1. Extension personnalisée my-security

L'extension **my-security** est une extension personnalisée pour **Spring-security** qui permet de simplifier les paramétrages les plus courants en s'appuyant sur de l'**auto-configuration** et un fichier **application-withSecurity.properties** simple à configurer.

1.1. Code source de cette extension

Référentiel git "https://github.com/didier-mycontrib/spring5plus"

*et tous les projets de type "**my-security-...**"*
Exemples d'utilisations "appliSpringWeb4 (standalone jwt)" et "appliSpringWeb5 (oauth2)"

1.2. Dépendances "maven" (projets "my-security-...")

projet	*contenu*	*utilisation*
my-security-common	@Primary (Bcrypt)PasswordEncoder et interfaces pour extensions	Dépendance indirecte/transitive. Utilisé en autre par my-security-configure et my-security-realm-configure
my-security-configure	Projet principal , configuration des "SecurityFilterChain"	Projet fondamental , toujours nécessaire
my-security-realm-configure	Configuration de certains "realm" (InMemory or Jdbc or ...)	Pas nécessaire en mode OAuth2/OIDC
my-security-standalone-jwt	Classes java pour une authentification d'api REST en mode "jwt bearer token" (filtre + WS de "login")	Nécessaire que pour sécuriser une api REST de manière autonome (sans délégation d'authentification OAuth2)
my-security-std-jwt-configure	Auto-configuration pour "standalone-jwt"	Nécessaire que pour sécuriser une api REST de manière autonome (sans délégation d'authentification OAuth2)
my-security-starter	Simple paquet regroupant tous les projets ci-dessus	A n'utiliser que si besoin de "standalone-jwt" .
...		
my-aggregate	*Simple projet utilitaire maven pour construire d'un coup tous les projets "my-..."*	*mvn **clean install** en mode -DskipTests*
...		

1.3. Utilisation type en mode standalone-jwt

```
<dependency>
    <groupId>org.mycontrib</groupId>
    <artifactId>my-security-starter</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</dependency>
```

et

src/main/resources/**application-withSecurity.properties**

```
#configurations specifiques pour profil "withSecurity"
#analysees via les classes de config du projet my-security-configure (pour partie mysecurity.area)
# et les classes de config du projet my-security-realm-configure (pour partie mysecurity.realm)

#mysecurity.area.rest.whitelist (permitAll GET/POST/PUT/DELETE/...):
mysecurity.area.rest.whitelist=/rest/api-news/news/**
#mysecurity.area.rest.whitelist=/rest/xxx-api/**;/rest/yyy-api/**

mysecurity.area.tools.whitelist=/h2-console/**;/swagger-ui/**;/v3/api-docs

#mysecurity.area.rest.readonlylist (permitAll GET only):
mysecurity.area.rest.readonlylist=/rest/api-devise/devise/**
#mysecurity.area.rest.readonlylist=/rest/api-devise/**;/rest/api-bank/customer/**;/rest/yyy-api/**

#mysecurity.area.rest.protectedlist (authentication is required):
mysecurity.area.rest.protectedlist=/rest/api-bank/customer/**;/rest/api-bank/compte/**

#mysecurity.area.site.whitelist=/site/calcul/**
mysecurity.area.site.protectedlist=/site/calcul/**

#mysecurity.chain.rest-auth-type=OAuth2ResourceServer(by default) or StandaloneJwt
mysecurity.chain.rest-auth-type=StandaloneJwt
```

src/main/resources/**application-withSecurity.properties (suite)**

```
#mysecurity.realm.with-global-default-secondary-in-memory-realm=true

mysecurity.realm.global.jdbc-realm.driverClassName=org.h2.Driver
mysecurity.realm.global.jdbc-realm.url=jdbc:h2:~/realmdb5
mysecurity.realm.global.jdbc-realm.username=sa
mysecurity.realm.global.jdbc-realm.password=

logging.level.org.mycontrib.mysecurity=DEBUG
```

