# Sécuriser son application avec Spring Security

Max Devulder

## Présentation
### Introduction

- **Framework** de sécurité léger basé sur le protocole HTTP.
- Support d’autorisation afin de sécuriser les applications Spring
- Fourni avec les algorithmes les plus populaires.

## Installation
### Introduction

- Installation avec spring-boot:

Ajouter la dépendance « starter-security » dans le **pom.xml**

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

## Utilisation
### Introduction

- Etape 1 : Implémenter le bean User Spring :

**org.springframework.security.core.userdetails.User**

Classe Spring

```
public class DlabsSpringUser extends User {

private static final long serialVersionUID = -2836522345185404025L;

     public DlabsSpringUser(final String username, final String password, final Collection<?
     extends GrantedAuthority> authorities) {
        super(username, password, authorities);
     }

}
```

## Etape 2 : Implémenter la classe d'authentification

**org.springframework.security.authentication.AuthenticationProvider;**

Interface Spring

Override

Repo classique

Récupération user/pwd

Cas simple : Si login trouvé en BDD OK

```java
public class DlabsAuthenticationProvider implements AuthenticationProvider {

@Autowired
private PasswordBO passwordBO;

@Autowired
private UserRepository repo;

@Override
public Authentication authenticate(final Authentication authentication) throws AuthenticationException {
        final String userName = authentication.getName();
        final String password = authentication.getCredentials().toString();

        final UserDO utilisateurEntity = repo.findUserWithName(userName).orElse(null);
        if (utilisateurEntity != null /*&& passwordBO.matches(password, utilisateurEntity.getPassword())*/) {

                // Création d'un bean perso pour ajouter des valeurs.
                final List<GrantedAuthority> grantedAuths = new ArrayList<>();
                final DlabsSpringUser principal = new DlabsSpringUser(userName, password, grantedAuths);

                return new UsernamePasswordAuthenticationToken(principal, password, grantedAuths);
        }

        // Arrivé ici alors KO.
        return null;
}
```

