

TRAVAUX PRATIQUES POUR LES SÉANCES

DU 23/02/2025 => 09/03/2025

A. SPRING SECURITY. AUTHENTIFICATION DES UTILISATEURS IN MEMORY

1. AJOUTER LE STARTER SPRING SECURITY AU POM.XML

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

2. DÉMARRER L'APPLICATION ET VÉRIFIER LA GÉNÉRATION DU MOT DE PASSE

```
2025-03-08 14:26:22,743 WARN [main] o.s.b.a.o.j.JpaBaseConfiguration$JpaWebConfiguration: spring
2025-03-08 14:26:23,373 WARN [main] o.s.b.a.s.s.UserDetailsServiceAutoConfiguration:
|
Using generated security password: 5ea01b8a-fe3f-4d9b-b3c2-018bfab2cb3f

This generated password is for development use only. Your security configuration must be updated
```

3. ENTRER AU SWAGGER ET VÉRIFIER QUE SPRING SECURITY DANS UNE PAGE LOGIN PAR DÉFAUT.



ENTRE AVEC LE MOT DE PASSE GÉNÉRÉ ET LE COMPTE "USER" EN MINISCULE.

4. DANS LE PACKAGE CONFIG CRÉER LA CLASSE "SECURITYCONFIG". CONFIGURER LES UTILISATEURS IN MEMORY DANS UN PREMIER TEMPS. LE BUT EST DE CRÉER UN BEAN USERDETAILSSERVICE DE SPRING SECURITY.

```
package ma.gov.pfe.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.configurat
ion.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import
```

```

org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.provisioning.InMemoryUserDetails
Manager;

import java.util.Arrays;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    UserDetailsService userDetailsService() {
        UserDetails user1 = User.builder()
            .password("123")
            .username("user1")
            .roles("ADMIN")
            .build();
        UserDetails user2 = User.builder()
            .password("123")
            .username("user2")
            .roles("USER")
            .build();

        return new
InMemoryUserDetailsManager(Arrays.asList(user1,user2));
    }
}

```

5. SPRING SECURITY EXIGE QUE LES MOTS DE PASSE SOIENT ENCODÉS. CRÉER UN BEAN PASSWORDENCODER DANS UNE CLASSE GLOBALCONFIG À CRÉER.

```

package ma.gov.pfe.config;

import ma.gov.pfe.validation.Password;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

```

```
public class GlobalSecurity {

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

6. INJECTOR LE BEAN PASSWORDENCODER AU NIVEAU DE LA CLASSE SECURITYCONFIG.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    // 1 usage
    private final PasswordEncoder passwordEncoder;

    public SecurityConfig(PasswordEncoder passwordEncoder) {
        this.passwordEncoder = passwordEncoder;
    }
}
```

7. APPLIQUER L'ENCODAGE AUX MOTS DE PASSES DES UTILISATEURS USER1 ET USER2 DÉJÀ CRÉÉES DANS LE BEAN USERDETAILSSERVICE.

```
@Bean
UserService userDetailsService() {
    UserDetails user1 = User.builder()
        .password(passwordEncoder.encode(rawPassword: "123"))
        .username("user1")
        .roles("ADMIN")
        .build();
    UserDetails user2 = User.builder()
        .password(passwordEncoder.encode(rawPassword: "123"))
        .username("user2")
}
```

8. DÉMARRER L'APPLICATION ET UTILISER LES COMPTES USER1 ET USER2 POUR S'AUTHTENTIFIER


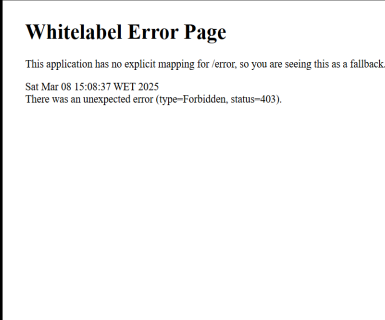
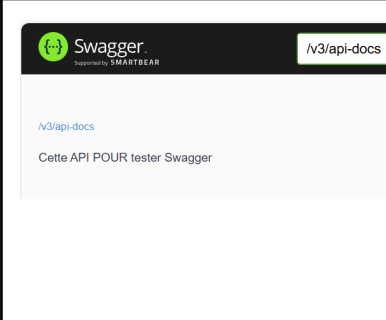
B. SPRING SECURITY. AUTHORIZATIONS

