# Mise en place de Spring Security

# Démonstration 1 du module 6

|  |
| --- |
| Les objectifs de cette démonstration sont   * Utilisation et configuration des starters de Spring Security * Test de l’utilisateur par défaut de Spring Security et Postman * Création d’une classe de configuration de Spring Security * Déclaration des utilisateurs via la base de données * Restriction des URLs et des méthodes http selon les rôles |

Contexte

* Nous reprenons notre application représentant une partie du cœur de métier de l’ENI Ecole.
  + Il s’agit de l’application demo-eni-ecole
  + Si votre application n’est pas complète, vous pouvez reprendre le code Java dans les ressources
  + Attention, le paramétrage du projet Spring Boot n’est pas inclus
    - Il vous faut reprendre les démonstrations des modules 1, 2 et 4 pour valider les starters de Spring Boot
* Dans cette itération, nous voulons ajouter l’authentification d’un utilisateur et ses habilitations

Pour la sécurité, nous voulons déclarer 2 rôles différents

* Rôle « EMPLOYE » - a les droits en lecture sur l’application (GET)
* Rôle « ADMIN » - a tous les droits du rôle « EMPLOYE » et les droits de créations, mise à jour et suppression (POST, PUT, PATCH, DELETE)

Déroulement

# Dépendances Spring Boot :

* Dans build.gradle ; ajouter le starter de sécurité

//Security

**implementation** 'org.springframework.boot:spring-boot-starter-security'

**testImplementation** 'org.springframework.security:spring-security-test'

* Faire un Gradle Refresh Project
* Votre projet est à présent un projet sécuriser par Spring Security

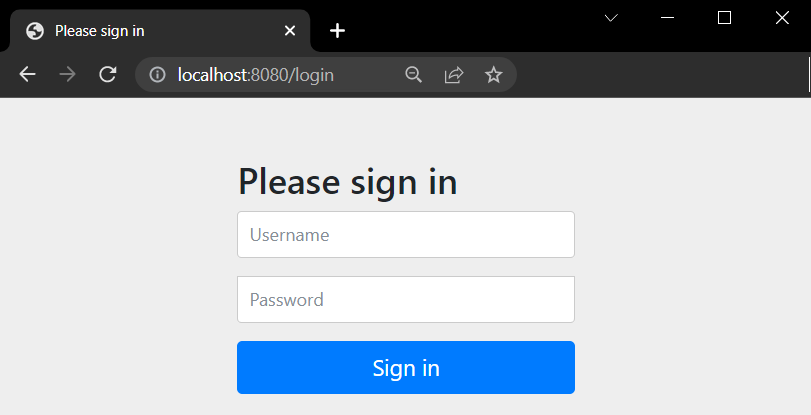
## Exécution

* Lancer l’application
  + Dans les traces, Spring Security nous fournit un mot de passe par défaut pour l’utilisateur par défaut « user »

Using generated security password: 311fef29-36e6-4d15-84e8-819cac2772d7

This generated password is for development use only. Your security configuration must be updated before running your application in production.

* Dès que nous voulons accéder à une URL,
  + Nous sommes redirigés vers le formulaire de connexion :



