Spring Security

UP ASI
Bureau E204

PLAN DU COURS

-Introduction

-Configuration de Spring Security

-Mise en place de Spring Security

-TP

INTRODUCTION

- **Spring Security** est un Framework de sécurité léger qui fournit une authentification et un support d'autorisation afin de sécuriser les applications Spring.

- Spring Security répond à deux questions majeures :

- Qui peut accéder à l'application ? ➔ **Authentification**
- Est-ce que l'utilisateur connecté a le droit d'accéder à cette ressource ? ➔ **Autorisation** permettant ou pas à un utilisateur d'accéder à une ressource.

Configuration de Spring Security

- Pour accéder à une application, nous pouvons généralement procéder de deux façons :

- **La connexion statique**: En utilisant des données statiques (en mémoire ou dans un fichier).
- **La connexion dynamique**: En utilisant des données dynamiques provenant d'une base de données.

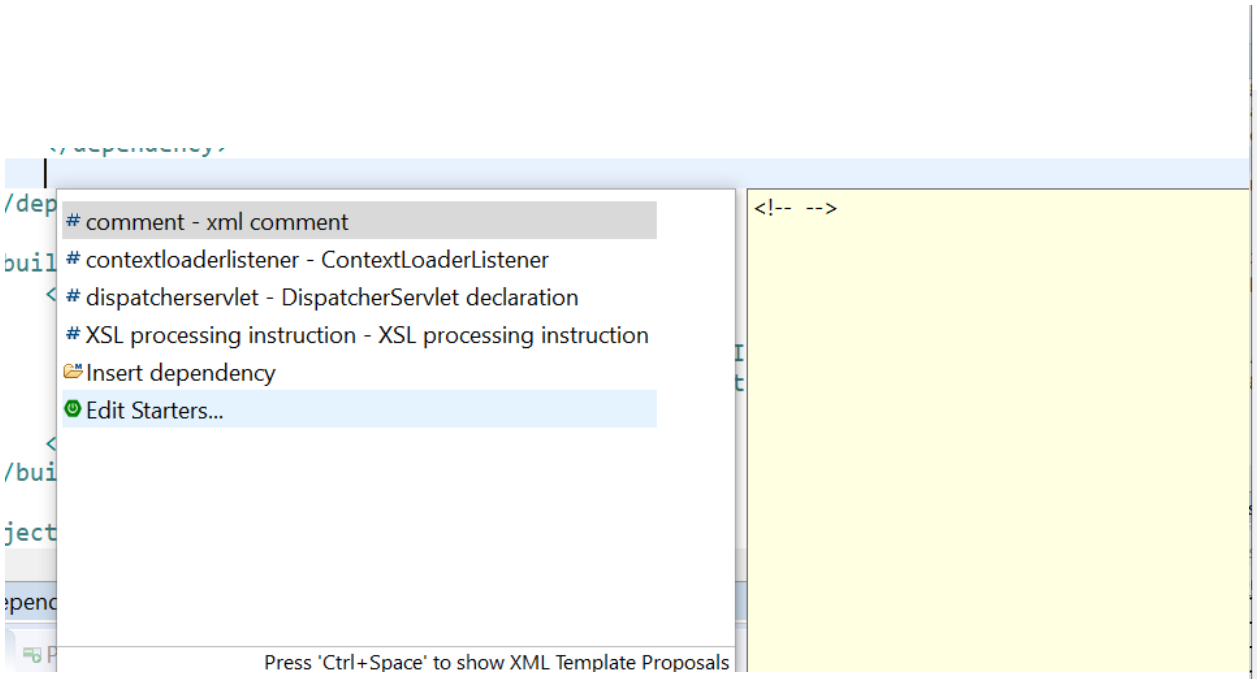
➔ **Pour notre cours, nous allons opter pour la connexion dynamique**

