

ENSET-M	II-BDCC 2
Java EE et middlewares	ELAAMIRI Essadeq

Activité Spring MVC, Spring Data et Spring Security

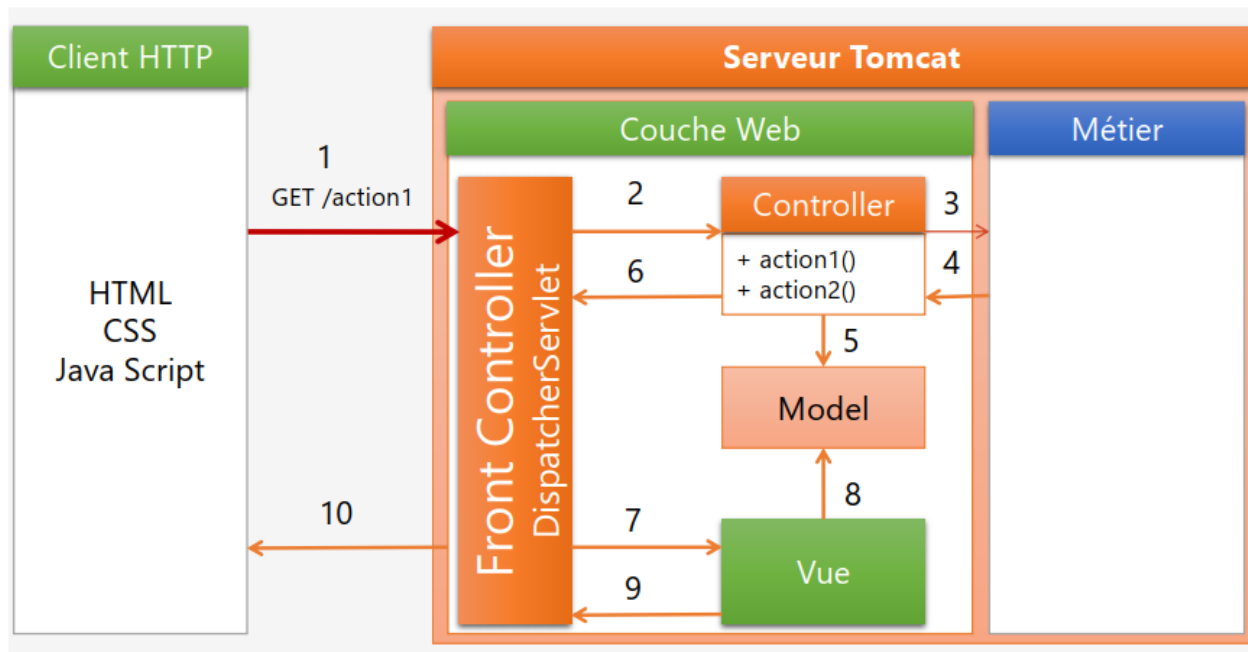
Table des matières

Activité Spring MVC, Spring Data et Spring Security	1
Spring MVC, WEB	2
Thymeleaf forms	5
Thymeleaf layouts	7
Formulaires	8
Validation	10
Spring Scurity et Authentification :	13
JDBC Authentication	22
Gestion des utilisateurs en utilisant la stratégie USER DETAILS SERVICE	25
Captures d'écrans :	34

Dans cette activité on va développer une application de gestion des patient, en utilisant Spring et Thymeleaf. L'application va utiliser la base de données MySQL.

Voici une documentation et au même temps un compte rendu dont j'ai essayé de résumer les informations sur l'application et le processus de sa réalisation.

Spring MVC, WEB

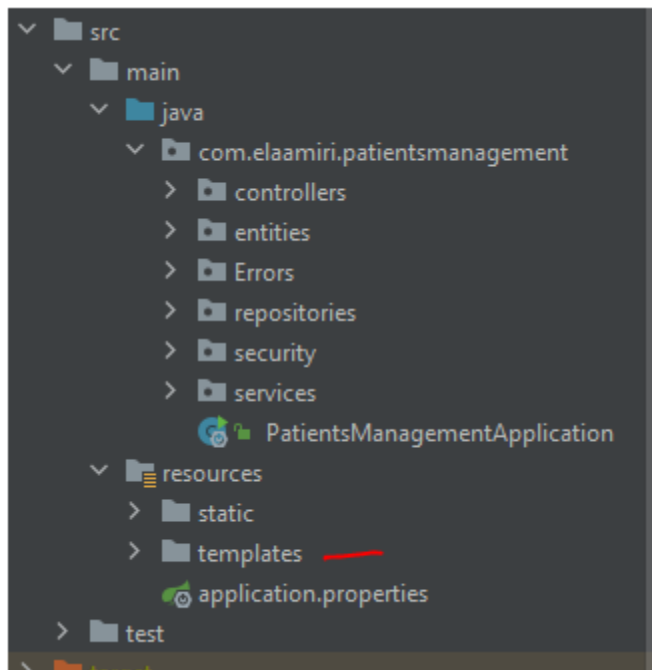


1. Le client envoie une requête HTTP de type GET ou POST...
2. Toutes les requêtes HTTP sont traitées par un contrôleur frontal fourni par Spring.
 - C'est une servlet nommée DispatcherServlet .
 - Chaque action de l'URL, DispatcherServlet devrait exécuter une opération associée à cette action.
 - Cette opération est implémentée dans une classe appelée Controller qui représente un sous contrôleur ou un contrôleur secondaire
3. Le sous contrôleur exécute le traitement associé à l'action en faisant appel à la couche métier et récupère le résultat.
4. 3 ▲▲
5. Le sous contrôleur stocke le résultat dans le modèle fourni par Spring MVC.
6. Le sous contrôleur retourne le nom de la vue et le modèle à DispatcherServlet.

7. Le contrôleur frontal `DispatcherServlet` fait appel à la vue et lui transmet le modèle.
8. La vue récupère les résultats à partir du modèle et génère un rendu HTML qui est retourné à `DispatcherServlet`
 - Pour générer du code HTML, on peut utiliser JSP, mais il est déconseillé car il y a mieux : les Moteurs de templates
 - Spring MVC offre des moteurs de templates comme Thymeleaf, FreeMaker, Mustach, qui permettent de faciliter la génération du code HTML coté serveur
9. ▲▲
10. `DispatcherServlet` envoie la réponse HTTP au client. Cette réponse http contient le code HTML générée par la vue.

Dépendances :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```



- Le repertoire 'Templates' va contenir nos vues Thymeleaf.

Contrôleur :

Une classe qui va gérer les requêtes http, elle utilise l'annotation @Controller (figure 2 suivante).

```
@GetMapping("/addNewPatient")
public String addNewPatient(Model model){
    // thymeleaf will access this empty object for binding from data ?
    Patient patient = new Patient();
    model.addAttribute(attributeName: "patientObject", patient);
    return "addNewPatient";
}
```

La fonction addNewPatient(), et liée à la route '/addNewPatient', c a d, elle va être appelée lorsque une requête de type GET, envoyer sur ce lien (@GetMapping()).

L'objet Model : Permet d'envoyer et récupérer des données à et depuis la vue.

La fonction retourne le nom de vue à être rendu.

```

@Controller
//@RestController in cas we're creating API
@AllArgsConstructor
public class PatientController {
    private final int MAXPAGES_INRANGE = 5;

    /**
     * Spring MVC controller
     */
    private PatientService patientService;

    @GetMapping("/") // hey you! if a http get method call the path URL "/", call this function
    // adding pagination
    public String displayPatientsList(Model model,
                                     @RequestParam(name = "page", defaultValue = "0") int page,
                                     @RequestParam(name = "size", defaultValue = "5") int size,
                                     @RequestParam(name = "searchKeyWord", defaultValue = "")
                                     String searchKeyWord){
        Page<Patient> patientPage;
        // if there a searchKeyWord so search with it if not get all
        if(!searchKeyWord.isEmpty() && searchKeyWord != null){
            patientPage = patientService.getPatientsListByKeyWord(searchKeyWord, PageRequest.of(page, size));

            model.addAttribute( attributeName: "searchKeyWord", searchKeyWord);
        }
    }
}

```

Récupérer les valeurs des paramètres de requêtes (Query Strings).