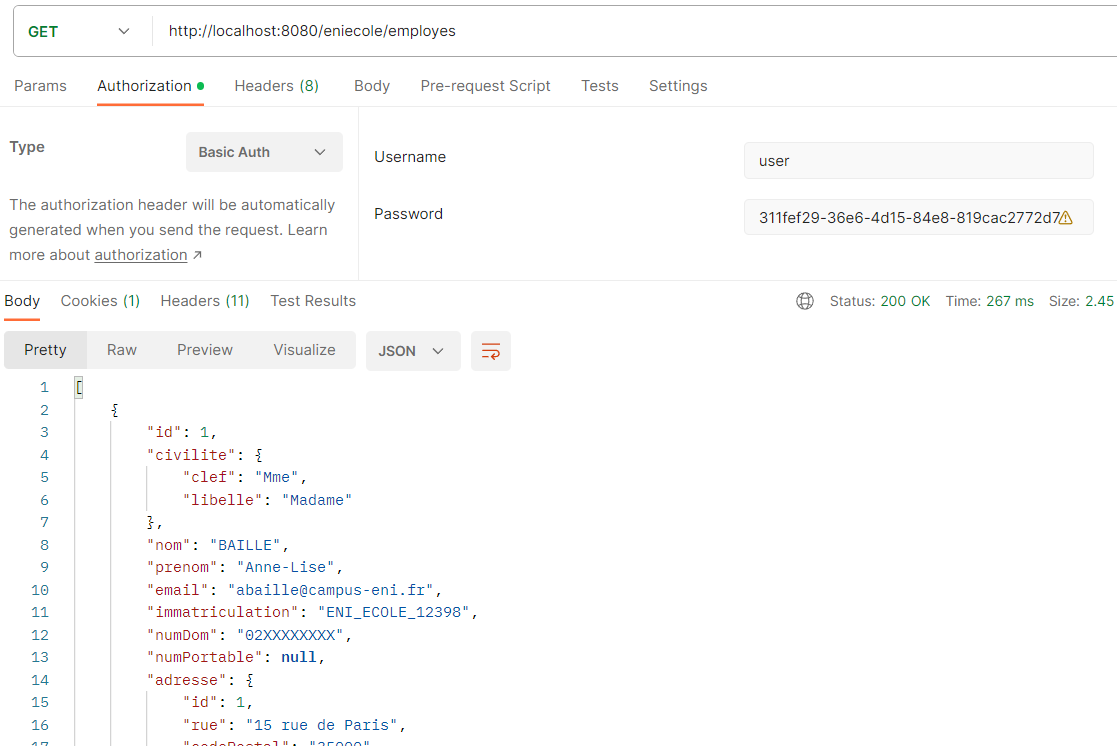
* + Il est possible d’utiliser user et ce mot de passe pour se connecter

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

## Dans Postman

* Test avec un appel sur la recherche des Employe
  + Créer une nouvelle requête avec l’URL http://localhost:8080/eniecole/employes
  + Ajouter une « Authorization » de type « Basic Auth »
    - Username : user
    - Password : le mot de passe fournit par Spring Security



* Vous pouvez afficher les données
* Modifier l’« Authorization » en « No Auth »; vous obtiendrez un échec
  + Statut : 401 – Unauthorized

Dès que nous activons la sécurité toutes les URLs de notre application demande un utilisateur.

# Déclaration des utilisateurs depuis la base :

* Nous allons créer la table users :

Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

* Nous avons donc pseudo qui permet d’identifier de manière unique notre utilisateur
  + Il correspond à l’email d’un Employe
* Le mot de passe est sur 68 caractères pour préciser l’algorithme de chiffrement au début de la chaîne {bcrypt}
  + Ainsi, si nous voulons changer l’algorithme, il suffira de le préciser dans le mot de passe enregistré.
* « authority » qui représente le rôle de l’utilisateur
* Un script de base de données vous est fourni dans les ressources
  + Remarquer dans ce script que les rôles sont préfixés par ROLE\_ pour suivre la configuration de Spring Security
* Création de la classe de configuration
  + Création d’une méthode userDetailsManager
    - Elle prend en paramètre la DataSource
    - Elle retourne UserDetailsManager
    - Elle permet de créer les 2 requêtes, pour récupérer
      * L’identifiant, le mot de passe et l’activation d’un utilisateur
      * La liste des rôles de l’utilisateur

**package** fr.eni.demo.security;

**import** javax.sql.DataSource;

**import** org.apache.commons.logging.\*;

**import** org.springframework.context.annotation.\*;

**import** org.springframework.security.provisioning.\*;

@Configuration

**public** **class** EniSecurityConfig {

**protected** **final** Log logger = LogFactory.*getLog*(getClass());

/\*\*

\* Récupération des utilisateurs de l'application via la base de données

\*/

@Bean

UserDetailsManager userDetailsManager(DataSource dataSource) {

JdbcUserDetailsManager jdbcUserDetailsManager = **new** JdbcUserDetailsManager(dataSource);

jdbcUserDetailsManager.setUsersByUsernameQuery("select pseudo, password, 1 from users where pseudo=?");

jdbcUserDetailsManager.setAuthoritiesByUsernameQuery("select pseudo, authority from users where pseudo=?");

