepository

```java
package com.nadhem.users.repos;

import org.springframework.data.jpa.repository.JpaRepository;
import com.nadhem.users.entities.Role;

public interface RoleRepository extends JpaRepository<Role, Long> {

	Role findByRole(String role);

}
```

Ajouter *Spring Security* et *auth0* au projet

7. Ajouter les dépendances Spring security et JWT au fichier pom.xml :

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

		<dependency>
			 <groupId>com.auth0</groupId>
			 <artifactId>java-jwt</artifactId>
			 <version>3.4.1</version>
		</dependency>
```

Créer la couche service

8. Créer l'interface UserService

```java
package com.nadhem.users.service;

import com.nadhem.users.entities.Role;
import com.nadhem.users.entities.User;

public interface UserService {
	User saveUser(User user);
	User findUserByUsername (String username);
	Role addRole(Role role);
	User addRoleToUser(String username, String rolename);
}
```

9. Créer l'implémentation UserServiceImpl

```java
package com.nadhem.users.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.nadhem.users.entities.Role;
import com.nadhem.users.entities.User;
import com.nadhem.users.repos.RoleRepository;
import com.nadhem.users.repos.UserRepository;

@Transactional
@Service
public class UserServiceImpl  implements UserService{

	@Autowired
	UserRepository userRep;

	@Autowired
	RoleRepository roleRep;


	@Autowired
	BCryptPasswordEncoder bCryptPasswordEncoder;

	@Override
	public User saveUser(User user) {

		user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
		return userRep.save(user);
	}

	@Override
	public User addRoleToUser(String username, String rolename) {
		User usr = userRep.findByUsername(username);
		Role r = roleRep.findByRole(rolename);

		usr.getRoles().add(r);
		return usr;
	}


	@Override
	public Role addRole(Role role) {
		return roleRep.save(role);
	}

	@Override
	public User findUserByUsername(String username) {
		return userRep.findByUsername(username);
	}

}
```

10. Modifier la classe `UsersMicoserviceApplication` pour ajouter les rôles et les utilisateurs

```java
import org.springframework.beans.factory.annotation.Autowired;
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 com.nadhem.users.entities.Role;
import com.nadhem.users.entities.User;
import com.nadhem.users.service.UserService;
import jakarta.annotation.PostConstruct;
```

```
    @Autowired
    UserService userService;

   @PostConstruct
    void init_users() {
          //ajouter les rôles
          userService.addRole(new Role(null,"ADMIN"));
          userService.addRole(new Role(null,"USER"));

          //ajouter les users
          userService.saveUser(new User(null,"admin","123",true,null));
          userService.saveUser(new User(null,"nadhem","123",true,null));
          userService.saveUser(new User(null,"yassine","123",true,null));

          //ajouter les rôles aux users
          userService.addRoleToUser("admin", "ADMIN");
          userService.addRoleToUser("admin", "USER");

          userService.addRoleToUser("nadhem", "USER");
          userService.addRoleToUser("yassine", "USER");
    }


    @Bean
    BCryptPasswordEncoder getBCE() {
          return new BCryptPasswordEncoder();
    }
```

11. Démarrer l'application pour tester l'ajout des utilisateurs et leurs rôles dans la base de données

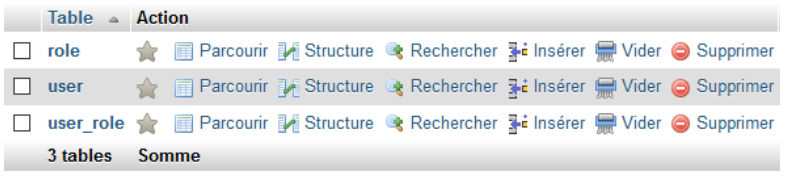
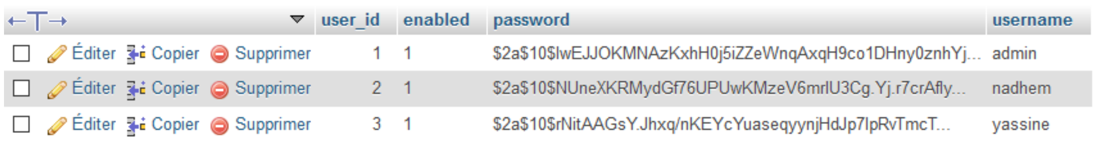


	Table	Action
☐	role	Parcourir Structure Rechercher Insérer Vider Supprimer
☐	user	Parcourir Structure Rechercher Insérer Vider Supprimer
☐	user_role	Parcourir Structure Rechercher Insérer Vider Supprimer
	3 tables	Somme



←T→	user_id	enabled	password	username
☐ Éditer Copier Supprimer	1	1	$2a$10$lwEJJOKMNAzKxhH0j5iZZeWnqAxqH9co1DHny0znhYj...	admin
☐ Éditer Copier Supprimer	2	1	$2a$10$NUneXKRMydGf76UPUwKMzeV6mrlU3Cg.Yj.r7crAfly...	nadhem
☐ Éditer Copier Supprimer	3	1	$2a$10$rNitAAGsY.Jhxq/nKEYcYuaseqyynjHdJp7lpRvTmcT...	yassine

Remarque :

Une fois les utilisateurs et leurs rôles sont enregistrés dans la base de données **commentez** la méthode init_users()

Créer la classe *SecurityConfig*

12. Créer la classe SecurityConfig, placez la dans le package security :

