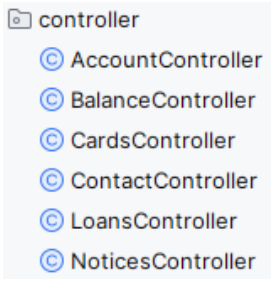
Atelier Spring Security

I. Partie 1.

1. Récupérer le projet SpringSecurity-main.

   Cette version de l'application expose l'ensemble des services sans aucune sécurité. L'objectif est de sécuriser les quatre services critiques avec un mécanisme d'authentification et de laisser les deux service **Cantact** et **Notices** publique sans aucune authentification.



2. Vérifier l'existence de la dépendance suivante dans le fichier pom.xml.

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

3. Exécuter l'application [http://localhost:8080/myCards]. Qui ce que vous remarquez ?
4. Entrer **user** comme login et pour le mot de passe utiliser le mot de passe généré dans la console de projet.
5. Pour changer le login et le mot de passe, modifier le fichier application.properties en ajoutant les deux lignes suivantes, et réexécuter l'application:

```
spring.security.user.name = xproce
spring.security.user.password = 12345
```

6. Pour satisfaire le comportement de la sécurité discuté dans la question 1, on doit tout d'abord explorer les deux mécanismes basic de Spring-Security :
   a. Configuration pour refuser toutes les requêtes.
   b. Configuration pour autoriser toutes les requêtes.
7. Créer un nouveau package "**config**", et créer la classe **ProjectSecurityConfig avec l'annotation @Configuration.**

```
@Configuration
public class ProjectSecurityConfig {… }
```

8. Dans la même classe, créer le bean suivant et tester l'application :

```
@Bean
SecurityFilterChain defaultSecurityFilterChain(  HttpSecurity http) throws Exception {
   //  Configuration to deny all the requests
   http.authorizeHttpRequests(requests -> requests.anyRequest().denyAll())
           .formLogin(Customizer.withDefaults())
           .httpBasic(Customizer.withDefaults());
   return http.build();
           }
```

9. De la meme manière modifier le bean de la question 7 :

```
@Bean
SecurityFilterChain defaultSecurityFilterChain(  HttpSecurity http) throws Exception {
  // Configuration to permit all the requests
        http.authorizeHttpRequests(requests -> requests.anyRequest().permitAll())
          .formLogin(Customizer.withDefaults())
          .httpBasic(Customizer.withDefaults());
      return http.build();
}
```

10. Dans notre cas (pratiquement dans tous des projets), on devra personnaliser cette configuration par défaut :

```
@Bean
SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exception {
   http.csrf((csrf) -> csrf.disable())
           .authorizeHttpRequests((requests)->requests
             .requestMatchers("/myAccount","/myBalance","/myLoans","/myCards").authenticated()
                 .requestMatchers("/notices","/contact","/register").permitAll())
           .formLogin(Customizer.withDefaults())
           .httpBasic(Customizer.withDefaults());
   return http.build();
}
```

II. Partie 2 : Gestion des utilisateurs

La configuration précédente vis a vis des utilisateurs n'est pas pratique, car tous les intervenants vont être obligés à partager les mêmes informations d'authentification. La première solution est de créer un nombre de compte en mémoire pour les différent intervenants, la deuxième méthode est d'utiliser la table **users** dans la base de données, et la troisième solution est d'utiliser une table personnalisée.

1. Créer la classe **UserSecurityConfig** dans le package config :

```
@Configuration
public class UserSecurityConfig { … }
```

2. Pour créer des comptes des utilisateurs dans la mémoire, créer le bean suivant qui utilise la méthode ***withDefaultPasswordEncoder*()** qui encode le mot de passe automatiquement:

```
@Bean
 public InMemoryUserDetailsManager userDetailsService() {
     /*Approach 1 where we use withDefaultPasswordEncoder() method while creating the user details*/
      UserDetails admin = User.withDefaultPasswordEncoder()
            .username("admin")
            .password("12345")
            .authorities("admin")
            .build();
     UserDetails user = User.withDefaultPasswordEncoder()
            .username("user")
            .password("12345")
            .authorities("read")
            .build();
     return new InMemoryUserDetailsManager(admin, user);
}
```