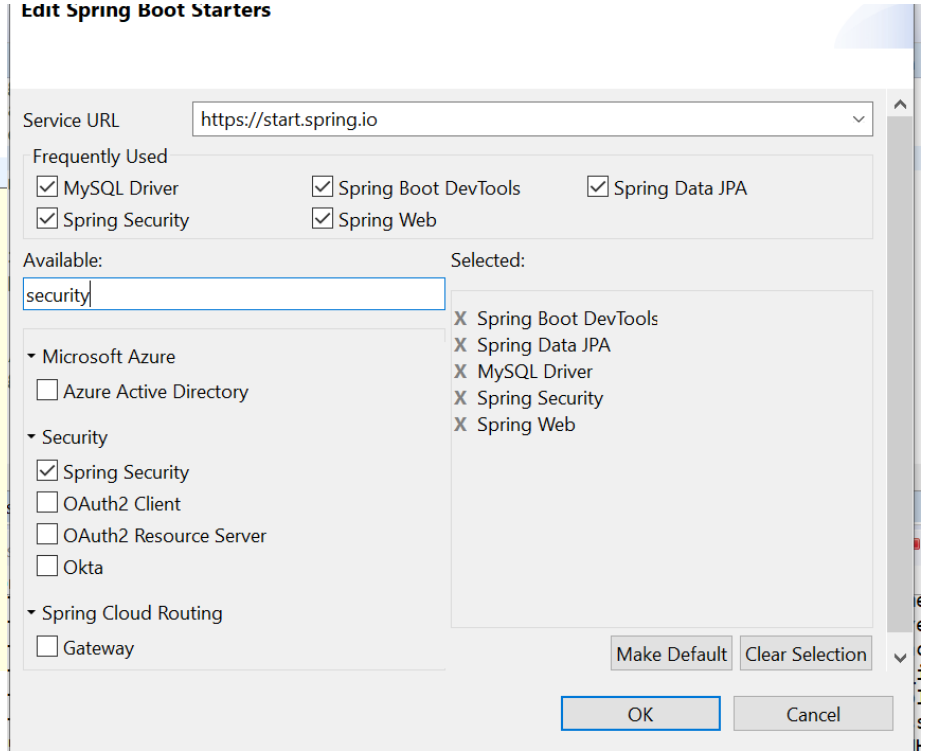
Configuration de Spring Security - DÉPENDANCES

- Pour configurer la partie sécurité au niveau de notre projet Spring Boot, il suffit d'aller dans le pom.xml et d'ajouter les dépendances Spring Security (ou le starter spring Security) :

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

Configuration de Spring Security - DÉPENDANCES





- Il suffit par la suite de faire un clean install et maven update de votre projet.

Spring Security- Création des entités User et Role

- Création des entités:
  - Pour gérer l'authentification et l'accès aux différentes fonctionnalités, il suffit de créer deux entités User et Role
  - Chaque user a ses propres attributs pour se connecter (username,password,etc…).
  - Chaque user peut avoir le rôle d'ADMIN ou SUPERADMIN.
  - Pour chaque rôle, l'utilisateur peut accéder à certaines fonctionnalités et sera interdit d'accéder à d'autres.
  - L'association entre user et rôle est ManyToMany

Spring Security- Création des entités User et Role

- Création des entités:
  - Créer les deux entités User et Role
  - Association: ManyToMany

```
@Entity
@Getter
@Setter

@AllArgsConstructor

@NoArgsConstructor
public class User  implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;
    private String userName;
    private String email;
    private String password;
    private String name;
    private String lastName;
    private Boolean active;
    @ManyToMany(cascade = CascadeType.PERSIST, fetch
    = FetchType.EAGER)
    private Set<Role> roles;
    }
```

```
@Entity @Getter @Setter

@AllArgsConstructor

@NoArgsConstructor
public class Role  implements Serializable {
class User   implements Serializable {
          @Id
    @GeneratedValue(strategy =
GenerationType.AUTO)

private int id;

@Enumerated(EnumType.STRING)

private RoleName role;
```

```
import org.springframework.security.core.GrantedAuthority;
public enum RoleName implements
GrantedAuthority {
ADMIN,
SUPERADMIN;
  @Override
  public String getAuthority() {
    return "ROLE_" + name();
  }}
```

Spring Security- Configuration de la couche Repository

- Dans la couche repository, créer les deux interfaces Repository respectifs aux entités User et Role: UserRepository et RoleRepository.
- Nous allons utiliser les deux fonctionnalités basées sur des **keywords** findByUserName et findByRole.

```
@Repository
public interface UserRepository extends CrudRepository<User,
Long> {

    User findByUserName(String userName);
}
```

```
@Repository
public interface RoleRepository extends CrudRepository<Role,
Integer> {
    Role findByRole(String role);

}
```