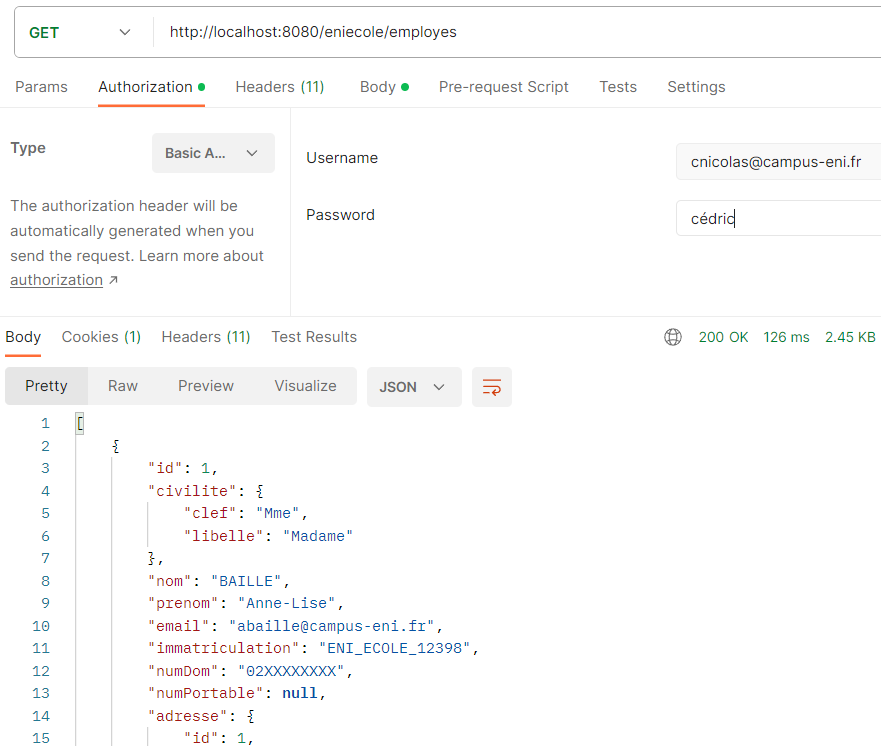
**return** jdbcUserDetailsManager;

}

…

## Exécution dans Postman

* Utiliser un des utilisateurs en base
  + Exemple cnicolas@campus-eni.fr et cédric



# Accès limités des URLs selon les rôles

Pour la sécurité, nous voulons déclarer 2 rôles différents

* Rôle « EMPLOYE » - a les droits en lecture sur l’application (GET)
* Rôle « ADMIN » - a tous les droits du rôle « EMPLOYE » et les droits de créations, mise à jour et suppression (POST, PUT, PATCH, DELETE)
* L’URL racine /eniecole doit être accessible à tout le monde (sans connexion)

## Modification de la classe de configuration :

* Ajout de l’annotation pour gérer les filtres sur les URLs et les méthodes du protocole http :
  + @EnableWebSecurity
* Déclaration d’un bean de type SecurityFilterChain au travers d’une méthode filterChain ;
  + Elle prend en argument HttpSecurity
  + Permet de configurer les méthodes et url du protocole http accessible selon un certain rôle
  + Les ressources accessibles par tous
  + Déclaration d’une connexion via le formulaire par défaut

