Configurez une page de connexion customisée avec Spring Security

Commençons par configurer la chaîne de filtres de sécurité de Spring Security pour l'authentification et l'autorisation.

Démarrez par le formulaire Spring Security par défaut, en utilisant la méthode `formLogin()` .

```
    @Override
    public void configure(HttpSecurity http) throws Exception {
         http
            .formLogin();
       }
```

Ajoutez des rôles d'utilisateur et d'administrateur à votre chaîne de filtres

Spring Security fournit des **filtres** qui peuvent être utilisés pour authentifier différents types de rôles. Définissez les deux rôles suivants dans votre app Spring Boot :

1. Utilisateur.
2. Administrateur.

Une fois qu'ils sont authentifiés, Spring Security se charge d'octroyer les autorisations appropriées, et contrôle l'accès aux données selon les rôles.

En premier lieu, ajoutez la méthode `authorizeRequests()` pour définir les rôles.

Ensuite, ajoutez la méthode `antMatchers()` pour définir l'association des rôles `USER` (utilisateur) et `ADMIN` (administrateur) avec des pages.

Cela permettra d'assurer **l'étape d'autorisation** pour les pages qui sont à la disposition des utilisateurs. Le rôle `ADMIN` se voit attribuer une page d'accueil spécifique, `/admin` , mais a également accès à `/user` . Les utilisateurs ont accès à la page d'accueil `/user` , mais ne devraient pas avoir accès à la page d'accueil `/admin` . Ajoutez `anyRequest().authenticated()` pour vous permettre d'utiliser le formulaire ci-dessous pour l'authentification.

Procédez comme suit, en ajoutant l'extrait de code à votre méthode `configure()` qui gère les requêtes HTTP.

```
@Override
public void configure(HttpSecurity http) throws Exception {
      http
         .authorizeRequests()
         .antMatchers("/admin").hasRole("ADMIN")
         .antMatchers("/user").hasRole("USER")
         .anyRequest().authenticated()
         .and()
         .formLogin();
}
```

Votre chaîne de filtres de sécurité est programmée avec HTTPSecurity.

Maintenant, nous pouvons créer une chaîne de filtres de sécurité pour l'étape d'authentification de ce petit chef-d'œuvre. Pour ce faire, utilisez AuthenticationManagerBuilder.

Ajoutez AuthenticationManagerBuilder

Essayons de mettre en place une simple page de connexion pour tester notre contrôle d'accès basé sur les rôles, avec Spring Security.
Utilisez **AuthenticationManagerBuilder** pour assigner les rôles d'utilisateur et d'administrateur. Ce filtre permet non seulement de créer des identifiants encodés, mais également de les assigner à des rôles.

Cela va nous permettre d'ajouter des utilisateurs avec des rôles prédéfinis afin de pouvoir nous connecter à notre application, sans aucune autre configuration.

Ensuite, vous utiliserez `auth.inMemoryAuthentication` . Cela signifie que les identifiants créés seront stockés dans la mémoire, plutôt que dans un jeton ou dans une base de données.

> Si vous souhaitez que votre authentification soit réalisée en connexion avec une base de données, vous pouvez mettre en place une authentification basée sur JDBC (**J**ava **D**ata**b**ase **C**onnectivity) de la même manière. Au lieu de recourir à `auth.inMemoryAuthentication` , utilisez `auth.jdbcAuthentication()` et ajoutez une configuration de base de données. Voici quelques infos pratiques sur cette **procédure**.

Voici la librairie pour **AuthenticationManagerBuilder** :

```
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder
```

```
@Override
```

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception
      auth.inMemoryAuthentication()
         .withUser("springuser").password(passwordEncoder().encode("spring123")).roles("USER")
         .and()
         .withUser("springadmin").password(passwordEncoder().encode("admin123")).roles("ADMIN", "USER");
}
```

Je vous ai préparé un utilisateur et un admin avec lesquels vous pouvez vous connecter. Voici les identifiants :

- **USER role** : *utilisateur : **springuser** ; mot de passe : **spring123***.
- **ADMIN role** : *utilisateur : **springadmin*** ; *mot de passe : **admin123***.

J'ai également ajouté un encodage aux mots de passe. Ainsi, même si j'ai écrit en dur des mots de passe simples, la méthode `encode()` permettra comme son nom l'indique de les encoder.

Comme nous l'avons fait pour la classe **SpringSecurityConfig**, ajoutez un **Bean** avec la méthode `passwordEncoder()` .

Cela va vous permettre de choisir le type d'algorithme de hachage avec lequel vous souhaitez encoder votre mot de passe. En l'occurrence, j'ai choisi **BCrypt,** car il s'agit d'un des algorithmes d'encodage les plus reconnus en ce qui concerne les mots de passe. Je vous le recommande.

> Si vous souhaitez en apprendre davantage sur les algorithmes de hachage, rendez-vous sur la page de la classe **PasswordEncoder**.

Pour profiter de cette fonctionnalité, ajoutez trois librairies à vos imports :