9. MAINTENANT NOUS SOUHAITONS AUTORISER SEULEMENT LES ADMINS À ENTRER AU SWAGGER DE NOTRE APPLICATION.

DANS LA CLASSE SECURITYCONFIG, CRÉER LE BEAN DE SPRING SECURITY.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
    return http.authorizeRequests()
        .antMatchers("/swagger-ui/**", "/v3/api-docs/**",
"/swagger-ui.html").hasRole("ADMIN")
        .anyRequest().authenticated()
        .and()
        .formLogin()
        .permitAll()
        .defaultSuccessUrl("/swagger-ui/index.html")
        .and()
        .build();
}
```

10. ENTRER AU SWAGGER AVEC LES DIFFÉRENTS COMPTES SUIVANTS ET VÉRIFIER LE RÉSULTAT.

Avec un user différent de user1 et user 2	Avec le compte user2 qui a le profil USER	Avec le compte user1 qui a le profil ADMIN
		

C. SPRING SECURITY. FILTERS

NOUS SOUHAITONS CRÉER DANS NOTRE APPLICATION DEUX FILTRES. UN POUR TRACER, DANS LES LOGS, LES REQUÊTES HTTPS REÇUES. UN DEUXIÈME POUR VÉRIFIER QUE LE MOT DE PASSE EST TOUJOURS PRÉSENT DANS LA REQUÊTE HTTP.

CRÉER UN PACKAGE "FILTRES" POUR TOUS LES FILTRES DE NOTRE API.

CRÉER LE PREMIER FILTER LOGGINGFILTER COMME SUIVANT.

```
package ma.gov.pfe.filters;
```

```

import lombok.extern.slf4j.Slf4j;
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
import java.util.Enumeration;

@WebFilter("/*")
@Slf4j
public class LoggingFilter implements Filter {
    @Override
    public void doFilter(ServletRequest request,
        ServletResponse response, FilterChain filterChain) throws
        IOException, ServletException {
        loggingRequest((HttpServletRequest) request);
        filterChain.doFilter(request, response);
    }

    private void loggingRequest(HttpServletRequest request) {
        log.debug("** URI : {} ", request.getRequestURI());
        log.debug("** Method: {} ", request.getMethod());
        loggingHeaders(request);
    }

    private void loggingHeaders(HttpServletRequest request) {
        Enumeration<String> names = request.getHeaderNames();
        while (names.hasMoreElements()) {
            String name = names.nextElement();
            String value = request.getHeader(name);
            log.debug("** Header {} value {} ", name, value);
        }
    }
}

```

11. DANS LA CLASSE SECURITY CONFIG, DANS LE BEAN SECURITY FILTERCHAIN. PLACER LE
 FILTRES LOGGINGFILTER AVANT LE FILTRE DE SPRING SECURITY
 USERNAMEPASSWORDAUTHENTICATIONFILTER

```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http
        .addFilterBefore(new LoggingFilter(), UsernamePasswordAuthenticationFilter.class)

```

12. LANCER L'APPLICATION ET ACCÉDER AU SWAGGER PUIS VÉRIFIER QUE LE FILTRE FONCTIONNE ET QUE LES TRACES SONT PRÉSENTES DANS LES LOGS.

```
1-exec-2] m.g.p.LoggingFilter: ** URI : /login
1-exec-2] m.g.p.LoggingFilter: ** URI : GET
1-exec-2] m.g.p.LoggingFilter: ** Header host value 127.0.0.1:9091
1-exec-2] m.g.p.LoggingFilter: ** Header connection value keep-alive
1-exec-2] m.g.p.LoggingFilter: ** Header upgrade-insecure-requests value 1
1-exec-2] m.g.p.LoggingFilter: ** Header user-agent value Mozilla/5.0 (Windows
1-exec-2] m.g.p.LoggingFilter: ** Header accept value text/html,application/xhtml
1-exec-2] m.g.p.LoggingFilter: ** Header sec-fetch-site value none
1-exec-2] m.g.p.LoggingFilter: ** Header sec-fetch-mode value navigate
```

Act

13. CRÉER LE DEUXIÈME FILTRE PASSWORD FILTER TOUJOURS DANS LE PACKAGE "FILTERS". SI LE MOT DE PASSE N'EST PAS PRÉSENT, ARRÊTE LA CHAÎNE DES FILTRES. SI LE MOT DE PASSE EST PRÉSENT ENCODER LE VIA BCrypt 2 OU 3 FOIS.