Spring Security- Configuration de la couche Service

Dans la couche service, créer la classe service **MyUserDetailsService**

- `Implémente` **`org.springframework.security.core.userdetails.UserDetailsService`** :
- Implémente les méthodes permettant de retourner des informations sur l'utilisateur connecté dans la base de données.
- Un objet de classe UserDetails est créé à partir de l'utilisateur et envoyé au fichier de configuration de sécurité

- La méthode **loadUserByUsername**() sera invoquée par Spring Security lors de l'authentification des utilisateurs.
- La méthode List<GrantedAuthority> getUserAuthority(Set<Role> userRoles) renvoie un ensemble de rôles (autorités) à utiliser par Spring Security dans le processus d'autorisation.

Configuration de la couche Service: La classe MyUserDetailsService

```
import javax.transaction.Transactional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
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;
@Service
public class MyUserDetailsService implements UserDetailsService {
    @Autowired
    private UserService userService;
    @Override
    @Transactional
    public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {
        User user = userService.findUserByUserName(userName);
        List<GrantedAuthority> authorities = getUserAuthority(user.getRoles());
        return new org.springframework.security.core.userdetails.User(user.getUserName(),
user.getPassword(),user.getActive(), true, true, true, authorities);    }
…
```

Configuration de la couche Service: La classe MyUserDetailsService

```
…
private List<GrantedAuthority> getUserAuthority(Set<Role> userRoles) {
        Set<GrantedAuthority> roles = new HashSet<GrantedAuthority>();
        for (Role role : userRoles) {
            roles.add(new SimpleGrantedAuthority(role.getRole()));       }
        List<GrantedAuthority> grantedAuthorities = new ArrayList<>(roles);
        return grantedAuthorities;
  }
```

Configuration de la couche Service: La classe UserService

- Dans la couche Service, créer la classe UserService qui contient les méthodes:
  - saveUser
  - findUserByUserName

- **`UserService`** implémente donc une méthode vérifiant l'existence d'un utilisateur selon la valeur de userName (en utilisant le repository).
- La méthode **saveUser** permet de sauvegarder l'utilisateur tout en cryptant le mot de passe dans la base de données.

Configuration de la couche Service: La classe UserService

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

@Service
public class UserService {
@Autowired
private UserRepository userRepository;
BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();

public User findUserByUserName(String userName) {
        return userRepository.findByUserName(userName);
    }
    public User saveUser(User user) {
      user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
      user.setActive(true);
      return userRepository.save(user);   }}
```

Spring Security- Configuration de la classe SecurityConfig

- Dans la couche sécurité (dans un package nommé security), créer la classe WebSecurityConfig (elle sert à configurer la sécurité de l'application):

  - Annotée **@EnableWebSecurity** pour activer la prise en charge de la sécurité Web de Spring Security.
  - Etend **WebSecurityConfigurerAdapter** et remplace quelques-unes de ses méthodes pour définir certaines spécificités de la configuration de la sécurité Web.
  - La méthode **configure(HttpSecurity**) définit quels chemins d'URL doivent être sécurisés et lesquels ne le doivent pas (le chemin /registration est configuré pour ne nécessiter aucune authentification. Tous les autres chemins doivent être authentifiés).

```
http.authorizeRequests() .antMatchers("/registration").permitAll()
```

Spring Security- Configuration de la classe SecurityConfig