On peut utiliser les mêmes noms de paramètres dans les arguments de la fonctions, Spring va les mapper sans le besoin d'utiliser l'annotation `@RequestParam()`.

Pour le mapping des requêtes on peut utiliser l'annotation :

```
@RequestMapping(value="/chercher",method=RequestMethod.GET)
```

Thymeleaf forms

► Thymeleaf résumé : <https://github.com/engma/thymeleaf-cheat-sheet>

Dépendance :

```

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

```

Namespace:

```
<html
  lang="en"
  xmlns:th="http://www.thymeleaf.org"
```

Afficher la valeur d'un objet dans un tag HTML:

```
<span class=" " th:text="${'Total number: ' + totalElements}"></span>
```

Référencier à un lien dans href :

```
<a class="" th:href="@{/apiv1/patients}" >get data as json (API)</a>
```

Donner la valeur d'un attribut de l'objet au valeur d'un Tag HTML :

```
<input type="text" class="" th:value="${searchKeyWord}">
```

Boucler dans les éléments d'une liste (patient est objet de la liste patientsList):

```
<tr th:each="patient : ${patientsList}">
```

Donner les paramètres (Query strings) dans un lien:

```
<a th:href="@{/editPatient/{id}(id=${patient.getId()},page=
${currentPage},searchKeyWord=${searchKeyWord})}" class="btn btn-sm
btn-primary"><i class="fas fa-edit"></i></a>
```

Lien

{id} paramètre [id=\${patient.getId()}] sa valeur]

Page= (nom de Query String)

\${currentPage} (la valeur du Query String)

Ne pas afficher un tag html si un condition et vrai :

```
<!-- show it unless (except if ) noSearchResultFoundMsg is null-->
<!-- Show it, except if noSearchResultFoundMsg == null, do not-->
<p class="alert alert-info" th:unless="${noSearchResultFoundMsg ==
null}" th:text="${noSearchResultFoundMsg}">
```

Ajouter un attribut avec Thymeleaf:

```
<button class=" " th:attr="data-deleteLink =
${'/deletePatient/'+patient.getId()+'?page='+currentPage+'&searchKeyWo
rd='+ (searchKeyWord == null ? '' : searchKeyWord)}" ><i class="fas fa-
trash"></i></button>
```

Jouer avec les classes HTML:

```
<a th:text="${page}"
  th:href="@{/page= ${page}, searchKeyWord=${searchKeyWord}}"
  th:class="'btn btn-sm mx-1 ' + ${page == currentPage ? 'btn-dark'
: 'btn-outline-dark'}" >

</a>
```

Thymeleaf layouts

Pour éviter la redondance dans nos page Thymeleaf on peut utiliser la fonctionnalité 'Thymeleaf layouts', qui consiste à utiliser des Templates communs, et y appeler les autres pages.

Installer l'indépendance :

```
<dependency>
  <groupId>nz.net.ultraq.thymeleaf</groupId>
  <artifactId>thymeleaf-layout-dialect</artifactId>
  <version>3.1.0</version>
</dependency>
```

La page Template qui va contenir les contenus communs entre les autres pages (navbar, footer ...).

```
<!DOCTYPE html>
<html lang="en"
  xmlns:th="http://www.thymeleaf.org"
  xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
>
...
<body>
  ...
  <div class=" d-flex justify-content-center align-items-center">
    <section layout:fragment="indexPageContent"></section>
  </div>
</body>
```

La section signalée dans le code, est où le contenu des pages va être injectée, voici comment le contenu à injecter doit être signaler, il faut dans un premier temps de déclarer le Template à utiliser, puis la section à être injectée :

```
<html
    lang="en" xmlns:th="http://www.thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
    layout:decorate="template"
>
...

<div class="container card mb-5" layout:fragment="indexPageContent">
... <!--contenu à être injecté -->
</div>
```

Dans ce cas tous les pages ayant un contenu avec 'layout:fragment="indexPageContent', ce contenu-là va être injecté au fragment de même nom dans le Template.

Formulaires

```
@GetMapping("/addNewPatient")
public String addNewPatient(Model model) {
    // thymeleaf will access this empty object for binding from data ?
    Patient patient = new Patient();
    model.addAttribute("patientObject", patient);
    return "addNewPatient";
}
```

La fonction ci-dessus est une fonction du Contrôleur de l'entité Patient, elle va être invoquée lorsqu'une requête 'GET' envoyée sur le lien '/addNewPatient', et retourne la vue '/templates/ addNewPatient'.html'.

Spring cherche les vues par défaut dans le répertoire : '/ressources/templates'.

Voici le formulaire de l'ajout :

Model : c'est un objet qui va nous permettre d'envoyer des attributs vers les vues, on peut donc les récupérer et les utiliser. Donc ce cas envoie un objet de type Patient, Spring va par la suite le remplir en mappant ses propriétés avec les champs du formulaire.

model.addAttribute("patientObject", patient);

Dans le formulaire on spécifier l'objet concerné par '**th :object**', la méthode utilisé est de type post, elle va envoyer vers les lien (la fonction) 'saveNewPatient'.

Dans le champs 'firstname' on a spécifier l'attribut auquel il est lié, en utilisant '**th :field**'.

```
<form th:action="@{saveNewPatient}" th:object="${patientObject}" method="POST" action="#">
  <div class="mb-3">
    <label for="firstName" class="form-label">First name</label>
    <input type="text"
      name="firstName"
      class="form-control"
      id="firstName"
      th:field="${firstName}"
      th:value="${patientObject.firstName}">
    <span th:errors="${patientObject.firstName}"></span>
  </div>
  <div class="mb-3">
    <label for="lastName" class="form-label">Last name</label>
    <input type="text" name="lastName" class="form-control" id="lastName">
  </div>
</form>
```

```
@PostMapping("/saveNewPatient") // hey you! if a post request this url, call the following function
public String saveNewPatient(@ModelAttribute("patientObject") Patient patient,
    @RequestParam(name = "page", defaultValue = "0") int page,
    @RequestParam(name = "size", defaultValue = "5") int size,
    @RequestParam(name = "searchKeyword", defaultValue = "") String searchKeyword){
    patientService.insertPatient(patient);
    return "redirect: /?page="+page+"&size="+size+"&searchKeyword="+searchKeyword;
}
```

Pour afficher la valeur d'un attribut de l'objet :

```
th:value="${patientObject.firstName}"
```

Les erreurs relatives au validation du champs vont être affichés dans le **span** en utilisant '**th :errors**'. Les erreurs issues de validation sur les attributs dans l'entité (Spring Data).

Pour les champs de types (true/ false) :

```
<input type="checkbox" name="malade" th:checked="${patient.malade}">
```

La fonction de l'insertion –ci-dessus-, reçoit l'objet patient qu'est rempli par Spring par le Model aussi la page, la taille de page (nombre des entrées) et le mot clef de la recherche par la requête (Query Strings).