```
package ma.gov.pfe.filters;

import lombok.extern.slf4j.Slf4j;
import org.springframework.security.crypto.password.PasswordEncoder;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;

@WebFilter
@Slf4j
public class PasswordFilter implements Filter {
    private final PasswordEncoder passwordEncoder;

    public PasswordFilter(PasswordEncoder passwordEncoder) {
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public void doFilter(ServletRequest request,
        ServletResponse response, FilterChain filterChain) throws
        IOException, ServletException {
```

```

        String password = ((HttpServletRequest)
request).getParameter("password");
        if (password != null) {
            log.debug("password encode 1 : {} ",
passwordEncoder.encode(password));
            log.debug("password encode 2 : {} ",
passwordEncoder.encode(password));
            filterChain.doFilter(request, response);
        } else {
            log.error("Password missed. The filter chain is
stopped !! ");
        }
    }
}

```

14. PLACER CE FILTRE APRÈS LOGGINGFILTER, TOUJOURS DANS LE BEAN SECURITYFILTERCHAIN CONFIGURÉ AU NIVEAU DE LA CLASSE SECURITY CONFIG.

```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http
        .addFilterBefore(new LoggingFilter(), UsernamePasswordAuthenticationFilter.class)
        .addFilterAfter(new PasswordFilter(passwordEncoder), LoggingFilter.class)
}

```

15. LANCER L'APPLICATION ET VÉRIFIER DANS LES LOGS QUE LES DEUX FILTRES S'EXÉCUTENT.

```

m.g.p.LoggingFilter: ** Header sec-fetch-dest value document
m.g.p.LoggingFilter: ** Header accept-encoding value gzip, deflate, br, zstd
m.g.p.LoggingFilter: ** Header accept-language value fr,fr-FR;q=0.9,en;q=0.8,en-GB;q=0.7
m.g.p.LoggingFilter: ** Header cookie value JSESSIONID=A8DF6E05960C0AF17B313DD62D935FF0
m.g.p.f.PasswordFilter: Password missed. The filter chain is stopped !!

```

16. MAINTENANT, DANS LA CLASSE PASSWORD FILTER, LAISSER PASSER LA CHAÎNE MÊME SI LE MOT DE PASSE EST VIDE POUR NE PAS BLOQUER LE CHARGEMENT DU SWAGGER.

```

    if (password != null) {
        log.debug("password encode 1 : {} ", passwordEncoder.encode(password));
        log.debug("password encode 2 : {} ", passwordEncoder.encode(password));
        filterChain.doFilter(request, response);
    } else {
        log.error("Password missed. The filter chain is stopped !!");
        filterChain.doFilter(request, response);
    }
}

```

17. LANCER L'APPLICATION, S'AUTHTIFIER ET VÉRIFIER DANS LES LOGS QUE LE BCRYPT DANS DES HASH DIFFÉRENTS POUR LE MÊME MOT DE PASSE.

```
PasswordFilter: password encode 1 : $2a$10$ICWlylm/Y.FiDXwTL4Vml.pG/rSwLn0UZ169CwgCoIKAV05gyjoCu
PasswordFilter: password encode 2 : $2a$10$Jy0eT7qA0SwodXoWgcwt0.2xyPPiy1uz5iXLAytv/nxRSADB73cAy
```

D. SPRING SECURITY. INTERCEPTEURS

LES FILTRES SONT GÉNÉRALEMENT CRÉÉS DANS LE CONTEXTE DES APPLICATIONS WEB ET DES REQUÊTES HTTP POUR FILTRER AVANT D'ARRIVER AU CONTRÔLEUR PRINCIPAL (SERVLET DISPATCHER POUR SPRING MVC). TOUTEFOIS, IL EST POSSIBLE DE FAIRE DES TRAITEMENTS POST/PRE ACTION MÊME APRÈS AVOIR ATTEINT LE CONTRÔLEUR PRINCIPAL EN CRÉANT DES INTERCEPTEURS.

18. CRÉER UN PACKAGE INTERCEPTORS, CRÉER UNE CLASSE URL INTERCEPTOR