```
import org.springframework.beans.factory.annotation.Autowired;
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;
@Configuration
@EnableWebSecurity
public class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {
@Autowired
private MyUserDetailsService userDetailsService;
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);}
@Override
protected void configure(HttpSecurity http) throws Exception {
   http.authorizeRequests() .antMatchers("/registration").permitAll()
.antMatchers("/getRevenuBrutProduit/{idProduit}/{startDate}/{endDate}").access("hasRole('SUPERADMIN')")
.antMatchers("/get**}").access("hasRole('ADMIN')")
.antMatchers("/retrieve-all-clients").access("hasRole('ADMIN')")
      .anyRequest()
      .authenticated()
      .and()
      .httpBasic().and().csrf().disable();
}}
```

L'ordre est important !

Spring Security- Configuration de la classe SecurityConfig

Configuration	Signification
La méthode configure(HttpSecurity http)	Configuration des utilisateurs authentifiés et autorisés
La méthode configure(AuthenticationManagerBuilder auth)	Dans laquelle Spring Security chargera les détails de l'utilisateur pour effectuer l'authentification et l'autorisation à travers l'implémentation de l'interface UserDetailsService
@EnableWebSecurity	pour activer la prise en charge de la sécurité Web de Spring Security et fournir l'intégration Spring MVC
.antMatchers("/registration").permitAll()	La page /registration ne demande pas une authentification
.antMatchers("/getRevenuBrutProduit/**").access ("hasRole ('SUPERADMIN')")	La page / getRevenuBrutProduit est accessible par le role superadmin
.antMatchers("/retrieve-all-clients").access("hasRole ('ADMIN')")	La page /retrieve-all-clients est accessible par le role admin
.passwordEncoder(bCryptPasswordEncoder);	Pour encoder le mot de passe
.csrf()	Utilisé pour empêcher la falsification de requêtes intersites, la protection CSRF est activée (par défaut).

Spring Security- Configuration de la couche Controller

- Il suffit par la suite d'exposer le service de registration :

```
@RestController
public class HomeController {
@Autowired
UserService userService;
@PostMapping("/registration")
    public String createNewUser( @RequestBody User user) {
        String msg="";
        User userExists = userService.findUserByUserName(user.getUserName());
        if (userExists != null) {
           msg="There is already a user registered with the user name provided";
        }        else {
            userService.saveUser(user);
            msg="OK";        }
      return msg;    }
…
```

TP –Ajout Admin et User

POST localhost:8090/SpringMVC/registration

Params Authorization Headers (10)

none form-data x-www-form-urle