►► **Erreur 400 ??** penser aux problèmes du format (date, time ...).

```
@Temporal(TemporalType.DATE)
@DateTimeFormat(pattern = "yyyy-MM-dd")
private Date birthDate;
```

Validation

Pour utiliser la validation il faut télécharger :

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

Les annotations de validations les plus utilisées :

@NotNull	field must not be null.
@NotEmpty	list field must not empty.
@NotBlank	string field must not be the empty string (i.e. it must have at least one character).
@Min and @Max	numerical field is only valid when it's value is above or below a certain value.
@Pattern	string field is only valid when it matches a certain regular expression.
@Email	to say that a string field must be a valid email address.
@AssertTrue	validates that the annotated property value is true.
@Size	validates that the annotated property value has a size between the attributes min and max; can be applied to String, Collection, Map, and array properties.
@Past and @Future	The attribute must be a date in the future or in the past
For more	https://docs.oracle.com/javaee/7/api/javax/validation/constraints/package-summary.html

```

@NotNull
@NotBlank
private String id;
@NotNull
@NotBlank
@Size(min = 2, max = 30)
private String firstName;
@NotNull
@NotBlank
@Size(max = 30)
private String lastName;
@Past // date in the past
@Temporal(TemporalType.DATE)
@DateTimeFormat(pattern = "yyyy-MM-dd")
private Date birthDate;
>Email // a valid email, we can use @pattern also
private String email;

```

Hibernate utilisera ces validations avant d'exécuter les requêtes SQL concernées.

Donc pour dire au Spring MVC d'utiliser les validations, dans les fonctions du contrôleur, il suffit d'ajouter l'annotation : **@Valid** avant l'objet qui va être récupéré par la requête POST.

► **Validation** : ajouter la dépendance + les annotations de validation dans les entités + les validations dans les contrôleurs + **'th:errors'** dans les pages Thymeleaf.

► l'annotation **@Bean** au-dessus d'une fonction dit au Spring d'exécuter celle-ci au démarrage.

Dans le contrôleur :

L'objet **BindingResult** a plusieurs informations sur les erreurs de validation.

En cas d'erreurs on fait une 'Redirect' vers le formulaire de l'ajut, pour afficher les erreurs de validation, et dans le cas contraire on fait le 'Redirect' vers la page d'accueil.

```
@PostMapping("/saveNewPatient") // hey you! if a post request this url, call the follow
public String
saveNewPatient(@ModelAttribute("patientObject") @Valid Patient patient,
                BindingResult bindingResult,
                @RequestParam(name = "page", defaultValue = "0") int page,
                @RequestParam(name = "size", defaultValue = "5") int size,
                @RequestParam(name = "searchKeyWord", defaultValue = "")
                String searchKeyWord){

    // tests
    if(bindingResult.hasErrors()){
        return "addNewPatient";
    }
    else{
        patientService.insertPatient(patient);
        return "redirect:/?page="+page+"&size="+size+"&searchKeyWord="+searchKeyWord;
    }
}
```

Récupérer la valeur d'un Query String.

Vue '/ressources/templates/addNewPatient'.

Add a new patient

First name

size must be between 2 and 30
must not be blank

Last name

must not be blank

Email address

We'll never share your email with anyone else.
must not be blank

Birth date

must not be null

```
return  
"redirect: /?page="+page+"&size="+size+"&searchKeyWord="+searchKeyWord;
```

C'est une redirection vers le chemin "/" avec des paramètres de requêtes.

► au lieu d'utiliser '**th :field**' on peut juste nommer les champs avec les mêmes noms que les attributs de l'objet.

Spring Scurity et Authentification :

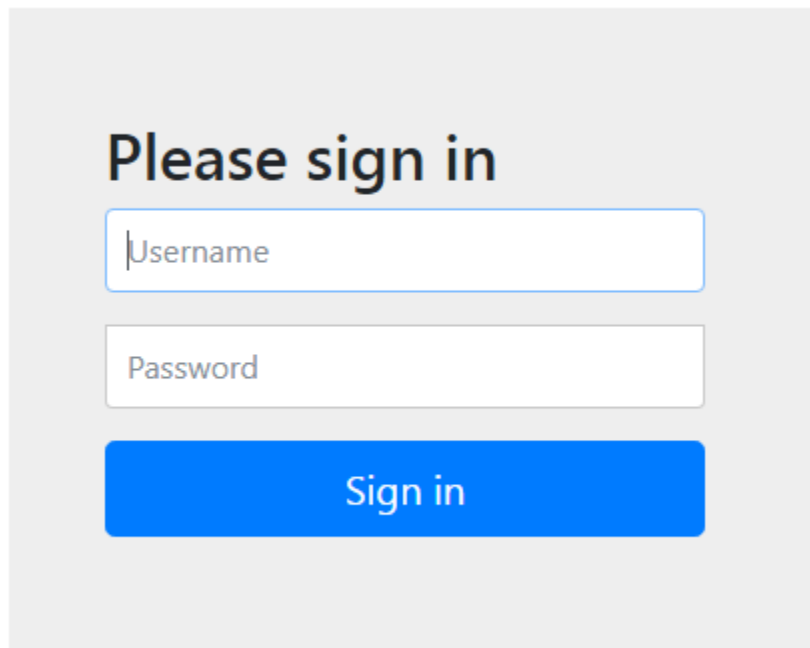
On va créer un système d'authentification :

- Consultation globale avec un compte 'USER'.
- Consultation et Gestion globale avec un compte 'ADMIN'

Installer la dépendance :

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

Spring Security a une configuration par défaut, au démarrage il ajout une couche de sécurité à l'application (Il démarre un **filtre** sur la requête avant d'arriver au contrôleur) :



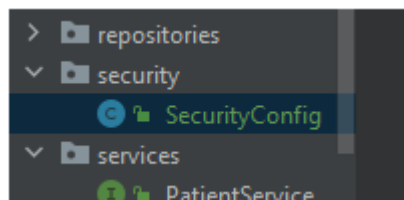
The image shows a simple login interface. At the top, it says "Please sign in". Below this, there are two input fields: the first is labeled "Username" and the second is labeled "Password". Both fields are empty. Below the password field is a blue button with the text "Sign in" in white.

Il nous donne le mot de passe dans le console :

```
Using generated security password: 55b4d43d-d03f-48cd-8c20-3b5822f5ce1d
```

```
Username : user
```

Pour configurer Spring Security on aura besoin d'une classe de configuration qui hérite de la classe '**WebSecurityConfigurerAdapter**'.



La classe doit utiliser l'annotation `@configuration`, pour dire au Spring que cette classe doit être instancié au premier lieu.

Et l'annotation `@EnableWebSecurity`, pour activer la sécurité web.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {

    }
}
```

Dans la fonction `configure` qui reçoit **AuthenticationManagerBuilder** on va configurer comment Spring Security va rechercher les utilisateurs (utilisateurs depuis la base de données, mémoire (Memory user), ou bien de l'annuaire de l'entreprise (Active directory)).

★ InMemoryUsers

```
@Override // Spécifier la stratégie avec laquelle Spring Sec va
// chercher les utilisateurs autorisés
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    // inMemory authentication = les utilisateurs à utiliser l'application
    // seront stockés dans la mémoire (utiles pour les tests)
    auth.inMemoryAuthentication() InMemoryUserDetailsManagerConfigurer<AuthenticationManagerBuilder>
        .withUser( username: "user1") UserDetailsManagerConfigurer<B, C>.UserDetailsBuilder
        .password("0000user")
        .roles("USER")
        .and() InMemoryUserDetailsManagerConfigurer<AuthenticationManagerBuilder>
        .withUser( username: "admin") UserDetailsManagerConfigurer<B, C>.UserDetailsBuilder
        .password("admin0000")
        .roles("USER", "ADMIN");
    //auth.inMemoryAuthentication() ... on peut reutiliser ...
}
```

Dans la fonction `configure` qui reçoit **HttpSecurity** on va spécifier les droits d'accès.

Spring utilise des algorithmes de hachage pour hacher les mots de passes, il faut donc lui indiquer quel algorithme on va utiliser pour encrypter les mots de passes :

Pour lui dire je ne vais pas utiliser un algorithme :

```
.password("{noop}0000user") // noop = no encryption needed
```

Spring dans ce cas il va pas encrypter le mot de passe.

Pour utiliser l'encodage des mots de passe, on va créer une méthode qui retourne un objet PasswordEncoder.