```
package ma.gov.pfe.interceptors;

import lombok.extern.slf4j.Slf4j;
import org.springframework.lang.Nullable;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@Slf4j
public class UrlInterceptor implements HandlerInterceptor {
    public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws
        Exception {
        log.debug("Process Pre-action");
        return true;
    }

    public void postHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler, @Nullable
        ModelAndView modelAndView) throws Exception {
        log.debug("Process Post-action");
    }
}
```



```

    public void afterCompletion(HttpServletRequest request,
        HttpServletResponse response, Object handler, @Nullable
        Exception ex) throws Exception {
        log.debug("Process after-completion");
    }
}

```

19. ENREGISTRER CET INTERCEPTEUR DANS LE REGISTRE DES INTERCEPTEURS, EN CRÉANT LA CONFIGURATION SUIVANTE.

```

package ma.gov.pfe.config;

import ma.gov.pfe.interceptors.UrlInterceptor;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.Interceptor
Registry;
import org.springframework.web.servlet.config.annotation.WebMvcConfi
gurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {
    public void addInterceptors(InterceptorRegistry registry)
    {
        registry.addInterceptor(new UrlInterceptor());
    }
}

```

20. LANCER L'APPLICATION ET VÉRIFIER DANS LES LOGS QUE LES MÉTHODES DE L'INTERCEPTEUR S'EXÉCUTENT.

```

ERROR [http-nio-9091-exec-6] m.g.p.f.PasswordFilter: Password missed. The filter
DEBUG [http-nio-9091-exec-8] m.g.p.i.UrlInterceptor: Process Pre-action
DEBUG [http-nio-9091-exec-9] m.g.p.i.UrlInterceptor: Process Pre-action
DEBUG [http-nio-9091-exec-7] m.g.p.i.UrlInterceptor: Process Pre-action
DEBUG [http-nio-9091-exec-6] m.g.p.i.UrlInterceptor: Process Pre-action
DEBUG [http-nio-9091-exec-5] m.g.p.i.UrlInterceptor: Process Post-action
DEBUG [http-nio-9091-exec-5] m.g.p.i.UrlInterceptor: Process after-completion

```

E. SPRING SECURITY. AUTHENTIFICATION DES UTILISATEURS IN DATABASE

21. SUPPOSONS QU'UN UTILISATEUR PEUT AVOIR PLUSIEURS ROLES. CRÉER L'ENTITY ROLEENTITY ET L'ASSOCIER AVEC USERENTITY COMME SUIVANT.

ROLEENTITY

```
package ma.gov.pfe.entities;

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

@Entity
@Table(name = "T_ROLES")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class RoleEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String label;
}
```

USERENTITY

```
package ma.gov.pfe.entities;

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

import javax.persistence.*;
import javax.validation.constraints.Email;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.util.Set;

@Entity
@Table(name = "T_USERS")
@Data
@AllArgsConstructor
@NoArgsConstructor
```

```

public class UserEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull(message = "Entity Name cannot be null")
    @Size(min = 2, max = 100, message = "Entity Name must be
between 2 and 100 characters")
    private String name;

    @NotNull(message = "Entity Address cannot be null")
    @Size(max = 255, message = "Entity Address can be a
maximum of 255 characters")
    private String address;

    @NotNull(message = "Entity Email cannot be null")
    @Email(message = "Entity Email should be valid")
    @Column(unique = true, nullable = false)
    private String email;
    private String password;

    @ManyToMany(fetch = FetchType.EAGER, cascade =
{CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH,
CascadeType.DETACH})
    private Set<RoleEntity> roles;
}

```

22. DANS L'INTERFACE `USERREPOSITORY`, DÉCLAREZ LA MÉTHODE SUIVANTE POUR RECHERCHER UN UTILISATEUR PAR SON EMAIL.

```
UserEntity findByEmail(String email);
```

23. DANS LE PACKAGE `DATA` À CRÉER, AJOUTER DES UTILISATEURS DE TEST AU NIVEAU DE LA BASE, VIA LE COMPOSANT `COMMAND LINE RUNNER` SUIVANT.