1.4. Variante d'utilisation type en mode oauth2

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

<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>

<dependency>
        <groupId>org.mycontrib</groupId>
        <artifactId>my-security-configure</artifactId>
        <version>0.0.1-SNAPSHOT</version>
</dependency>

<dependency> < !-- optional , if "site part" aside "api_rest" -->
        <groupId>org.mycontrib</groupId>
        <artifactId>my-security-realm-configure</artifactId>
        <version>0.0.1-SNAPSHOT</version>
</dependency>
```

et

src/main/resources/**application-withSecurity.properties**

```
mysecurity.area.rest.whitelist=/rest/api-news/news/**
mysecurity.area.tools.whitelist=/h2-console/**;/swagger-ui/**;/v3/api-docs
mysecurity.area.rest.readonlylist=/rest/api-devise/devise/**
mysecurity.area.rest.protectedlist=/rest/api-bank/customer/**;/rest/api-bank/compte/**

#mysecurity.chain.rest-auth-type=OAuth2ResourceServer(by default) or StandaloneJwt
mysecurity.chain.rest-auth-type=OAuth2ResourceServer

#### VERY IMPORTANT CONFIG ####
spring.security.oauth2.resourceserver.jwt.issuer-uri=https://www.d-defrance.fr/keycloak/realms/sandboxrealm
```

1.5. Conventions d'URL à idéalement respecter

Partie API(s) REST	**/rest/**api-xxx/yyy (ou **/rest/**xxx-api/yyy)
Partie Site Web SpringMVC (@Controller + JSP ou Thymeleaf)	**/site/**xxx , **/site/login , ...**
Autres parties	/**

1.6. Paramétrages des zones à protéger

mysecurity.area.xxx.yyy=aaa;bbb;ccc

...area.**rest**	Api rest
...area.**site**	Eventuelle partie site web SpringMvc (@Controller + JSP ou Thymeleaf)
...area.**tools**	Outils utiliaires (ex : h2-console , swagger, ...)
...area.**other**	Autres parties (pages statiques html/css/js avec images .jpg, .gif, .svg, ...)

...**blacklist**	Accès temporairement interdit **.denyAll()**
...**whitelist**	Accès libre / public **.permitAll()**
...**readonlylist**	Accès libre en mode lecture **.permitAll() sur mode .GET seulement**
...**protected**	Accès autorisé que si utilisateur authentifié **.authenticated()**

NB1: Tout chemin de la partie .readonlylist sera automatiquement reporté dans la partie .protected (pour les autres modes POST/PUT/DELETE/...)

*NB2 : Soit **.whitelist** (ou .readonlylist) et pas de @PreAuthorise() sur @RestController*

*Soit **.protected** et **éventuel complément @PreAuthorise() sur @RestController***

NB3:les chemins correspondants aux constantes suivantes de l'interface org.mycontrib.mysecurity.common.extension.MySecurityExtension sont automatiquement accessibles (en mode .permitAll())

```
public static final String DEFAULT_REST_STANDALONE_LOGIN_PATH="/rest/api-login/public/login";
public static final String DEFAULT_SITE_FORM_LOGIN_URI="/site/login";
public static final String DEFAULT_SITE_FORM_LOGOUT_URI="/site/logout";
```

*et la constante suivante correspond à la valeur par défaut de **mysecurity.area.other.whitelist***

```
public static final String[] DEFAULT_STATIC_WHITELIST = { "/", "/favicon.ico", "/**/*.png", "/**/*.gif",
                                                          "/**/*.svg", "/**/*.jpg","/**/*.html", "/**/*.css", "/**/*.js" };
```

NB4 :

Dans certaines situations rares (ex : expérimentations temporaires) , on peut

redéfinir le sous-configurateur paramétrant les zones à protéger (mais en faisant cela , on perd une grande partie de la valeur ajoutée de my-security-configure et tous les paramétrages en "mysecurity.area..." du fichier application-withSecurity.properties seront ignorés).