```
package com.nadhem.users.security;

import org.springframework.beans.factory.annotation.Autowired;
```

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {


      @Autowired
      AuthenticationManager authMgr;


      @Bean
      public AuthenticationManager authManager(HttpSecurity http,
                  BCryptPasswordEncoder bCryptPasswordEncoder,
                  UserDetailsService userDetailsService)
        throws Exception {
          return http.getSharedObject(AuthenticationManagerBuilder.class)
            .userDetailsService(userDetailsService)
            .passwordEncoder(bCryptPasswordEncoder)
            .and()
            .build();
      }

       @Bean
     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
                 http.csrf().disable()
      .sessionManagement().
      sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
                                     .authorizeHttpRequests()
                                     .requestMatchers("/login").permitAll()
                                     .anyRequest().authenticated();
              return http.build();
      }
   }
```

Créer la classe *MyUserDetailsService*

13. Créer la classe MyUserDetailsService qui implémente UserDetailsService:

```java
package com.nadhem.users.security;

import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import com.nadhem.users.entities.User;
import com.nadhem.users.service.UserService;



@Service
public class MyUserDetailsService implements UserDetailsService {
   @Autowired
   UserService userService;

@Override
public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
   User user = userService.findUserByUsername(username);

if (user==null)
    throw new UsernameNotFoundException("Utilisateur introuvable !");

   List<GrantedAuthority> auths = new ArrayList<>();

    user.getRoles().forEach(role -> {
          GrantedAuthority auhority = new
SimpleGrantedAuthority(role.getRole());
          auths.add(auhority);
    });

   return new org.springframework.security.core.
                 userdetails.User(user.getUsername(),user.getPassword(),auths);
  }
}
```

Générer le token JWT à la suite d'une authentication

14. Créer la classe `JWTAuthenticationFilter`

```java
package com.nadhem.users.security;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;
import com.fasterxml.jackson.core.JsonParseException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.nadhem.users.entities.User;
```

```
public class JWTAuthenticationFilter extends
UsernamePasswordAuthenticationFilter{


   private AuthenticationManager authenticationManager;



   public JWTAuthenticationFilter(AuthenticationManager authenticationManager)
{
         super();
         this.authenticationManager = authenticationManager;
   }

   @Override
   public Authentication attemptAuthentication(HttpServletRequest request,
HttpServletResponse response)
                throws AuthenticationException {

         User user =null;
         try {
                user = new ObjectMapper().readValue(request.getInputStream(),
User.class);
         } catch (JsonParseException e) {
                e.printStackTrace();
         } catch (JsonMappingException e) {
                e.printStackTrace();
         } catch (IOException e) {
                e.printStackTrace();
         }

         return authenticationManager.
                      authenticate(new
UsernamePasswordAuthenticationToken(user.getUsername(),user.getPassword()));
   }

   @Override
   protected void successfulAuthentication(HttpServletRequest request,
HttpServletResponse response, FilterChain chain,
                Authentication authResult) throws IOException, ServletException
{

         org.springframework.security.core.userdetails.User springUser =
                (org.springframework.security.core.userdetails.User)
authResult.getPrincipal();

         List<String> roles = new ArrayList<>();
         springUser.getAuthorities().forEach(au-> {
                roles.add(au.getAuthority());
         });
                      String jwt = JWT.create().
                        withSubject(springUser.getUsername()).
         withArrayClaim("roles", roles.toArray(new String[roles.size()])).
         withExpiresAt(new Date(System.currentTimeMillis()+10*24*60*60*1000)).
         sign(Algorithm.HMAC256("nadhemb@yahoo.com"));

         response.addHeader("Authorization", jwt);
   }
   }
```