```

package ma.gov.pfe.data;

import ma.gov.pfe.entities.RoleEntity;
import ma.gov.pfe.entities.UserEntity;
import ma.gov.pfe.repositories.UserRepository;
import org.springframework.boot.CommandLineRunner;
import
org.springframework.security.crypto.password.PasswordEncoder;

```

```

import org.springframework.stereotype.Component;

import java.util.HashSet;
import java.util.Set;

@Component
public class UsersDump implements CommandLineRunner {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    public UsersDump(UserRepository userRepository,
PasswordEncoder passwordEncoder) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public void run(String... args) throws Exception {
        Set<RoleEntity> roles = new HashSet() {
            {
                add(new RoleEntity("ADMIN"));
                add(new RoleEntity("USER"));
            }
        };

        String user = "user" + 1;
        UserEntity userEntity = UserEntity.builder()
            .name(user)
            .email(user + "@gmail.com")
            .password(passwordEncoder.encode(user))
            .address(user)
            .roles(roles)
            .build();
        userRepository.save(userEntity);
    }
}

```

24. CHANGER LA CLASSE `USERServiceImpl` POUR IMPLÉMENTER AUSSI L'INTERFACE `USERDetailsService` DE SPRING SECURITY.

```

package ma.gov.pfe.services;

```

```

import ma.gov.pfe.dtos.UserDto;
import ma.gov.pfe.entities.UserEntity;
import ma.gov.pfe.mappers.UserMapper;
import ma.gov.pfe.repositories.UserRepository;
import org.springframework.security.core.userdetails.User;
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 javax.transaction.Transactional;
import java.util.List;
import java.util.stream.Collectors;

@Transactional
@Service
public class UserServiceImpl implements IUserService,
UserDetailsService {

    private final UserRepository userRepository;
    private final UserMapper userMapper;

    public UserServiceImpl(
        UserRepository userRepository,
        UserMapper userMapper) {
        this.userMapper = userMapper;
        this.userRepository = userRepository;
    }

    @Override
    public UserDto add(UserDto userDto) {
        return
userMapper.toDto(userRepository.save(userMapper.toEntity(user
Dto)));
    }

    @Override
    public UserDto modify(UserDto userDto) {

```

```

        return
userMapper.toDto(userRepository.save(userMapper.toEntity(user
Dto)));
    }

    @Override
    public void delete(Long id) {
        userRepository.deleteById(id);
    }

    @Override
    public List<UserDto> getAll() {
        return
userRepository.findAll().stream().map(userEntity ->
userMapper.toDto(userEntity)).collect(Collectors.toList());
    }

    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        UserEntity userEntity =
userRepository.findByEmail(email);
        if (userEntity == null)
            throw new UsernameNotFoundException("User not
found ! ");
        UserDetails user = User.builder()
            .password(userEntity.getPassword())
            .username(userEntity.getEmail())

            .roles(userEntity.getRoles().stream().map(roleEntity ->
roleEntity.getLabel()).toArray(String[]::new))
            .build();
        return user;
    }
}

```

25. CHANGER LA CONFIG DANS LA CLASSE SECURITYCONFIG, NE PLUS UTILISER IN MEMORY STRATEGY. ET LA REMPLACER PAR L'INJECTION DE DEPENDENCE VIS A VIS DE USERDETAILSERVICE. ET AUSSI LA CRÉATION DU BEAN AUTHENTICATIONMANAGER

```

package ma.gov.pfe.config;

```

```

import ma.gov.pfe.LoggingFilter;
import ma.gov.pfe.filters.PasswordFilter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.AuthenticationMan
ager;
import
org.springframework.security.config.annotation.authentication
.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.H
ttpSecurity;
import
org.springframework.security.config.annotation.web.configurat
ion.EnableWebSecurity;
import
org.springframework.security.core.userdetails.UserDetailsServ
ice;
import
org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePassw
ordAuthenticationFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    private final PasswordEncoder passwordEncoder;
    private final UserDetailsService userDetailsService;

    public SecurityConfig(PasswordEncoder passwordEncoder,
UserDetailsService userDetailsService) {
        this.passwordEncoder = passwordEncoder;
        this.userDetailsService = userDetailsService;
    }