3. Tester l'application.
4. Changer le bean de la question 2 changeant la methode ***withDefaultPasswordEncoder***() avec **withUsername**(String username) :

```
    @Bean
     public InMemoryUserDetailsManager userDetailsService() {/*Approach 2 where we use
    NoOpPasswordEncoder Bean while creating the user details*/
        UserDetails admin = User.withUsername("admin")
              .password("12345")
              .authorities("admin")
              .build();
         UserDetails user = User.withUsername("user")
              .password("12345")
              .authorities("read")
              .build();
        return new InMemoryUserDetailsManager(admin, user);
}
```

5. Avant de tester l'application, ajouter le bean suivant qui va assurer que le mot de passe ne sera pas encodé

```
/* NoOpPasswordEncoder is not recommended for production usage.  Use only for non-prod.
@return PasswordEncoder */
        @Bean
        public PasswordEncoder passwordEncoder() {
                return NoOpPasswordEncoder.getInstance();
        }
```

Cette solution a résolu le premier problème, mais on dépend toujours à un nombre restreint des comptes. Une des meilleures solutions est d'utiliser une base de données pour enregistrer tant de comptes qu'on veut.

6. Créer les tables de la base de données :

```
create database xproce;
use xproce;

CREATE TABLE `users` (
`id` INT NOT NULL AUTO_INCREMENT,
`username` VARCHAR(45) NOT NULL,
`password` VARCHAR(45) NOT NULL,
`enabled` INT NOT NULL,
PRIMARY KEY (`id`));

CREATE TABLE `authorities` (
  `id` int NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `authority` varchar(45) NOT NULL,
  PRIMARY KEY (`id`));

INSERT IGNORE INTO `users` VALUES (NULL, 'xproce', '123456', '1');
INSERT IGNORE INTO `authorities` VALUES (NULL, 'xproce', 'write');

CREATE TABLE `customer` (
  `id` int NOT NULL AUTO_INCREMENT,
  `email` varchar(45) NOT NULL,
  `pwd` varchar(200) NOT NULL,
  `role` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
);

INSERT INTO `customer` (`email`, `pwd`, `role`)
 VALUES ('xproce@example.com', '54321', 'admin');
```

7. Créer une nouvelle classe qui va contenir les différents beans:

```
public class ProjectSecurityConfigMYSQL {
    @Bean
    public UserDetailsService userDetailsService(DataSource dataSource) {
        return new JdbcUserDetailsManager(dataSource);
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }
}
```

8. Vérifier les différentes dépendances : MySQL, JPA, ….

9. Configurer la base de données via le fichier application.properties :

```
spring.datasource.url=jdbc:mysql://localhost/xproce
spring.datasource.username=root
spring.datasource.password=password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

10. Tester l'application [http://localhost:8080/myCards].

Personnalisé les paramètres d'authentication

11. Dans le package config, créer la classe BankUserDetails avec l'annotation @Service qui implémente l'interface UserDetailsService.

```
@Service
public class BankUserDetails implements UserDetailsService {…}
```

12. Créer l'entité Customer représentant la table `customer` de la question 6.
13. Créer le repository CustomerRepository.
14. Le code final de la classe est :

```
@Service
public class BankUserDetails implements UserDetailsService {

    @Autowired
    private CustomerRepository customerRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        String userName, password;
        List<GrantedAuthority> authorities;
        List<Customer> customer = customerRepository.findByEmail(username);
        if (customer.size() == 0) {
            throw new UsernameNotFoundException("User details not found for the user : " + username);
        } else {
            userName = customer.get(0).getEmail();
            password = customer.get(0).getPwd();
            authorities = new ArrayList<>();
            authorities.add(new SimpleGrantedAuthority(customer.get(0).getRole()));
        }
        return new User(userName, password, authorities);
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }
}
```

15. Tester l'application [http://localhost:8080/myCards].