```
@Bean
// exécuté au démarrage, et place
// l'objet retourné dans le contexte
// comme Spring Bean (il peut être injecté n'importe où)
PasswordEncoder getPasswordEncoder(){
    // retourner le type d'encodage
    return new BCryptPasswordEncoder();
}
```

On utilise le PasswordEncoder.

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    PasswordEncoder passwordEncoder = getPasswordEncoder();
    // inMemory authentication = les utilisateurs à utiliser l'application
    // seront stockés dans la mémoire (utiles pour les tests)
    auth.inMemoryAuthentication() InMemoryUserDetailsManagerConfigurer<AuthenticationMan
        .withUser( username: "user1") UserDetailsManagerConfigurer<B, C>.UserDetailsBuild
        .password(passwordEncoder.encode( rawPassword: "0000user"))
        .roles("USER")
        .and() InMemoryUserDetailsManagerConfigurer<AuthenticationManagerBuilder>
        .withUser( username: "admin") UserDetailsManagerConfigurer<B, C>.UserDetailsBuild
        .password(passwordEncoder.encode( rawPassword: "admin0000"))
        .roles("USER", "ADMIN");
    //auth.inMemoryAuthentication() ... on peut réutiliser ...
    /*auth.inMemoryAuthentication()
        .withUser("user1")
        .password("{noop}1234") // // noop = no encryption needed
        .roles("USER");*/
}
```

Pour utiliser Thymeleaf avec Spring Security, il faut ajouter une dépendance et un namespace :

Info : <https://www.thymeleaf.org/doc/articles/springsecurity.html>

<https://github.com/thymeleaf/thymeleaf-extras-springsecurity>

```
<html xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
```

Pour gérer la contextualisation : ??

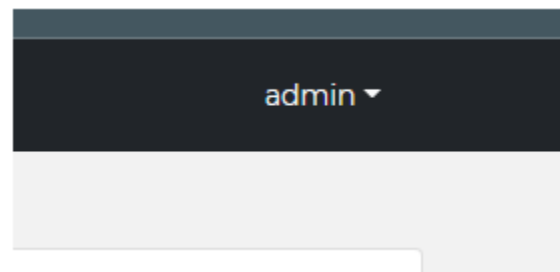
```
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

```
// get spring version
System.out.println(SpringVersion.getVersion()); // 5.3.16
```

```
<!-- tempale.html-->
<html lang="en"
  xmlns:th="http://www.thymeleaf.org"
  xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
  xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
>
```

Afficher l'utilisateur authentifié.

```
<a class=" "
  data-bs-toggle="dropdown" href="#"
  sec:authentication="name"
>
  <i class="fas fa-user"></i>
</a>
```



Les liens vers logout et login par défaut:

```
<a class="dropdown-item" th:href="@{/login}">Login</a>
<a class="dropdown-item" th:href="@{/logout}">logout</a>
```

Ajouter les restrictions :

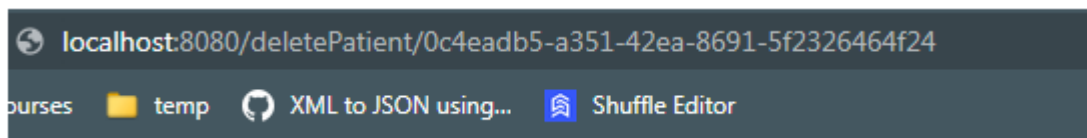
```
<td sec:authorize="hasAnyRole('ADMIN')">
  <a th:href=""><i class="fas fa-edit"></i></a>
  <button><i class="fas fa-trash"></i></button>
</td>
```

Email	Action
emosedale1n@dropbox.com	/
scornes2b@who.int	
kpickup1a@bloglovin.com	
dbiggs2o@hexun.com	
afrier2g@pen.io	

Si l'utilisateur n'a pas le rôle 'ADMIN', il ne va pas avoir le droit de voir ou utiliser les boutons de modification, ou suppression.

○○ Problème :

Si on a par exemple écrit :



Même si on est authentifié en tant qu'USER, mais l'entrée est supprimée. Car cette 'sec:authorize="hasAnyRole('ADMIN')"' ne rétracte pas l'action, mais juste l'affichage.

Il faut spécifier les droits dans la configuration :

Info : <https://askcodez.com/quand-utiliser-antmatcher-de-spring-security.html>

```

public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override // spécifier les droits d'accès
    protected void configure(HttpSecurity http) throws Exception {
        // hey Spring, je veux utiliser un 'form' d'authentification
        http.formLogin(); // default login form
        //http.formLogin().loginPage("/login"); // my own login form
        // droits d'accès
        http.authorizeRequests().antMatchers( ...antPatterns: "/editPatient",
            , "/deletePatient/**", "/saveEditedPatient/**"
            , "/saveNewPatient/**", "/addNewPatient/**")
            .hasRole("ADMIN");

        // toutes les requets nécessite une authentification
        http.authorizeHttpRequests().anyRequest().authenticated();
    }
}

```

Et quand je tente de faire le même truque de supprimer une entrée par le lien :

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Apr 03 08:33:12 WET 2022

There was an unexpected error (type=Forbidden, status=403).

Forbidden

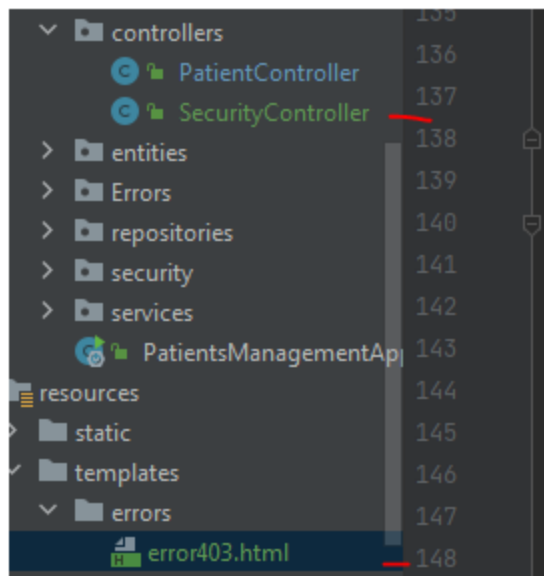
Pour configurer les messages d'erreurs dans la fonction configure (HttpSecurity) :

```

// configure errors (/error403 is a vue)
http.exceptionHandling().accessDeniedPage("/error403");

```

On doit donc créer un contrôleur et une vue.



```
@Controller
public class SecurityController {

    @GetMapping("/error403")
    public String accessDeniedError() {
        return "errors/error403";
    }
}
```

localhost:8080/addNewPatient

temp XML to JSON using... Shuffle Editor Getting Started wit...



► Il 'est toujours préféré de commencer par les spécifications des exigences, avant de masque les choses sur la vue.

```
@Override // spécifier les droits d'accès
protected void configure(HttpSecurity http) throws Exception {
    // hey Spring, je veux utiliser un 'form' d'authentification
    http.formLogin(); // default login form
    //http.formLogin().loginPage("/login"); // my own login form
    // droits d'accès
    http.authorizeRequests().antMatchers(...antPatterns: "/home").permitAll(); // for all |
    http.authorizeRequests().antMatchers(...antPatterns: "/editPatient/**"
        , "/deletePatient/**", "/saveEditedPatient/**"
        , "/saveNewPatient/**", "/addNewPatient/**")
```

La page '/home' est accessible par tous (authentifier ou non).

Ne pas afficher un élément que si l'utilisateur est authentifié.