    @Bean
    public AuthenticationManager
authenticationManager(HttpSecurity httpSecurity) throws
Exception {

```

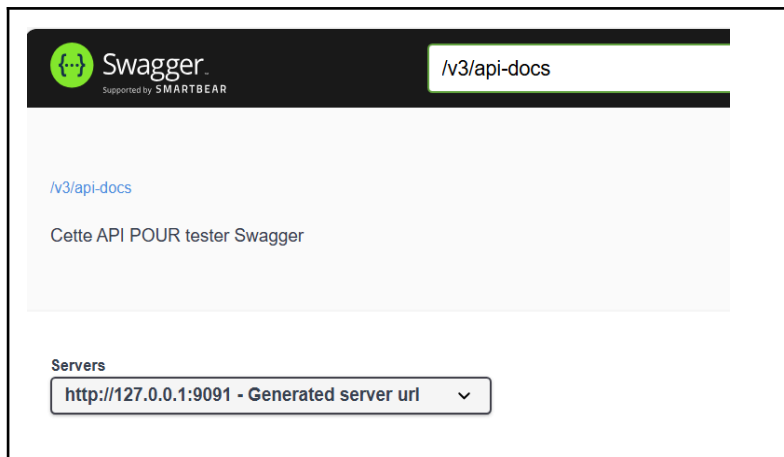
```

        AuthenticationManagerBuilder builder =
httpSecurity.getSharedObject(AuthenticationManagerBuilder.class);
        return builder
            .userDetailsService(userDetailsService)
            .passwordEncoder(passwordEncoder)
            .and()
            .build();
    }

    @Bean
    public SecurityFilterChain
securityFilterChain(HttpSecurity http) throws Exception {
        return http
            .addFilterBefore(new LoggingFilter(),
UsernamePasswordAuthenticationFilter.class)
            .addFilterAfter(new
PasswordFilter(passwordEncoder), LoggingFilter.class)
            .authorizeRequests()
            .antMatchers("/swagger-ui/**",
"/v3/api-docs/**", "/swagger-ui.html").hasRole("ADMIN")
            .anyRequest().authenticated()
            .and()
            .formLogin()
            .permitAll()
            .defaultSuccessUrl("/swagger-ui/index.html")
            .and()
            .build();
    }
}

```

26. LANCER L'APPLICATION, ESSAYER LE COMPTE USER1@GMAIL.COM ET LE MOT DE PASSE USER1 EN MINUSCULE. VOUS DEVEZ ENTRER AU SWAGGER AVEC CE COMPTE.



27. ESSAYER LE COMPTE USER2@GMAIL.COM ET LE MOT DE PASSE USER2 EN MINUSCULE. VOUS NE DEVEZ PAS ENTRER AU SWAGGER AVEC CE COMPTE.

Please sign in

Les identifications sont erronées

Sign in

F. METHOD REFERENCE.

28. SOIT LA CLASSE SUIVANTE CALCUL SUIVANTE:

```
package ma.gov.pfe;

public class Calcul {
    private int cof;

    public Calcul(int cof) {
        this.cof = cof;
    }

    public int add (int a,int b){
        return cof* (a+b);
    }

    public static int mult (int a,int b){
        return a+b;
    }
}
```

```
}  
}
```

29. SOIT LES DEUX INTERFACE FONCTIONNELLES SUIVANTES:

```
interface I1 {  
    int compute(int a, int b);  
}  
  
interface I2 {  
    Calcul compute(int a);  
}
```

30. LA TRANSFORMATION DE CES DEUX INTERFACE EN EXPRESSION LAMBDA DONNERA

```
public class Main {  
    public static void main(String[] args) {  
        Calcul calcul = new Calcul(10);  
        I1 i1 = (a, b) -> calcul.add(a, b);  
        I1 i2 = (a, b) -> Calcul.mult(a, b);  
        I2 i3 = a -> new Calcul(a);  
    }  
}
```

31. IL EST RECOMMANDÉ DE CHANGER LES FLÈCHES PAR :: (METHOD REFERENCE)

```
public class Main {  
    public static void main(String[] args) {  
        Calcul calcul = new Calcul(10);  
        I1 i1 = calcul::add;  
        I1 i2 = Calcul::mult;  
        I2 i3 = Calcul::new;  
    }  
}
```