**package** fr.eni.demo.security;

**import** javax.sql.DataSource;

**import** org.apache.commons.logging.\*;

**import** org.springframework.context.annotation.\*;

**import** org.springframework.http.HttpMethod;

**import** org.springframework.security.config.Customizer;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

**import** org.springframework.security.provisioning.\*;

**import** org.springframework.security.web.SecurityFilterChain;

@Configuration

@EnableWebSecurity

**public** **class** EniSecurityConfig {

…

/\*\*

\* Restriction des URLs selon la connexion utilisateur et leurs rôles

\*/

@Bean

SecurityFilterChain filterChain(HttpSecurity http) **throws** Exception {

http.authorizeHttpRequests(auth -> {

auth

//Permettre l'accès à l'URL racine à tout le monde

.requestMatchers("/eniecole").permitAll()

//Permettre aux rôles EMPLOYE et ADMIN de manipuler les URLs en GET

.requestMatchers(HttpMethod.***GET***, "/eniecole/employes/\*\*").hasAnyRole("EMPLOYE","ADMIN")

//Restreindre la manipulation des méthodes POST, PUT, PATCH et DELETE au rôle ADMIN

.requestMatchers(HttpMethod.***POST***, "/eniecole/employes").hasRole("ADMIN")

.requestMatchers(HttpMethod.***PUT***, "/eniecole/employes").hasRole("ADMIN")

.requestMatchers(HttpMethod.***PATCH***, "/eniecole/employes").hasRole("ADMIN")

.requestMatchers(HttpMethod.***DELETE***, "/eniecole/employes").hasRole("ADMIN")

//Toutes autres url et méthodes HTTP ne sont pas permises

.anyRequest().denyAll();

});

//Use Http Basic Authentication

http.httpBasic(Customizer.*withDefaults*());

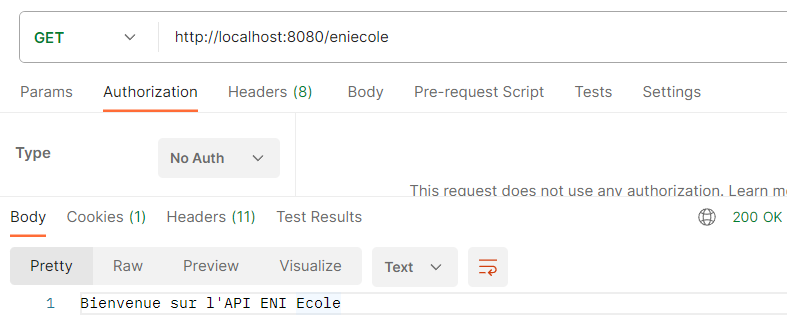
**return** http.build();

}

}

* Dans cette démonstration, nous avons fait le choix d’interdire toutes autres URL ou méthodes

## Exécution dans Postman d’un GET sur l’URL racine /eniecole



Toute demande sur l’URL racine est acceptée sans avoir besoin d’être authentifié.