```
<ul class="navbar-nav" sec:authorize="isAuthenticated()">
```

```
<a class="dropdown-item" sec:authorize="isAuthenticated()"
th:href="@{/logout}">logout</a>
```

Afficher un élément si l'utilisateur n'est pas authentifié.

```
<a class="dropdown-item" sec:authorize="isAnonymous()"
th:href="@{/login}">Login</a>
```

Info: <https://bushansirgur.in/everything-need-to-know-about-matchers-methods-in-spring-security/>

► Il' est préférable d'organiser les routes qui nécessite le rôle 'ADMIN', par exemple comme '/admin/...', de même pour le rôle 'USER'.

Pour être facile de les spécifier dans la fonction antMatchers().

```
zeRequests().antMatchers(...antPatterns: "/admin/**").hasRole("ADMIN");
zeRequests().antMatchers(...antPatterns: "/user/**").hasRole("USER");
```

```
// toutes les requets nécessite une authentification
http.authorizeHttpRequests().anyRequest().authenticated();
```

JDBC Authentication

Les utilisateurs seront stockés dans la base de données :

Créer les utilisateurs et leurs rôles :

```
create table users (
  username varchar(15) PRIMARY KEY NOT NULL,
  password varchar(255) not null, # pour que le hachage est grand
  isActive int(1) # 0/1
);
```

```
create table role (
  roleName varchar(30) PRIMARY KEY NOT NULL
);
```

```
insert into users
VALUES
('user2', 'user2', 0),
('user', 'user', 0),
('admin', 'admin', 0);
```

```
CREATE TABLE `patients-management-crud1`.`user_role`
  `username` VARCHAR(15) NOT NULL ,
  `roleName` VARCHAR(30) NOT NULL ,
  PRIMARY KEY (`username`, `roleName`)
) ENGINE = InnoDB;
```

```
insert into user_role
VALUES
('admin','ADMIN'),
('user','USER'),
('user2', 'USER');
```

Dans la configuration de la sécurité 'SecurityConfig'.

```
@Autowired
private DataSource dataSource;

protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
    PasswordEncoder passwordEncoder = getPasswordEncoder();
    auth.jdbcAuthentication() // il doit établir une connexion avec la
    db
        .dataSource(dataSource) // la même base de donnée de l'app
    (injectée)
        .usersByUsernameQuery("select " +
            " username as principal" + // pour que Spring
    savoir que c'est le username
            ", password as credentials" +
            ", isActive " +
            "from users " +
            "where username=?" ) // requête à exécutée pour
    récupérer les users
        .authoritiesByUsernameQuery("select " +
            "username as principal" +
            ", roleName as role" +
            "from user_role" +
            "where username = ? ")
        .rolePrefix("ROLE_") // 'USER' -> 'ROLE_USER' dans la
    session
        .passwordEncoder(passwordEncoder); // algo pour decrypter
    les passwords
}
```

Noter le contenu de la classe «SecurityConfig»

```
@Override // spécifier les droits d'accès
protected void configure(HttpSecurity http) throws Exception {

    // droits d'accès
    http.authorizeRequests()
        .antMatchers("/resources/**", "/home", "/").permitAll()
        .antMatchers("/editPatient/**"
            , "/deletePatient/**", "/saveEditedPatient/**"
            , "/saveNewPatient/**", "/addNewPatient/**").hasRole("ADMIN")
        .antMatchers("/patients/**").hasRole("USER")
        .anyRequest().authenticated(); // PROBLEM SOLVED :
    // it must be a children under the same authorizeRequests()
    // order matters
    // allowing static resources by configure(WebSecurity web)
    //.formLogin();
    //http.authorizeRequests().antMatchers("/", "/home").permitAll(); // for
    all
}
```

```

    // toutes les requets nécessite une authentification
    //http.authorizeHttpRequests().anyRequest().authenticated(); // PROBLEM :
prevent anonymous access home

    // hey Spring, je veux utiliser un 'form' d'authetification
    http.formLogin(); // default login form
    //http.formLogin().loginPage("/login"); // my own login form

    // configure errors (/error403 is a vue)
    http.exceptionHandling().accessDeniedPage("/error403");
}

```

```

@Override // Spécifier la stratégie avec laquelle Spring Sec va
// chercher les utilisateurs autorisés
protected void configure(AuthenticationManagerBuilder auth) throws Exception
{
    PasswordEncoder passwordEncoder = getPasswordEncoder();
    // inMemory authentication = les utilisateurs à utiliser l'application
    // seront stockés dans la mémoire (utiles pour les testes)
    auth.inMemoryAuthentication()
        .withUser("user")
        .password(passwordEncoder.encode("user"))
        .roles("USER")
        .and()
        .withUser("admin")
        .password(passwordEncoder.encode("admin"))
        .roles("USER", "ADMIN");
    //auth.inMemoryAuthentication() ... on peut reutiliser ...
    /*auth.inMemoryAuthentication()
        .withUser("user1")
        .password("{noop}1234") // // noop = no encryption needed
        .roles("USER");*/
}

```

```

@Override
public void configure(WebSecurity web) throws Exception {
    web.ignoring().antMatchers("/resources/**", "/static/**", "/css/**",
"/js/**", "/images/**", "/img/**");
}

@Bean
// executé au démarrage, et place
// l'objet retourné dans le context
// comme Spring Bean (il peut etre injecté n'importe où)
PasswordEncoder getPasswordEncoder() {
    // retourner le type d'encodage
    // get spring version
    //System.out.println(SpringVersion.getVersion()); // 5.3.16
}

```



```
        return new BCryptPasswordEncoder();
    }
```

Gestion des utilisateurs en utilisant la stratégie **USER DETAILS SERVICE**.

C'est la troisième manière de gestion des utilisateurs, dans cet manière la couche service de notre application qui va s'occuper de la gestion de l'authentification.

Pour faire cela on va avoir besoin d'un objet qui implémente l'interface **UserDetailsService**.

```
@Override // Spécifier la stratégie avec laquelle Spring Sec va
// chercher les utilisateurs autorisés
protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
    PasswordEncoder passwordEncoder = getPasswordEncoder();
    auth.userDetailsService(new UserDetailsService() {
        @Override
        public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
            return null;
        }
    });
}
```

Pour un code bien structurer on va créer une classe qui implémente l'interface **UserDetailsService**.

Mais d'abord on va créer la class AppUser et Role et gérer la relation entre les deux (Many to Many) dans le package Security.

```
public class AppUser {
    @Id
    private String userId;
    @Column(unique = false)
    private String username;
    private String password;
    private boolean active;
    @ManyToMany(fetch = FetchType.EAGER)
    // un utilisateur peut avoir plusieurs roles
    // EAGER car je souhaite avoir la listes
    // des role de l'utilisateur automatiquement lorsque
    // je charge l'user depuis la base de donée
    // avec LAZY, la liste ne va pas etre charger que
    // quand j'appelle user.getAppRoleList ..
    private List<AppRole> appRoleList = new ArrayList<>();
}
```

```
public class AppRole {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long roleId;
    @Column(unique = true)
    // Je dois pas avoir deux role
    // avec le meme nom
    private String roleName;
    private String description;
}
```

Leurs interfaces repository.

```
@Repository
public interface AppRoleRepository extends JpaRepository<AppRole, Long> {
    AppRole findAppRoleByRoleName(String roleName);
}
```

```
@Repository
public interface AppUserRepository extends JpaRepository<AppUser, String> {
    AppUser findAppUserByUsername(String username);
}
```

Et l'interface Service et son Implémentation.