```
package org.mycontrib.appliSpringWeb;

import org.mycontrib.mysecurity.common.extension.WithSecurityFilterChainSubConfig;
import org.springframework.context.annotation.Profile;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.stereotype.Component;

//@Component //NB: this extension in currently desactivated !!!
@Profile("withSecurity")
public class SpecificWithSecurityMainFilterChainSubConfig implements WithSecurityFilterChainSubConfig {

	//NB: en phase de test , penser éventuellement à supprimer le cookie JSESSIONID
	// et/ou le "authToken" dans sessionStorage du navigateur

	@Override
	public HttpSecurity prepareDefaultOtherWebPartFilterChain(HttpSecurity http) throws Exception {
		http=http.authorizeRequests(
				authorizeRequests -> authorizeRequests
					.antMatchers("/", "/favicon.ico", "/**/*.png", "/**/*.gif", "/**/*.svg",
							"/**/*.jpg","/**/*.html", "/**/*.css", "/**/*.js").permitAll()
					.antMatchers( "/**").authenticated() //by default
				)
					.csrf().disable();
		return http;
	}

	/*
	//METHODES with DEFAULT implementations:

	public  HttpSecurity prepareRestApiFilterChain(HttpSecurity http) throws Exception {
		http.authorizeRequests().antMatchers( "/rest/**").authenticated();
		return http;
	}


	public  HttpSecurity prepareSpringMvcSiteFilterChain(HttpSecurity http) throws Exception {
		http.authorizeRequests().antMatchers( "/site/**").authenticated();
		return http;
	}

	public boolean cancelRestConfigPostAction() { return false; }

	public AuthenticationManager provideSpecificRealmAuthenticationManager(HttpSecurity http,
				RealmPurposeEnum realmPurpose) {    return null; }

	*/
}
```

1.7. Paramétrages du Realm / AuthenticationManager

Dans le cas (très fréquent en mode "production") **d'une authentification déléguée à un seveur d'authorisations "OAuth2/Oidc"** tel que "keycloak" , "azure directory" , "amazon cognito" , "okta" ou autres , **les seules configurations absoluments nécessaires sont les suivantes** :

```
#mysecurity.chain.rest-auth-type=OAuth2ResourceServer(by default) or StandaloneJwt
mysecurity.chain.rest-auth-type=OAuth2ResourceServer

spring.security.oauth2.resourceserver.jwt.issuer-uri=https://www.d-defrance.fr/keycloak/realms/sandboxrealm
```

Dans le cas contraire (serveur autonome sans délégation d'authentification) , le AuthenticationManager qui sera automatiquement construit et utilisé obéit aux règles suivantes :

- **plusieurs modes possibles (UserDetails , jdbc , InMemory) soit en mode "exclusif" , soit en mode complémentaire/hierarchique.**
- **Le "UserDetails" spécifique est prioritaire sur "jdbc" qui est lui même prioritaire sur "InMemory"**
- **plusieurs variantes peuvent cohabiter ("rest" pour api-rest standalone , "site" pour partie "site springMvc avec .jsp ou thymeleaf , "global" pour un même "realm" utilisé par les 2 parties "rest et "site")**

1.8. InMemory Realm

En phase de développement , le mode "InMemory" peut être activé via la propriété suivante :

```
mysecurity.realm.with-global-default-secondary-in-memory-realm=true
```

Et sans autre configuration complémentaire, les utilisateurs auto-configurés par défaut sont les suivants :

```
.withUser("user1").password(passwordEncoder.encode("pwd1")).roles("USER")
.and().withUser("admin1").password(passwordEncoder.encode("pwd1")).roles("ADMIN")
.and().withUser("user2").password(passwordEncoder.encode("pwd2")).roles("USER")
.and().withUser("admin2").password(passwordEncoder.encode("pwd2")).roles("ADMIN")
```

De manière à redéfinir/personnaliser la liste des utilisateurs préconfigurés en mode "InMemory" (ou "jdbc") on pourra coder un composant de ce type dans le projet "appli springBoot" :