```
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
```

```
import org.springframework.security.crypto.password.PasswordEncoder;
```

Ajoutez le code suivant à votre classe **SpringSecurityConfig** :

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

Révisez votre chaîne de filtres de sécurité

Révisons le fichier **SpringSecurityconfig.java** que vous venez de compléter. Il devrait ressembler à peu près à ça si vous avez bien suivi (soyez attentif au format) :

```
package com.openclassrooms.login.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
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.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableWebSecurity
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser("springuser").password(passwordEncoder().encode("spring123"))
            .roles("USER")
            .and()
            .withUser("springadmin").password(passwordEncoder().encode("admin123"))
            .roles("ADMIN", "USER");
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/admin").hasRole("ADMIN")
            .antMatchers("/user").hasRole("USER")
            .anyRequest().authenticated()
            .and()
            .formLogin()
            .and()
            .oauth2Login();
    }

    @Bean
```

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

Mettez en place votre Contrôleur

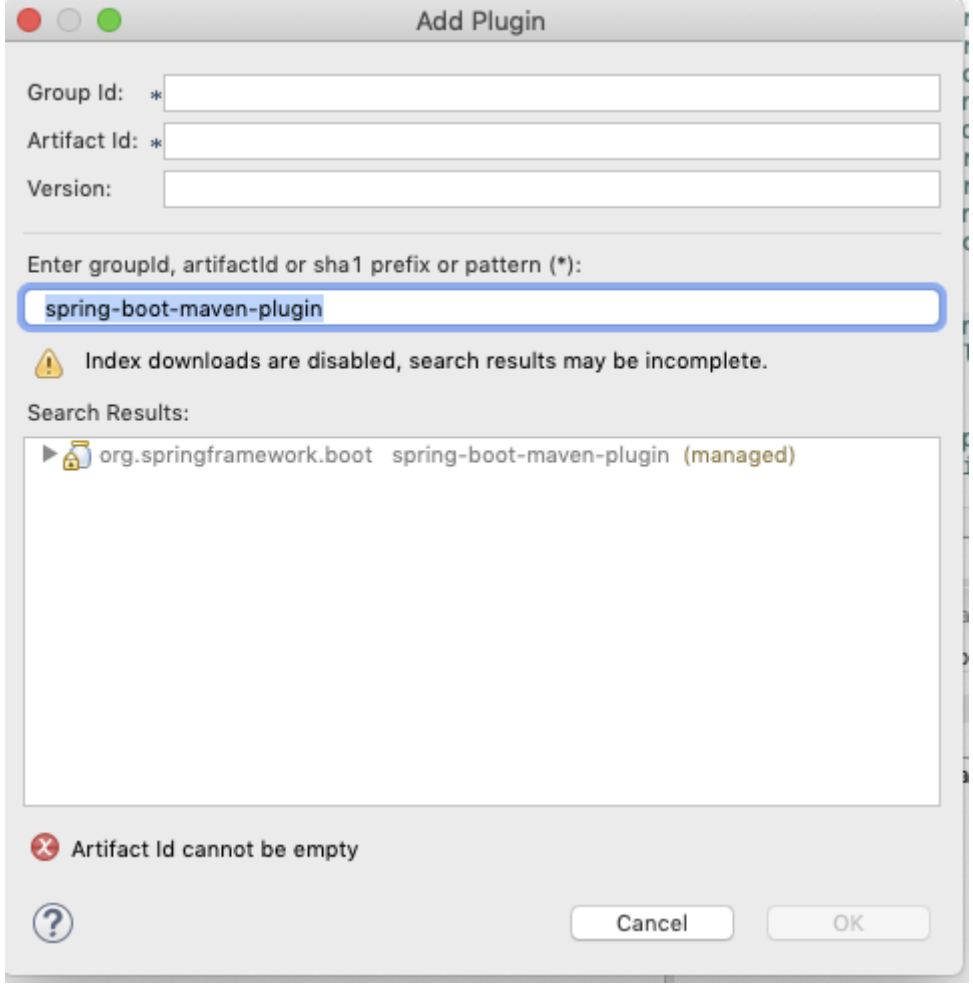
Maintenant, il est temps de mettre en place votre contrôleur. Ce contrôleur va gérer l'affichage des pages une fois que nous serons authentifiés.

Nous allons créer deux pages :

- une pour l'utilisateur ;
- et une autre pour l'administrateur.

Ensuite, vous allez pouvoir mettre en place votre contrôleur **REST**, qui se chargera de créer vos pages d'accueil. Pour garantir son fonctionnement, assurez-vous de disposer d'une version à jour de Maven, et d'avoir défini le Spring Boot Maven plugin en tant que dépendance. Eclipse rend cette procédure très facile :

- Référez-vous au fichier **pom.xml** qui contient vos dépendances.
- Faites un clic droit sur *pom.xml -> Maven -> Update project.*
- Vous devez également ajouter le Spring Boot Maven plugin pour un support web supplémentaire.
- Faites un clic droit sur *pom.xml-> Maven -> Add Plugin.*
- Tapez `spring-boot-maven-plugin` .



Ajoutez le plugin Maven

Une fois le plugin ajouté, il devrait être visible dans votre fichier **pom.xml**. Les librairies dont vous avez besoin pour faire fonctionner votre contrôleur **REST** devraient se mettre à jour. Il est également essentiel de créer un nouveau package pour votre contrôleur.

- Faites un clic droit sur votre package ***com.openclassrooms*** *-> New -> Package.*
- Pour le nom du package, ajoutez le mot *controller* à votre nom de package principal. Le mien s'intitule **com.openclassrooms.controller**.
- Puis, faites un clic droit sur le package que vous venez de réaliser, ***com.openclassrooms.controller*** *-> New -> Class.*
- Intitulez votre classe contrôleur **LoginController.java**.
- Ajoutez l'annotation `@RestController` pour la prise en compte de votre classe en tant que contrôleur REST par Spring.

Ensuite, créez une méthode différente pour chaque rôle, en utilisant les classes `@RolesAllowed` et `@RequestMapping` pour associer l'URL au rôle.