## Etape 3 : Implémenter la configuration par défaut

**org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter**

```java
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired
private DlabsAuthenticationProvider authProvider;

@Override
protected void configure(final AuthenticationManagerBuilder auth) throws Exception {
   auth.authenticationProvider(authProvider);
}

@Override
protected void configure(final HttpSecurity http) throws Exception {

http.addFilterBefore(new CORSFilter(), BasicAuthenticationFilter.class);

// 1. Arrêt de la gestion Cors/Csrf/FrameOptions via Srping
// Gestion manuelle de la response http => CORSFilter.java
http.cors().and().csrf().disable();
http.headers().frameOptions().disable();

// 2. Protection de tout ce qui n'est pas derrière /public
// @formatter:off
http.authorizeRequests()
.antMatchers(HttpMethod.OPTIONS, "/**").permitAll()
.antMatchers("/public/**").permitAll()
.antMatchers("/**").authenticated()
.and().httpBasic();
// @formatter:on

}

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

Cf. slide précédent

Gestion CORS manuelle

Protocole HTTP :
Il faut toujours laisser passer les requêtes OPTIONS !

Tout ce qui est derrière /public est autorisé sans AUTH

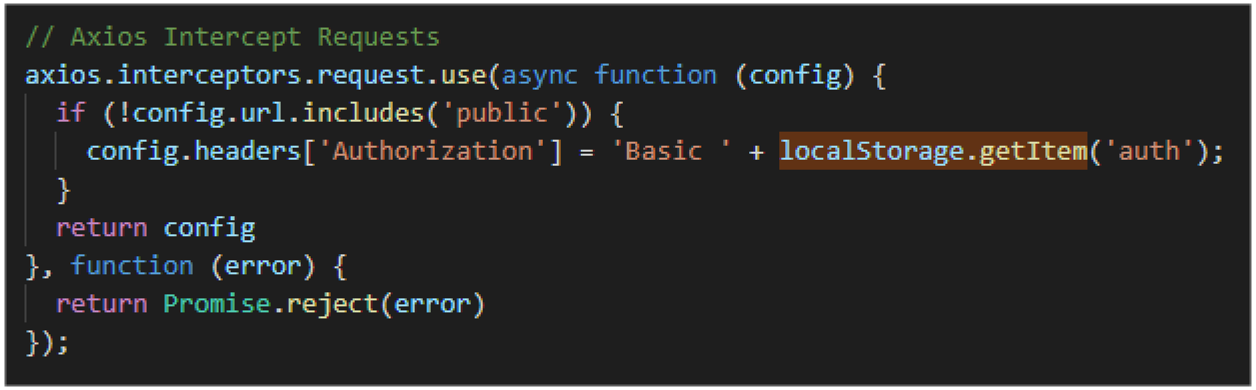
Sinon il faut être authentifié

Méthode de cryptage du mot de passe

# Utilisation
## Introduction

### Coté front

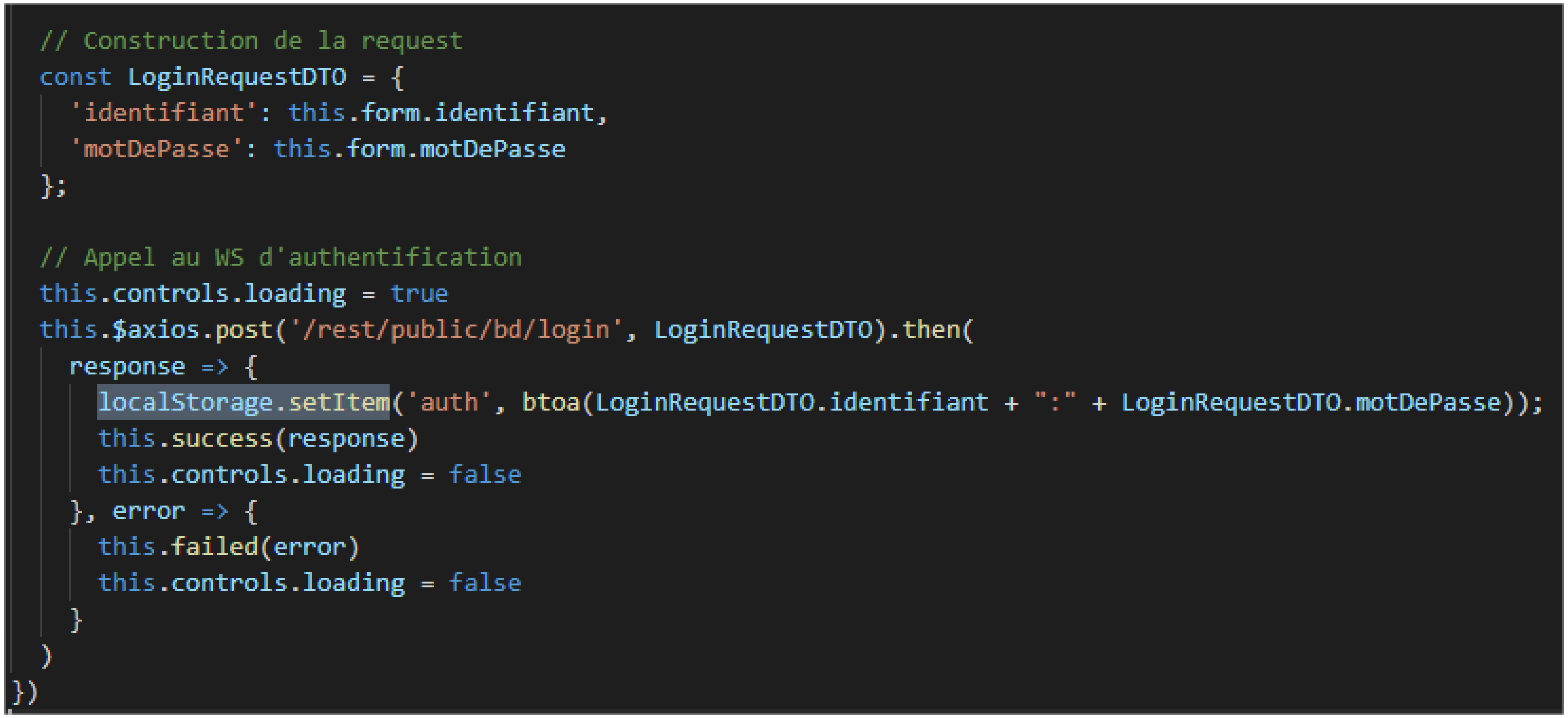
Envoi des credentials en base64, « Basic Auth »