```
@Service
public interface SecurityService {
    AppUser saveNewUser(String username, String password, String
passwordConfirmation);
    AppRole saveNewRole(String roleName, String description);
    void addRoleToUser(String username, String roleName);
    void removeRoleFromUser(String username, String roleName);

    AppUser loadUserByUsername(String username);

    void removeRoleOfUser();
}
```

L'implémentation de l'interface SecurityService

```
@Service
@AllArgsConstructor
// pour l'injection par constructeur
@Slf4j // logging purposes
@Transactional
```

```

// spring will manage the transactions
// all the methods are transactional !!
// more details:
// https://www.marcobehler.com/guides/spring-transaction-management-transactional-in-depth
public class SecurityServiceImpl implements SecurityService {
    private AppUserRepository appUserRepository;
    private AppRoleRepository appRoleRepository;
    private PasswordEncoder passwordEncoder;

    @Override
    public AppUser saveNewUser(String username, String password, String passwordConfirmation) {
        // verifying if the username is already there
        Optional<AppUser> appUserByName =
appUserRepository.findAppUserByUsername(username);
        if (appUserByName.isPresent()) {
            throw new RuntimeException("The username "+username+" is already taken!");
        }
        // verifying username and password
        if (username.length() < 3) throw new RuntimeException("Username should be longer than 3 chars");
        if (!password.equals(passwordConfirmation)) throw new RuntimeException("Password does not match !!");
        String hashedPass = passwordEncoder.encode(password);
        AppUser appUser = new AppUser();
        appUser.setUsername(username);
        appUser.setPassword(password);
        appUser.setActive(true);
        appUser.setUserId(UUID.randomUUID().toString());
        AppUser saveAppUser = appUserRepository.save(appUser);
        return saveAppUser;
    }

    @Override
    public AppRole saveNewRole(String roleName, String description) {
        AppRole appRole = new AppRole();
        // verifying if the role already exists
        Optional<AppRole> appRoleByName =
appRoleRepository.findAppRoleByRoleName(roleName);
        if (appRoleByName.isPresent()) {
            throw new RuntimeException("The role " +roleName+ " already exist!");
        }
        appRole.setRoleName(roleName);
        appRole.setRoleDescription(description);
        return appRoleRepository.save(appRole);
    }

    @Override
    public void addRoleToUser(String username, String roleName) {
        Optional<AppUser> appUser =
appUserRepository.findAppUserByUsername(username);
        Optional<AppRole> appRole =
appRoleRepository.findAppRoleByRoleName(roleName);
        if (appUser.isEmpty()) {

```

```

        throw new RuntimeException("User "+username+" not found!");
    }
    if(appRole.isEmpty()){
        throw new RuntimeException("Role "+roleName+" not found!");
    }

    appUser.get().getAppRoleList().add(appRole.get());
}

@Override
public void removeRoleFromUser(String username, String roleName) {
    Optional<AppUser> appUser =
appUserRepository.findAppUserByUsername(username);
    Optional<AppRole> appRole =
appRoleRepository.findAppRoleByRoleName(roleName);
    if(appUser.isEmpty()){
        throw new RuntimeException("User "+username+" not found!");
    }
    if(appRole.isEmpty()){
        throw new RuntimeException("Role "+roleName+" not found!");
    }

    appUser.get().getAppRoleList().remove(appRole.get());
}

@Override
public void changeUserActiveStatus(String username, boolean newStatus) {
    Optional<AppUser> appUser =
appUserRepository.findAppUserByUsername(username);
    if(appUser.isEmpty()){
        throw new RuntimeException("User "+username+" not found!");
    }
    appUser.get().setActive(newStatus);
}

@Override
public AppUser loadUserByUsername(String username) {
    return
appUserRepository.findAppUserByUsername(username).orElseGet(null);
}
}

```

Dans cette classe les méthodes vont être toutes transactionnelles pour plus d'info :

<https://www.marcobehler.com/guides/spring-transaction-management-transactional-in-depth>

Pour tester ces fonctionnalités:

```

@Bean
CommandLineRunner run(SecurityService securityService){
    // adding some users and roles
    return args-> {

```

```

securityService.saveNewUser("user", "user", "user");
//securityService.saveNewUser("user", "user", "user"); // exception
//securityService.saveNewUser("admin", "admin", "user"); // exception
securityService.saveNewUser("admin", "admin", "admin");

securityService.saveNewRole("USER", "user disc");
//securityService.saveNewRole("USER", "user disc"); //exception
securityService.saveNewRole("ADMIN", "admin disc");

securityService.addRoleToUser("user", "USER");
securityService.addRoleToUser("admin", "admin");
securityService.addRoleToUser("admin", "USER");

};
}

```

Résultat:

app_user_user_id	app_role_list_role_id
6f6d4972-db03-41dc-9547-23c843581887	1
b1286fe7-091c-46a7-b505-9906f383d9a7	2
b1286fe7-091c-46a7-b505-9906f383d9a7	1

Maintenant pour gérer l'authentification, nous allons créer une classe où on va implémenter l'interface UserDetailsService.

On va avoir donc dans SecurityConfig

```

@Autowired
private UserDetailsServiceImpl userDetailsService;
. . .
@Override // Spécifier la stratégie avec laquelle Spring Sec va
// chercher les utilisateurs autorisés
protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
    PasswordEncoder passwordEncoder = getPasswordEncoder();
    // userDetailsService injected
    auth.userDetailsService(userDetailsService);
}

```

Don ce cas Spring automatiquement, lorsqu'un utilisateur entre son username et mot de passe, fait appel à la méthode (loadUserByUsername()) redéfinie dans notre implémentation de l'interface UserDetailsService).

```

@Service
@AllArgsConstructor
public class UserDetailsServiceImpl implements UserDetailsService {
    private SecurityService securityService;
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        // get user
        AppUser appUser =
securityService.loadUserByUsername(username);
        // il faut retourner un objet UserDetails
        // Collection pour les role
        Collection<GrantedAuthority> grantedAuthorities = new
ArrayList<>();
        /*
        appUser.getAppRoleList().forEach(appRole -> {
            // de AppRole as String vers simpleGrantedAuthority
(GrantedAuthority)
            SimpleGrantedAuthority simpleGrantedAuthority = new
SimpleGrantedAuthority(appRole.getRoleName());
            grantedAuthorities.add(simpleGrantedAuthority);
        });
        */
        // en utilisant l'API Stream
        grantedAuthorities = appUser
            .getAppRoleList()
            .stream()
            .map(appRole -> {
                return new
SimpleGrantedAuthority(appRole.getRoleName());
            }).collect(Collectors.toList());