```
{
"userName":"admin",
"email":"admin@gmail.com",
"password":"admin",
"name":"admin",
"lastName":"admin",
"roles":[{"role":"ADMIN"}]
```

POST localhost:8090/SpringMVC/registration

Params Authorization Headers (10) Body Pre-request Script Tests Settings

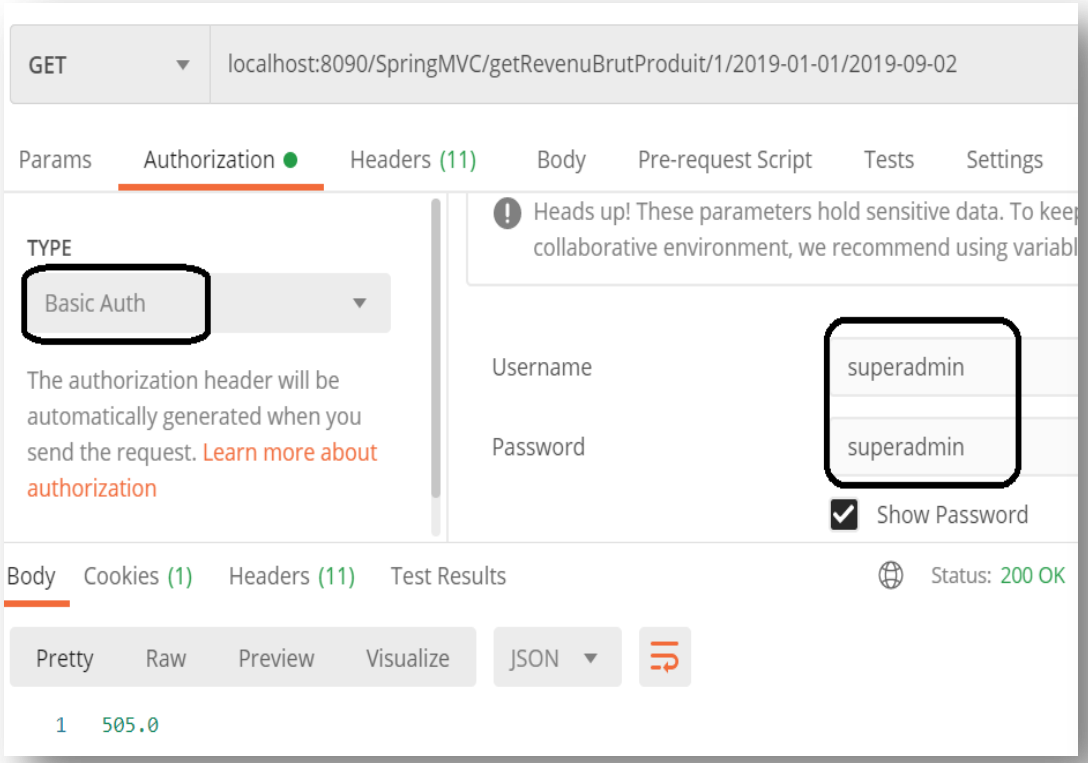
none form-data x-www-form-urlencoded raw binary GraphQL JSON

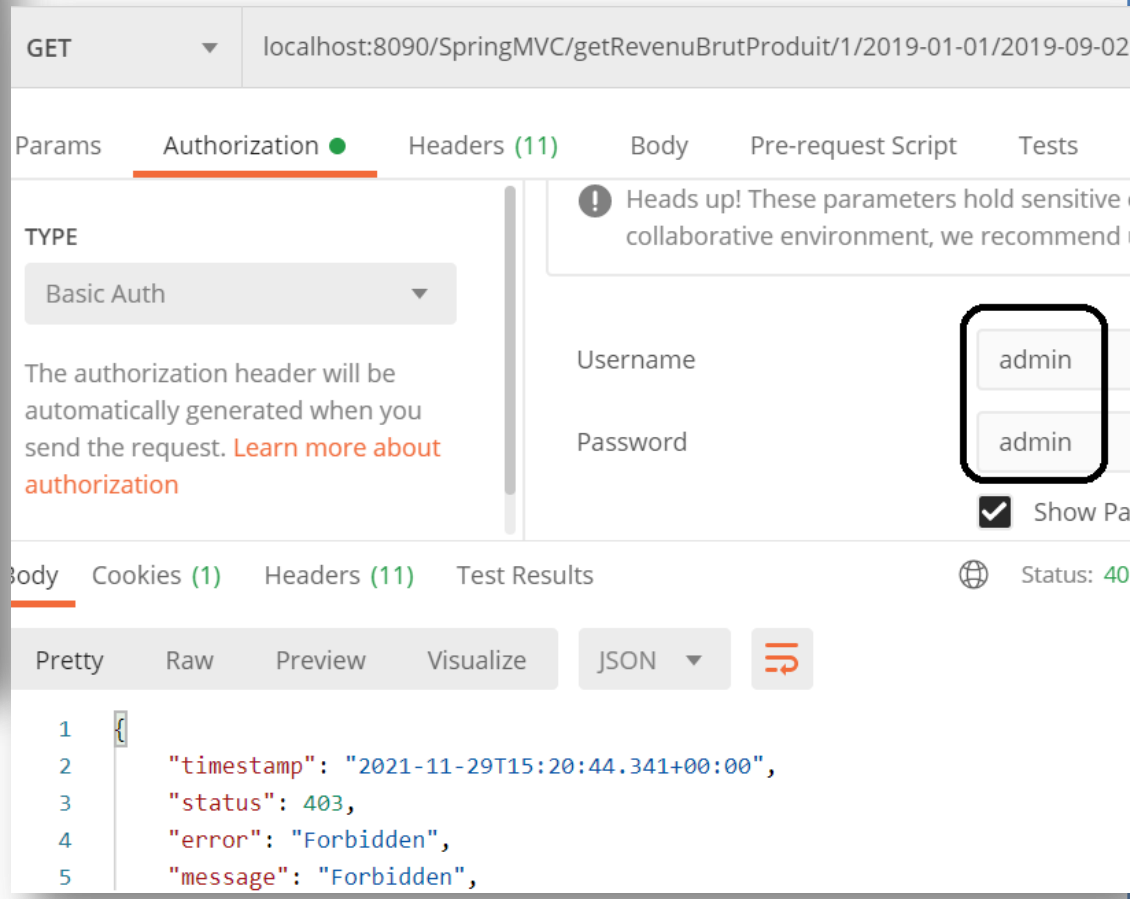
```
{
"userName":"superadmin",
"email":"superadmin@gmail.com",
"password":"superadmin",
"name":"superadmin",
"lastName":"superadmin",
"roles":[{"role":"SUPERADMIN"}]