```
// Axios Intercept Requests
axios.interceptors.request.use(async function (config) {
  if (!config.url.includes('public')) {
    config.headers['Authorization'] = 'Basic ' + localStorage.getItem('auth');
  }
  return config
}, function (error) {
  return Promise.reject(error)
});
```

# Utilisation
## Introduction

- Coté front

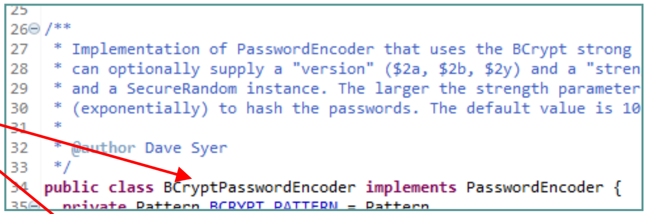


```
// Construction de la request
const LoginRequestDTO = {
  'identifiant': this.form.identifiant,
  'motDePasse': this.form.motDePasse
};

// Appel au WS d'authentification
this.controls.loading = true
this.$axios.post('/rest/public/bd/login', LoginRequestDTO).then(
  response => {
    localStorage.setItem('auth', btoa(LoginRequestDTO.identifiant + ":" + LoginRequestDTO.motDePasse));
    this.success(response)
    this.controls.loading = false
  }, error => {
    this.failed(error)
    this.controls.loading = false
  }
 )
})
```

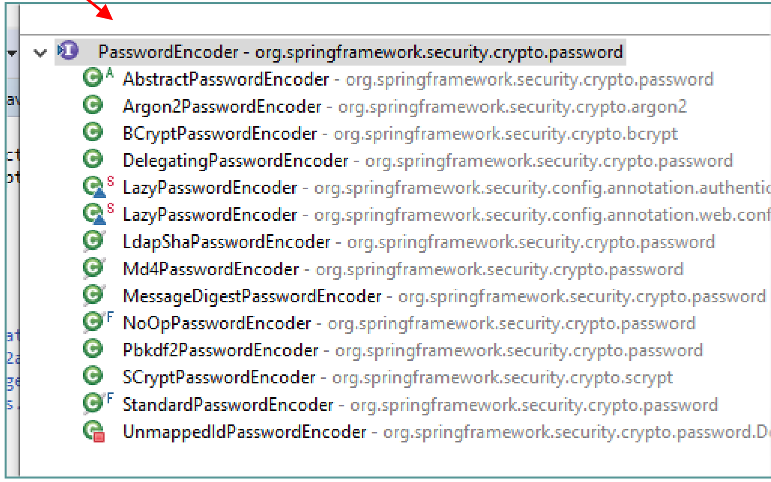
## Gestion du password en BCrypt

```
@Service
public class PasswordBO {

@Autowired
private BCryptPasswordEncoder passwordEncoder;

/**
 * Le password correspond il à celui crypté ?
 * @param BCryptFormat : Crypté
 * @param rawFormat : Clair
 * @return
 */
public Boolean matches(final String rawFormat, final String BCryptFormat) {
   return passwordEncoder.matches(rawFormat, BCryptFormat);
}

/**
 * Encode le mot de passe
 * @param BCryptFormat
 * @param textFormat
 * @return
 */
public String encode(final String textFormat) {
   return passwordEncoder.encode(textFormat);
}