```
package com.test.controller;

import javax.annotation.security.RolesAllowed;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class LoginController

{
   @RolesAllowed("USER")
   @RequestMapping("/*")
   public String getUser()
   {
      return "Welcome User";
   }

   @RolesAllowed({"USER","ADMIN"})
   @RequestMapping("/admin")
   public String getAdmin()
   {
      return "Welcome Admin";
   }
}
```

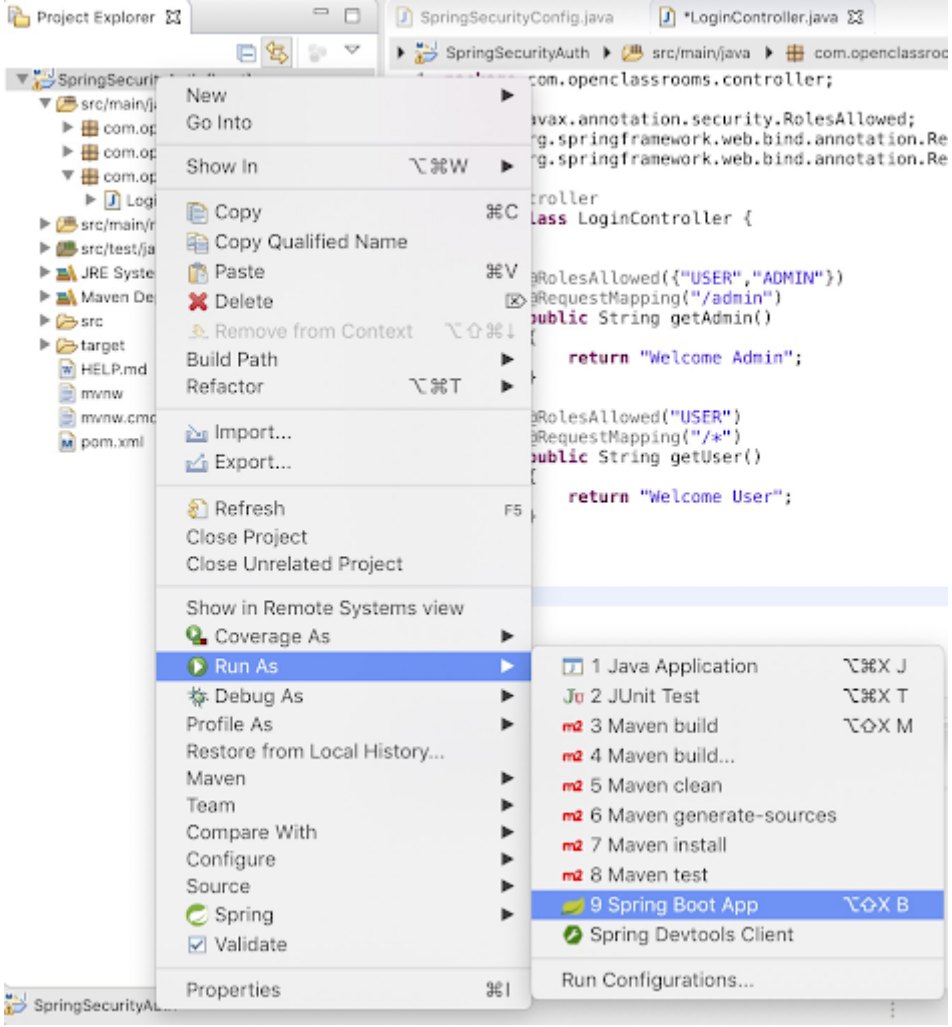
Pas besoin de s'embêter à créer une nouvelle page d'accueil pour l'admin et l'utilisateur : le contrôleur les construit pour vous de manière dynamique. Trop facile !

Maintenant, voyons ce que ça donne dans le navigateur.

Mettez en place votre Serveur Web

Lorsque vous lancez votre app indépendante, vous pouvez retrouver votre page de connexion en utilisant **Apache Tomcat** en serveur web. Aucune configuration n'est nécessaire ; tout s'effectue automatiquement à la compilation et à l'exécution du code.

- **Étape 1 :** sur Eclipse, rendez-vous sur votre dossier dans votre Project Explorer qui affiche le nom de votre app. La mention [boot] apparaît à côté, entre crochets. Cliquez sur *Run As -> Spring Boot App.*