}
```

id	active	email	last_name	name	password	user_name
1	1	admin@gmail.com	admin	admin	$2a$10$Rry2IZFLqilTngEraG8w.uPyJk3E6dDqP…	admin
3	1	superadmin@gmail.com	superadmin	superadmin	$2a$10$YCWTXnoxRljqt64tz9mioeyGj.F.X6L/kc…	superadmin

TP

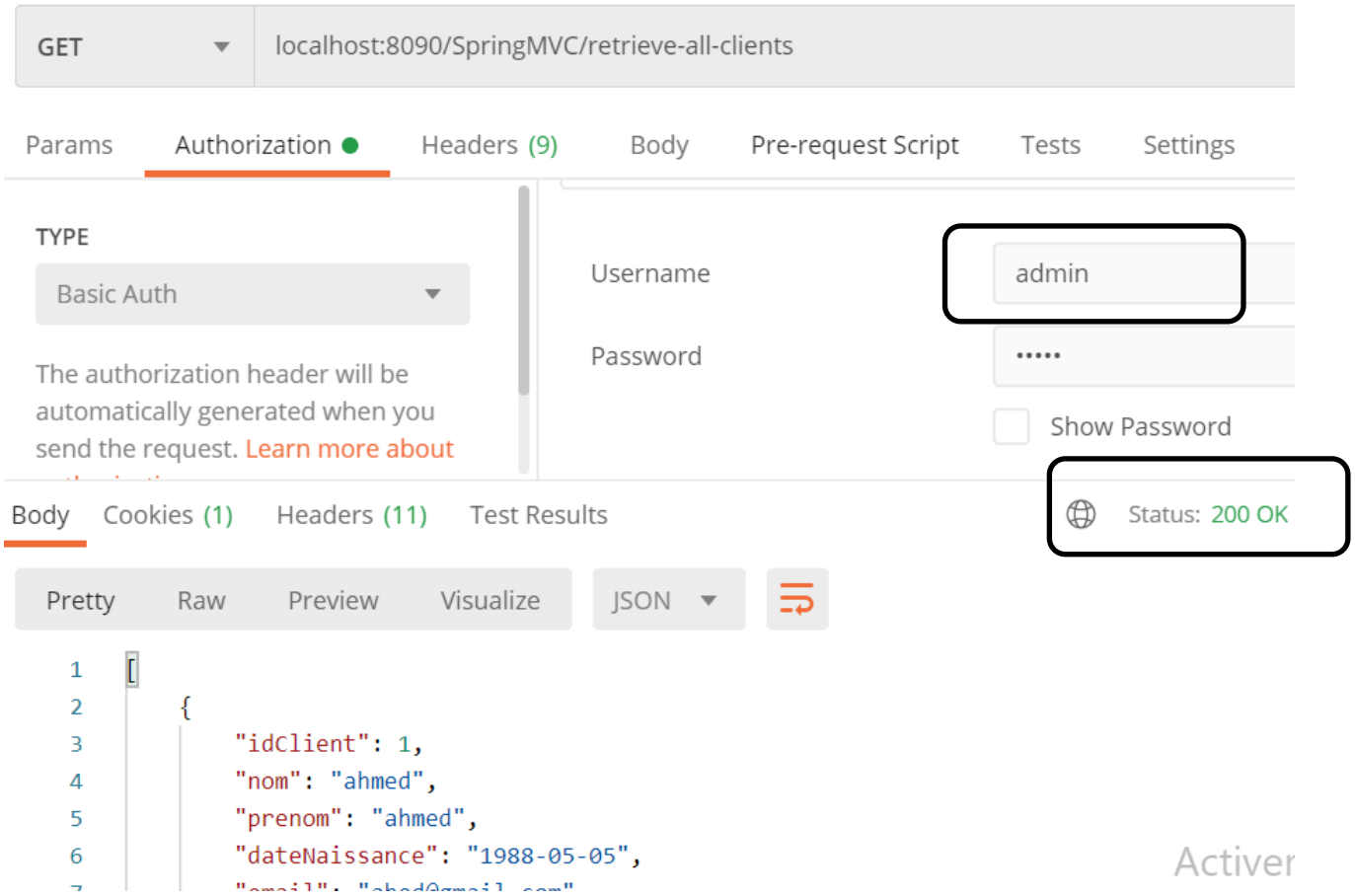
- L'accès au service /getRevenuBrutProduit n'est autorisé qu'au user dont le rôle est superadmin.





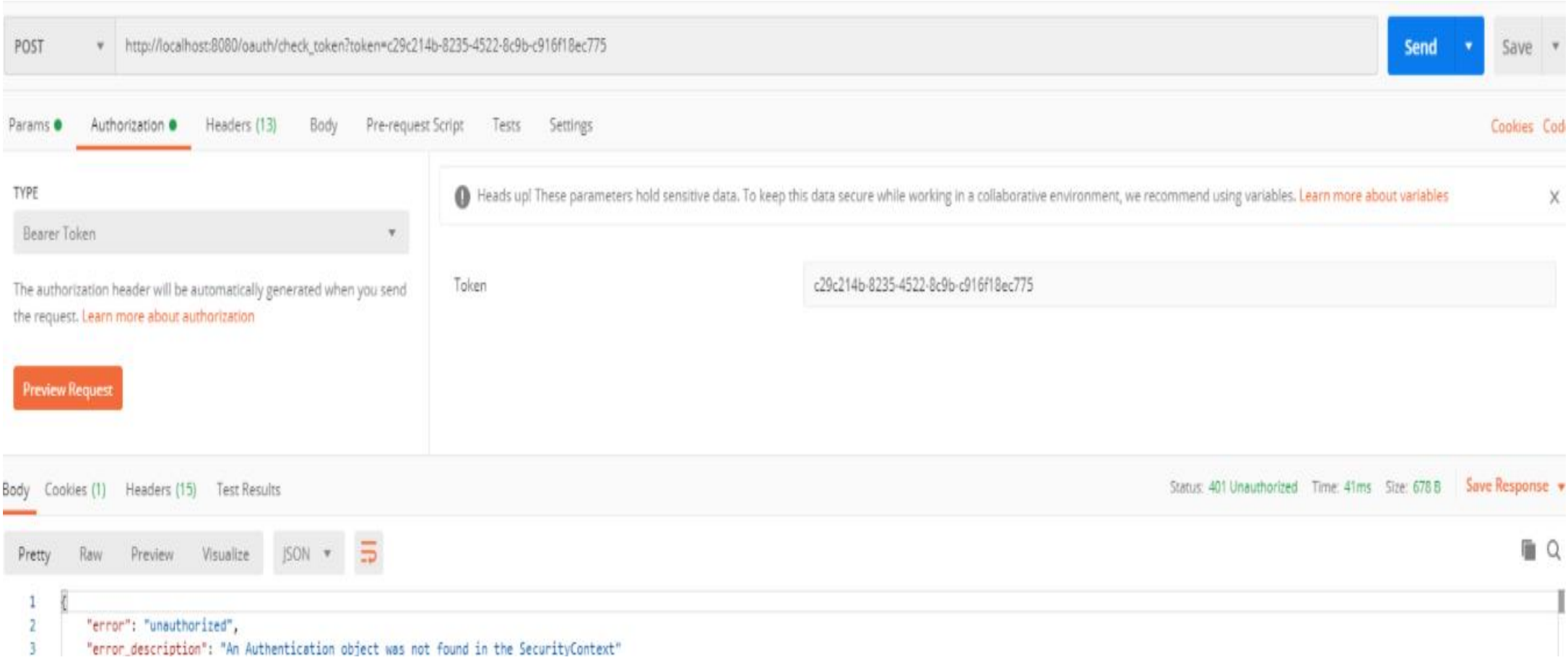
TP

- L'accès au service /retrieve-all-clients n'est autorisé qu'au user dont le rôle est admin.



TP

- Dans notre exemple, nous nous sommes basés sur la notion de basic authentification. Plusieurs autres cas existent pour assurer la sécurité de notre application tel que la génération des tokens.



Si vous avez des questions, n'hésitez pas à nous contacter :

Département Informatique
UP ASI
Bureau E204