        // créer un User de Spring Security
        //org.springframework.security.core.userdetails.User;
        // de AppUser vers User (UserDetails)
        User springSecurityUser = new User(appUser.getUsername(),
appUser.getPassword(), grantedAuthorities);
        return springSecurityUser;
    }
}

```

Dans notre implémentation, on crée un User de Spring par notre AppUser, dans ce cas le User de Spring Security a besoin du Username, Password et la collection des GrantedAuthorities qu'on a créé par la listes des rôle de notre AppUser.

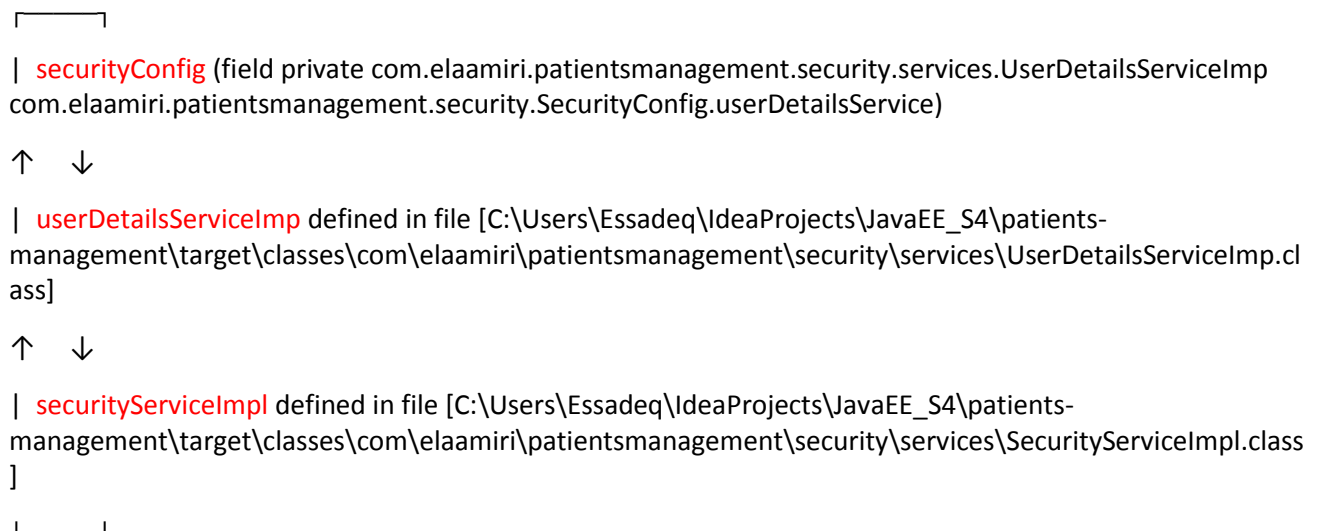
Donc normalement, Spring a l'habilité maintenant de gérer notre authentification, et lorsqu'un utilisateur entre le mot de passe et le Username, Spring va utiliser cette implémentation, pour créer UserDetails qui l'a besoin.

Mais il y a un problème :  la dépendance circulaire entre les Beans.

APPLICATION FAILED TO START

Description:

The dependencies of some of the beans in the application context form a cycle:



Action:

Relying upon circular references is discouraged and they are prohibited by default. Update your application to remove the dependency cycle between beans. As a last resort, it may be possible to break the cycle automatically by **setting spring.main.allow-circular-references to true**.

On peut éviter le problème en ajoutant la propriété :

```
# avoid circular dependence between Beans problem
spring.main.allow-circular-references = true
```

Dans le fichier **application.properties**, mais il faut trouver l'origine du problème en tous cas.

Pour plus d'info : <https://www.baeldung.com/circular-dependencies-in-spring>

Et dans la class ScurityConfig en utilise 'hasAuthority()' au lieux de 'hasRole()' dans la configuration des droits :

```
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/resources/**", "/home", "/").permitAll()
        .antMatchers("/editPatient/**"
            , "/deletePatient/**", "/saveEditedPatient/**"
            , "/saveNewPatient/**",
"/addNewPatient/**").hasAuthority("ADMIN")
        .antMatchers("/patients/**").hasAuthority("USER")
        .anyRequest().authenticated();
}
```

Si non on ne va pas avoir les droits.

Car Authority peut avoir plusieurs Roles dans la structure de Spring.

Il faut aussi corriger la contextualisation dans les vues en changeant hasRole() avec hasAuthority().

```
<li sec:authorize="hasAuthority('ADMIN')">
```

► Notez :

Reviendrons au problème des dépendances cycliques : l'utilisation de propriété