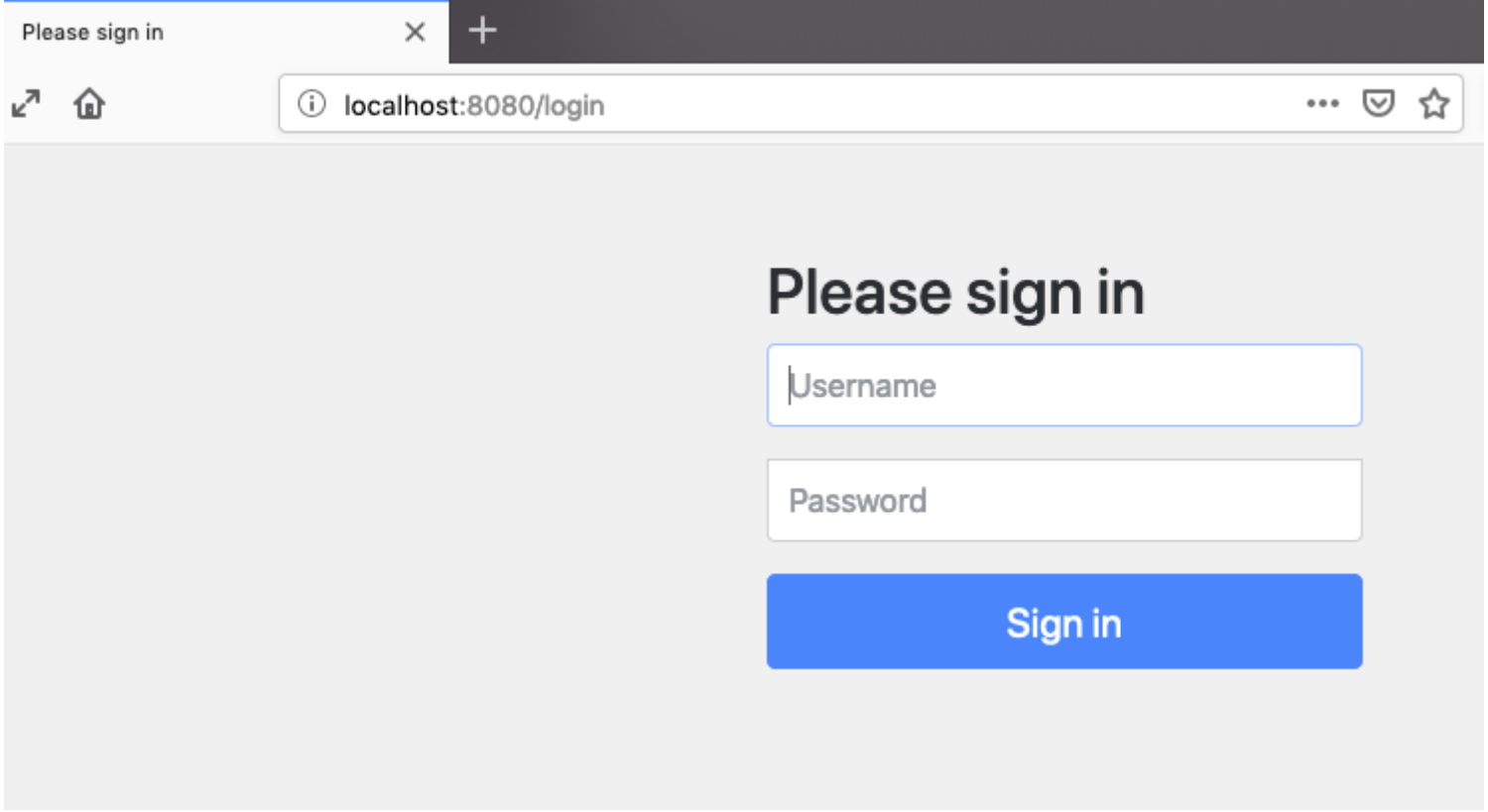
Exécutez SpringSecurityAuth

Vous constatez que votre code se compile dans votre console. Une fois qu'il est complété, vous pouvez ouvrir votre application web sur votre navigateur.

- **Étape 2 :** Ouvrez votre navigateur favori et tapez cette URL : http://localhost:8080.

  Si vous voulez changer le port que vous utilisez en un autre, comme 8090, vous pouvez ajouter `server.port=8090` dans le fichier **application.properties** > **src/main/resources**.

Maintenant, connectez-vous en tant que *user* et voyez ce qu'il se passe.



Le formulaire de connexion de Spring Security par défaut

Yes, ça fonctionne !

Voici quelques tests à mener pour être certain que vous l'avez correctement mis en place :

- Que se passe-t-il lorsque vous rentrez le mauvais mot de passe ?
- Que se passe-t-il lorsque vous tentez de consulter la page /admin alors que vous êtes connecté en tant que user ?

Si vous rencontrez des problèmes, vérifiez tous vos codes et dépendances pour vous assurer que tout est correctement programmé. Eclipse trouve les erreurs facilement. Vous pouvez traiter chaque dysfonctionnement au fil de l'eau.

> Assurez-vous de fermer votre connexion à `localhost:8080` lorsque vous avez terminé.
>
> Cela évitera que la précédente connexion soit toujours active la prochaine fois que vous le compilerez et l'exécuterez.

Vous pouvez faire cela en cliquant sur le carré rouge dans la console d'Eclipse.

En résumé

Vous avez fait du beau travail, vous savez maintenant :

- Créer des filtres avec `authorizeRequests()` et `antMatchers()` pour appliquer le contrôle d'accès basé sur des rôles à différentes URL.
- Configurer le rôle des utilisateurs en utilisant la classe **AuthenticationManagerBuilder**.
- Mettre en place les méthodes permettant de créer des pages d'accueil dynamiques pour les utilisateurs, en ayant recours aux classes `RolesAllowed` et `RequestMapping` .

On dirait que votre page d'authentification Spring Security est prête.

Dans la prochaine partie, nous allons créer un formulaire de connexion en utilisant OAuth. OAuth vous permet de vous connecter via Facebook, Google et autres. Retrouvons-nous juste après ce quiz, pour comprendre tous les avantages de OAuth.