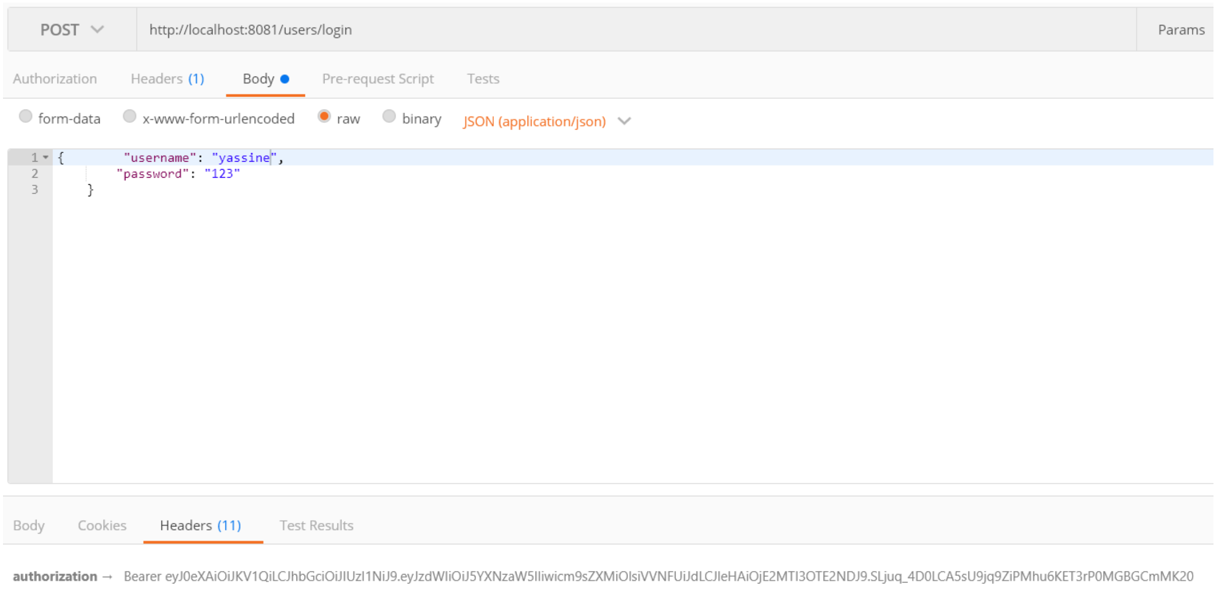
15. Ajouter le filtre JWTAuthenticationFilter à la méthode configure de la classe SecurityConfig :

```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

         •••
   .anyRequest().authenticated().and()
  .addFilterBefore(new
  JWTAuthenticationFilter(authMgr),UsernamePasswordAuthenticationFilter.class)
  ;

          return http.build();
   }
```

16. Tester la génération du JWT avec POSTMAN


POST http://localhost:8081/users/login

Body — raw — JSON (application/json)

```json
{        "username": "yassine",
        "password": "123"
    }
```

Headers (11)

```
authorization → Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ5YXNzaW5lIiwicm9sZXMiOlsiVVNFUiJdLCJleHAiOjE2MTI3OTE2NDJ9.SLjuq_4D0LCA5sU9jq9ZiPMhu6KET3rP0MGBGCmMK20
```

Vérifier votre token JWT sur : https://jwt.io/

17. Créer une interface pour regrouper les constantes

```java
package com.nadhem.users.sercurity;

public interface SecParams {
   public static final long  EXP_TIME = 10*24*60*60*1000;
   public static final String SECRET = "nadhemb@yahoo.com";

}
```