```
package org.mycontrib.appliSpringWeb;

import org.mycontrib.mysecurity.common.extension.MySecurityDefaultUsersSimpleConfigurer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Profile;
import org.springframework.security.config.annotation.authentication.configurers.provisioning.UserDetailsManagerConfigurer;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

//@Component //NB: this extension in currently desactivated !!!
@Profile("withSecurity")
public class MySecurityDefaultUsersSimpleConfigurerSpecificImpl implements
MySecurityDefaultUsersSimpleConfigurer {

private PasswordEncoder passwordEncoder;

	@Autowired
	public MySecurityDefaultUsersSimpleConfigurerSpecificImpl(PasswordEncoder passwordEncoder){
		this.passwordEncoder=passwordEncoder;
	}

	@Override
	public void configureGlobalDefaultUsers(UserDetailsManagerConfigurer udmc) {
		udmc
		//same as default implementation:
		.withUser("user1").password(passwordEncoder.encode("pwd1")).roles("USER")
		.and().withUser("admin1").password(passwordEncoder.encode("pwd1")).roles("ADMIN")
		.and().withUser("user2").password(passwordEncoder.encode("pwd2")).roles("USER")
		.and().withUser("admin2").password(passwordEncoder.encode("pwd2")).roles("ADMIN")
		//specific others users:
		.and().withUser("user3").password(passwordEncoder.encode("pwd3")).roles("USER")
		.and().withUser("admin3").password(passwordEncoder.encode("pwd3")).roles("ADMIN");
	}

	//Effet immediat en mode InMemory
	//attention, en mode jdbc , la base de données "realmdb..." doit être (manuellement) réinitialisée
}
```

Ainsi défini et enregistré , ce composant sera prioritaire vis à vis de l'implémentation par défaut (codée dans le projet my-security-realm-configure) .

NB : l'interface contractuelle org.mycontrib.mysecurity.common.extension.MySecurityDefaultUsersSimpleConfigurer *permet si besoin de distinguer des listes d'utilisateurs différentes pour les besoins "rest" et "site" .*

1.9. *Jdbc Realm*

*mysecurity.realm.**site**.... ou mysecurity.realm.**rest**.... ou mysecurity.realm.**global**....*

```
mysecurity.realm.global.jdbc-realm.driverClassName=org.h2.Driver
mysecurity.realm.global.jdbc-realm.url=jdbc:h2:~/realmdb5
mysecurity.realm.global.jdbc-realm.username=sa
mysecurity.realm.global.jdbc-realm.password=
```

NB : le mode "jdbc-realm" est prioritaire vis à vis du mode "InMemory" et la liste des utilisateurs préconfigurés *(liste éventuellement vide)* peut se configurer via un composant de type MySecurityDefaultUsersSimpleConfigurerSpecificImpl partagé avec le mode "InMemory" (avec même liste d'utilisateurs par défaut)

L'idée est de pouvoir basculer en douceur du mode "InMemory" au mode "jdbc" .

En mode production, le mode "jdbc" peut être pratique pour configurer une petite liste d'utilisateurs privilégiés (ex : "administrateurs") .

NB :

Au sein de l'implémentation courante/actuelle , les tables "users" et "authorities" sont automatiquement créées si elles n'existent pas déjà.

1.10. UserDetails Realm

Le mode "**UserDetails**" est le plus souple/fréquent au sein d'une application autonome.

Il consiste à configurer un service d'adaptation pour que Spring-security puisse utiliser les services métiers et "DAO" de l'application pour récupérer une liste d'utilsateurs dans une base de données spécifiques à l'application (et via les connectiques spring habituelles (jpa ou mongo , spring-data, …).
Sans surprise, il faut pour cela implémenter l'interface standard ***UserDetailsService*** de spring-security.

Seule adapation à effectuer pour que ce soit bien pris en compte par le projet *my-security-realm-configure* : il faut donner comme nom à ce service une des constantes de l'interface org.mycontrib.mysecurity.common.extension.***MySecurityExtension*** .

Exemple :