}
```

Fourni par Spring



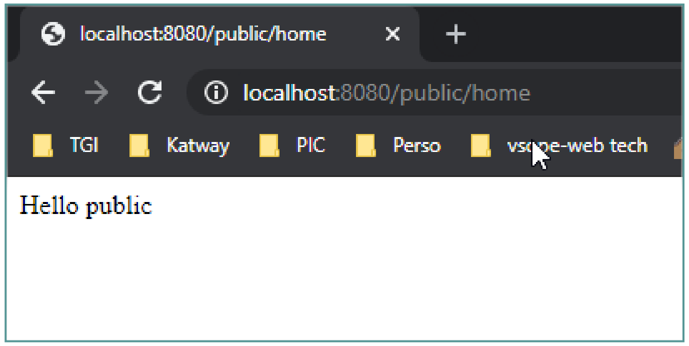
```
/**
 * Implementation of PasswordEncoder that uses the BCrypt strong
 * can optionally supply a "version" ($2a, $2b, $2y) and a "stren
 * and a SecureRandom instance. The larger the strength parameter
 * (exponentially) to hash the passwords. The default value is 10
 *
 * @author Dave Syer
 */
public class BCryptPasswordEncoder implements PasswordEncoder {
    private Pattern BCRYPT_PATTERN = Pattern
```



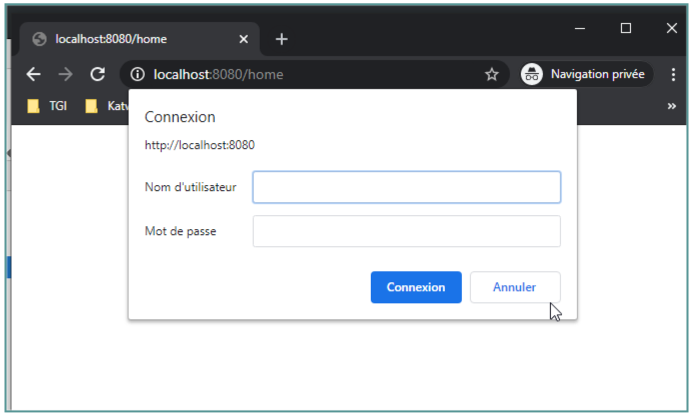
PasswordEncoder - org.springframework.security.crypto.password
- AbstractPasswordEncoder - org.springframework.security.crypto.password
- Argon2PasswordEncoder - org.springframework.security.crypto.argon2
- BCryptPasswordEncoder - org.springframework.security.crypto.bcrypt
- DelegatingPasswordEncoder - org.springframework.security.crypto.password
- LazyPasswordEncoder - org.springframework.security.config.annotation.authentic
- LazyPasswordEncoder - org.springframework.security.config.annotation.web.conf
- LdapShaPasswordEncoder - org.springframework.security.crypto.password
- Md4PasswordEncoder - org.springframework.security.crypto.password
- MessageDigestPasswordEncoder - org.springframework.security.crypto.password
- NoOpPasswordEncoder - org.springframework.security.crypto.password
- Pbkdf2PasswordEncoder - org.springframework.security.crypto.password
- SCryptPasswordEncoder - org.springframework.security.crypto.scrypt
- StandardPasswordEncoder - org.springframework.security.crypto.password
- UnmappedIdPasswordEncoder - org.springframework.security.crypto.password.D

## Résultat

- Notre API est sécurisée !



Hello public



Connexion

http://localhost:8080

Nom d'utilisateur

Mot de passe

```
@RestController
@RequestMapping(value = "/public/home")
@Transactional
public class HomePublicBD {

@RequestMapping(method = RequestMethod.GET)
public String sayHello() {
   return "Hello public";
}

}
```

```
@RestController
@RequestMapping(value = "/home")
@Transactional
public class HomePrivateBD {

@RequestMapping(method = RequestMethod.GET)
public String sayHello(final Principal principal) {
   return "Hello " + principal.getName();
}

}
```

Le user est disponible au besoin au travers de l'objet Principal

## Utilisation
### Introduction

- Extrait code source:

  https://gitlab.com/ulco-jee/dlabs/-/tree/master