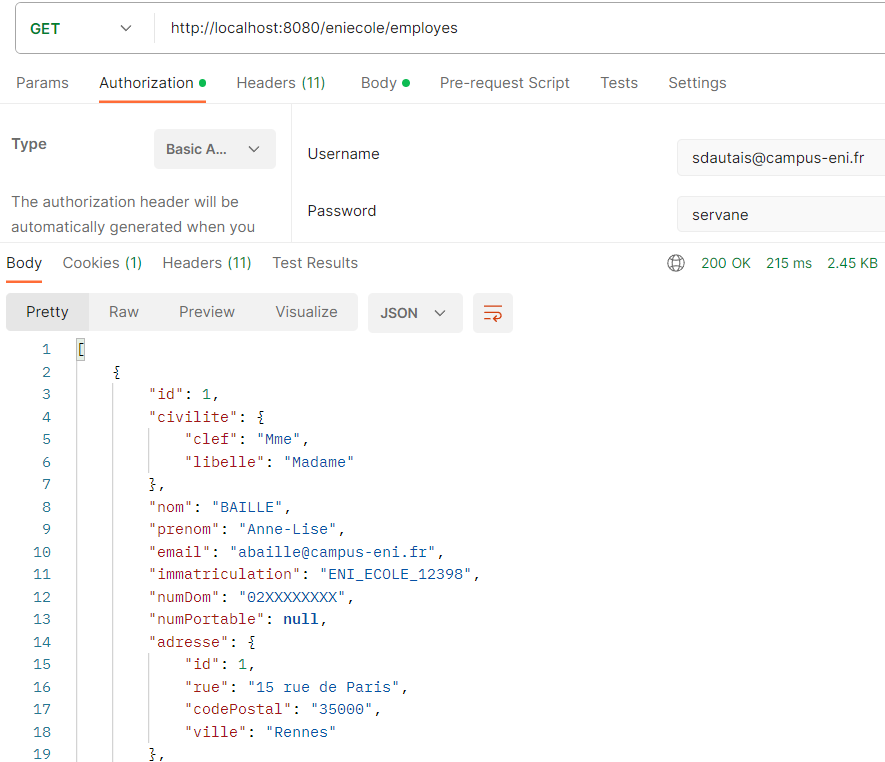
## Exécution dans Postman de GET

* GET et cnicolas@campus-eni.fr et cédric [Rôle : ADMIN]

Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

* GET et sdautais@campus-eni.fr et servane [Rôle : EMPLOYE]



Tous les utilisateurs reconnus ont accès aux URLs en GET

* Si vous tester avec un utilisateur inconnu : jtrillard@campus-eni.fr et julien
  + Vous obtenez un 401 « Unauthorized »

## Exécution dans Postman de POST

* POST et cnicolas@campus-eni.fr et cédric [Rôle : ADMIN]
  + Et un nouvel employé :

{

    "civilite": {

            "clef": "M"

    },

    "nom":"GROUSSARD",

    "prenom":"Thierry",

    "email":"tgroussard@campus-eni.fr",

    "immatriculation":"ENI\_ECOLE\_19090",

    "numPortable":"06XXXXXXXX",

     "adresse":{

       "rue":"Rue du Pont",

       "codePostal":"35000",

       "ville":"Rennes"

    }

}

* Si vous le tester directement, vous constatez que cet utilisateur qui a le rôle ADMIN ne peut pas créer de nouveau formateur
  + Vous obtenez un 401 « Unauthorized »
* Postman est un client http indépendant de notre application et du navigateur
  + Ajout de CSRF en disable
    - Pour permettre de manipuler l’API REST au travers d’outil autre que le navigateur web (ex : Postman)
  + **CSRF** (Cross-Site Request Forgery) est une attaque qui usurpe l'identité d'un utilisateur de confiance et envoie des commandes non désirées sur un site web.
    - Cela peut être réalisé, par exemple, en ajoutant des paramètres malveillants dans un URL associé à un lien qui prétend aller quelque part ailleurs.
  + Spring Security préconise de ne pas le désactiver
    - Mais pour nos manipulations au travers de Postman, nous le faisons
    - Bien se rendre compte que c’est nécessaire pour les POST, PUT, PATCH et DELETE
    - Pour les GET nous pouvions laisser par défaut le CSRF actif.

…

/\*\*

\* Restriction des URLs selon la connexion utilisateur et leurs rôles

\*/

@Bean

SecurityFilterChain filterChain(HttpSecurity http) **throws** Exception {

http.authorizeHttpRequests(auth -> {

auth

//Permettre l'accès à l'URL racine à tout le monde

.requestMatchers("/eniecole").permitAll()

//Permettre aux rôles EMPLOYE et ADMIN de manipuler les URLs en GET

.requestMatchers(HttpMethod.***GET***, "/eniecole/employes/\*\*").hasAnyRole("EMPLOYE","ADMIN")

//Restreindre la manipulation des méthodes POST, PUT, PATCH et DELETE au rôle ADMIN

.requestMatchers(HttpMethod.***POST***, "/eniecole/employes").hasRole("ADMIN")

.requestMatchers(HttpMethod.***PUT***, "/eniecole/employes").hasRole("ADMIN")

.requestMatchers(HttpMethod.***PATCH***, "/eniecole/employes").hasRole("ADMIN")

.requestMatchers(HttpMethod.***DELETE***, "/eniecole/employes").hasRole("ADMIN")

//Toutes autres url et méthodes HTTP ne sont pas permises

.anyRequest().denyAll();