```
@Profile("withSecurity")
@Service(MySecurityExtension.MY_EXCLUSIVE_USERDETAILSSERVICE_NAME)
//@Service(MySecurityExtension.MY_ADDITIONAL_USERDETAILSSERVICE_NAME)
public class MyUserDetailsService implements UserDetailsService {

    Logger logger = LoggerFactory.getLogger(MyUserDetailsService.class);

  @Autowired
   private PasswordEncoder passwordEncoder;

  @Autowired
  private ServiceCustomer serviceCustomer;

  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        UserDetails userDetails=null;
        logger.debug("MyUserDetailsService.loadUserByUsername() called with username="+username);

        List<GrantedAuthority> authorities = new ArrayList<GrantedAuthority>();
        String password=null;
    if(username.equals("james_Bond")) {
                password=passwordEncoder.encode("007");//simulation password ici
                authorities.add(new SimpleGrantedAuthority("ROLE_AGENTSECRET"));
                userDetails = new User(username, password, authorities);
    }
    else {
        //NB le username considéré comme potentiellement égal à firstname_lastname
        try {
        String firstname = username.split("_")[0];
        String lastname = username.split("_")[1];

        List<Customer> customers =
                 serviceCustomer.rechercherCustomerSelonPrenomEtNom(firstname,lastname);
        if(!customers.isEmpty()) {
                Customer firstCustomer = customers.get(0);
                authorities.add(new SimpleGrantedAuthority("ROLE_CUSTOMER"));
                        //ou "ROLE_USER" ou "ROLE_ADMIN"
                password=firstCustomer.getPassword();// déjà stocké en base en mode crypté
```

```
                //password=passwordEncoder.encode(firstCustomer.getPassword());
                //si pas stocké en base en mode crypté (PAS BIEN !!!)
                userDetails = new User(username, password, authorities);
            }
        } catch (Exception e) {
                //e.printStackTrace();
        }
    }

    if(userDetails==null) {
        //NB: il est important de remonter UsernameNotFoundException
        //(mais pas null , ni une autre exception)
        //si l'on souhaite qu'en cas d'échec avec cet AuthenticationManager
        //un éventuel AuthenticationManager parent soit utilisé en plan B
        throw new UsernameNotFoundException(username
                                            + " not found by MyUserDetailsService");
    }
   return userDetails;
  }
}
```

1.11. Realm combiné "UserDetails + jdbc/InMemory"

Si l'on souhaite que l'application utilise un realm combiné de type "UserDetails + jdbc/InMemory" , il suffit de donner un nom de type MY_**ADDITIONAL**_USERDETAILSSERVICE_NAME plutôt que MY_EXCLUSIVE_USERDETAILSSERVICE_NAME au composant spring MyUserDetailsService .

NB : L'interface *org.mycontrib.mysecurity.common.extension.MySecurityExtension* recense les valeurs possibles :

```
public static final String MY_EXCLUSIVE_USERDETAILSSERVICE_NAME
public static final String MY_ADDITIONAL_USERDETAILSSERVICE_NAME

//Si le UserDetailsService ne doit être utilisé que sur partie "rest"
public static final String MY_EXCLUSIVE_RESTONLY_USERDETAILSSERVICE_NAME
public static final String MY_ADDITIONAL_RESTONLY_USERDETAILSSERVICE_NAME

//Si le UserDetailsService ne doit être utilisé que sur partie "site"
public static final String MY_EXCLUSIVE_SITEONLY_USERDETAILSSERVICE_NAME
public static final String MY_ADDITIONAL_SITEONLY_USERDETAILSSERVICE_NAME
```

Comportement d'un realm combiné en mode "combiné / ADDITIONAL" :

1. L'utilisateur souhaitant s'authentifier est d'abord authentifié par le "USERDETAILSSERVICE".
2. Si l'authentification échoue (retourne "UsernameNotFoundException") , alors une seconde tentative d'authentification est alors effectuée via le AuthenticationManager parent s'il existe (souvent en mode "jdbc" ou "InMemory") .