});

//Use Http Basic Authentication

http.httpBasic(Customizer.*withDefaults*());

//Désactivé Cross Site Request Forgery

// Non préconisé pour les API REST en stateless. Sauf pour POST, PUT, PATCH et DELETE

http.csrf(csrf -> {

csrf.disable();

});

**return** http.build();

}

* Retester le POST et cnicolas@campus-eni.fr et cédric [Rôle : ADMIN]
  + Et un nouvel employé :

{

    "civilite": {

            "clef": "M"

    },

    "nom":"GROUSSARD",

    "prenom":"Thierry",

    "email":"tgroussard@campus-eni.fr",

    "immatriculation":"ENI\_ECOLE\_19090",

    "numPortable":"06XXXXXXXX",

     "adresse":{

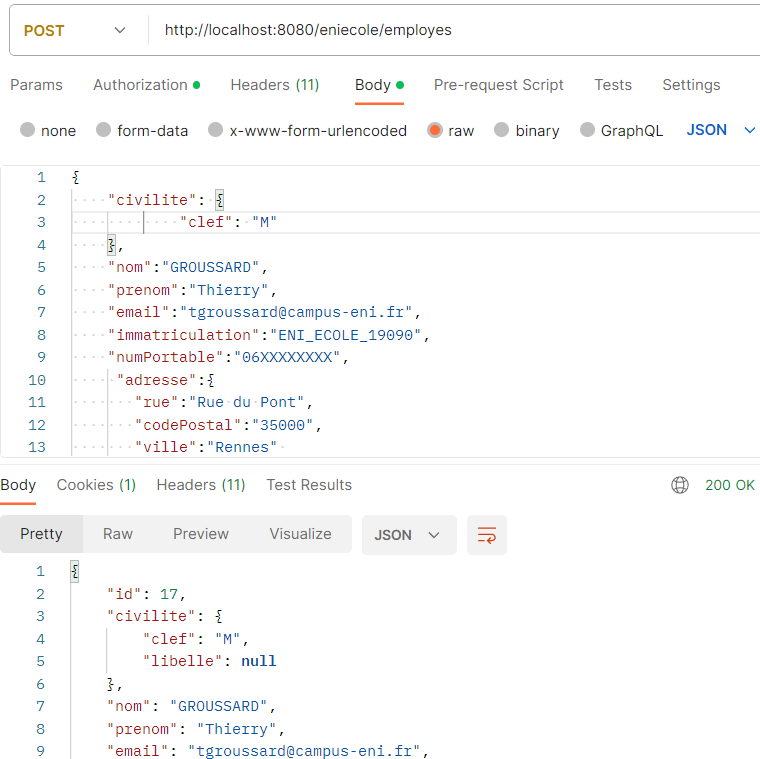
       "rue":"Rue du Pont",

       "codePostal":"35000",

       "ville":"Rennes"

    }

}



* Maintenant, l’utilisateur avec un rôle ADMIN peut créer des nouveaux employés
* Tester le POST et sdautais@campus-eni.fr et servane [Rôle : EMPLOYE]
  + Et un nouvel employé :

{

    "civilite": {

            "clef": "Mme"

    },

    "nom":"MANDART",

    "prenom":"Soazig",

    "email":"smandart@campus-eni.fr",

    "immatriculation":"ENI\_ECOLE\_20090",

    "numPortable":"06XXXXXXXX",

     "adresse":{

       "rue":"Rue du Chat",

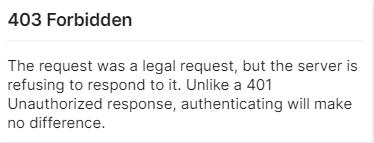
       "codePostal":"35000",

       "ville":"Rennes"

    }

}

* + Vous obtenez un 403 « Forbidden »



* + Cet utilisateur a le rôle EMPLOYE ; donc n’a pas le droit d’utiliser POST, PUT, PATCH ou DELETE

Maintenant notre application web service REST est protégée par la sécurité de Spring.