```
spring.main.allow-circular-references = true
```

pour éviter le problème, n'est pas d'un bon pratique, en fait, l'erreur c'est qu'on a déclaré le @Bean

PasswordEncoder dans la classe SecurityConfig.

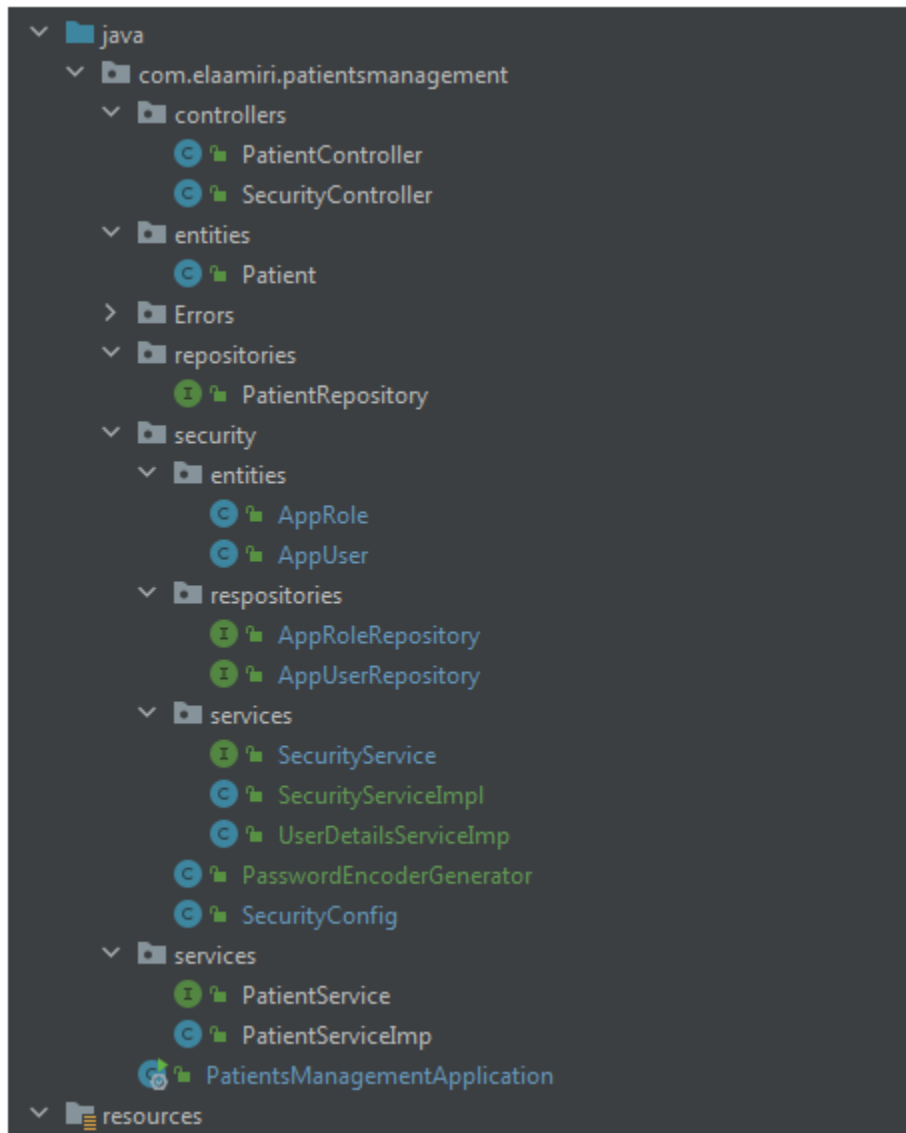
```
@Bean
// executé au démarrage, et place
// l'objet retourné dans le context
// comme Spring Bean (il peut etre injecté n'importe où)
PasswordEncoder getPasswordEncoder() {
    // retourner le type d'encodage
    // get spring version
    //System.out.println(SpringVersion.getVersion()); // 5.3.16
    return new BCryptPasswordEncoder();
}
```

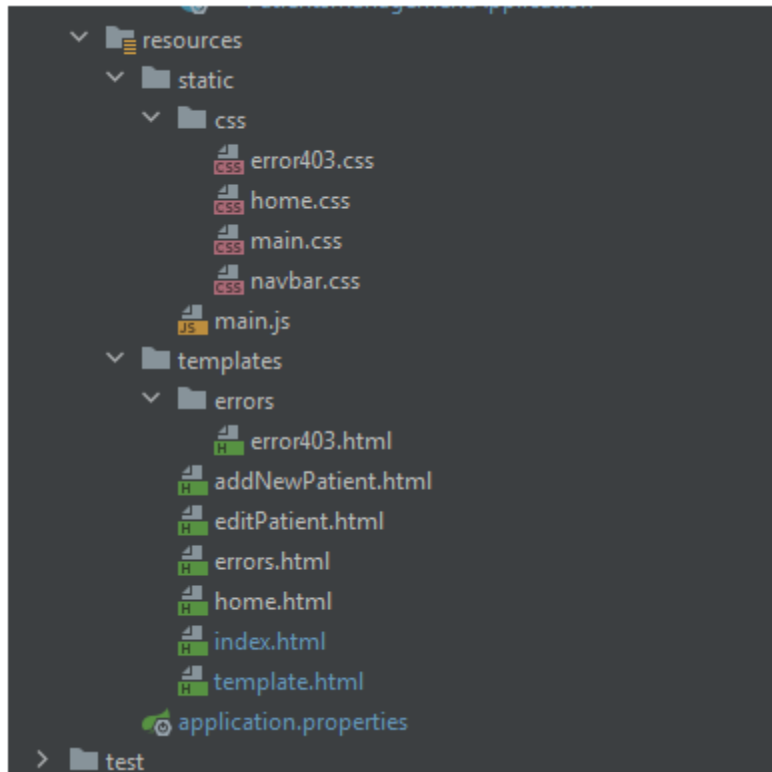

Donc comme solution on doit séparer le Bean vars une autre classe @Configuration, ou bien le mettre dans la classe main.

Je vais utiliser la première.

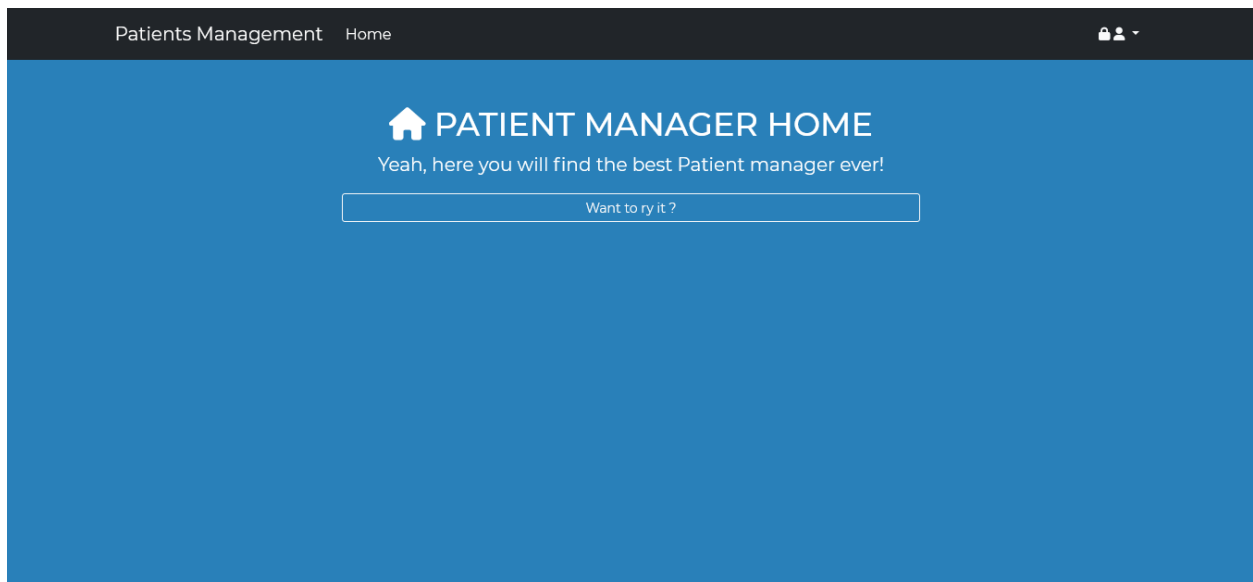
```
@Configuration
public class PasswordEncoderGenerator extends BCryptPasswordEncoder{
    // executé au démarrage, et place
    // l'objet retourné dans le context
    // comme Spring Bean (il peut etre injecté n'importe où)
    private PasswordEncoder getPasswordEncoder(){
        // retourner le type d'encodage
        // get spring version
        //System.out.println(SpringVersion.getVersion()); // 5.3.16
        return new BCryptPasswordEncoder();
    }
}
```

Captures d'écrans :





Anonymous



ADMIN

Please sign in

Sign in

Patients Management Home Patients ▾

admin ▾

PATIENTS LIST

Total number: 115 page: 0/23 GET DATA AS JSON (API)

Search a patient by first or last name



« < 0 1 2 3 4 > »

#ID	First Name	Last Name	Birth Day	Email	Action
0f50f4fd-9648-41af-ba9a-6c697bb0a13f	Kermy	Pickup	1978-07-31	kpickupla@bloglovin.com	
11115b3c-8643-4413-b89a-add6973752db	Davin	Biggs	1960-06-08	dbiggs2o@hexun.com	

Patients Management Home Patients ▾

admin ▾

PATIENTS LIST

Total number: 10 page: 0/2 GET DATA AS JSON (API)

Search a patient by first or last name



« < 0 1 > »

#ID	First Name	Last Name	Birth Day	Email	Action
12ddb9d9-62ed-4948-af9e-191e3d807428	ssaleim	ssaleim 55	2004-02-01	essadeq0701@gmail.com	
1752c54c-1bb0-47ad-b18d-2c86f41d0fde	ssaleim144	z144	2022-03-30	z1@w.com	

Patients Management

Home

Patients

admin

+ Add new

Search

PATIENTS LIST

Total number: 10

page: 0/2

GET DATA AS JSON (API)

Search a patient by first or last name

sal

Q

<<

<

0

1

>

>>

#ID	First Name	Last Name	Birth Day	Email	Action
12ddb9d9-62ed-4948-af9e-191e3d807428	ssalem	ssaleim 55	2004-02-01	essadeq0701@gmail.com	<div><div></div><div></div></div>
1752c54c-1bb0-47ad-b18d-2c86f41d0fde	ssalem144	z144	2022-03-30	z1@w.com	<div><div></div><div></div></div>

Patients Management

Home

Patients

admin

Add a new patient

First name

ali

Last name

ramda

Email address

valid@email.com

We'll never share your email with anyone else.

Birth date

01/02/2099

must be a past date

Cancel

Save

Patients Management

Home

Patients

admin

Edit patient :

ali ramda

ID: b5f6a8df-8645-4e88-945c-3685b7fe1df6

First name

ali

Last name

ramda

Email address

valid@email.com

We'll never share your email with anyone else.

Birth date

01/02/1999

SAVE

← TO THE LIST

ELAAMIRI-ENSET@2022

USER

PATIENTS LIST

Total number: 116 page: 0/24 GET DATA AS JSON (API)

Search a patient by first or last name

Type your key word

[«](#) [<](#) **0** [1](#) [2](#) [3](#) [4](#) [>](#) [»](#)

#ID	First Name	Last Name	Birth Day	Email	Action
0f50f4fd-9648-41af-ba9a-6c697bb0a13f	Kermy	Pickup	1978-07-31	kpickup1a@bloglovin.com	
11115b3c-8643-4413-b89a-add6973752db	Davin	Biggs	1960-06-08	dbiggs2o@hexun.com	
12ddb9d9-62ed-4948-af9e-191e3d807428	ssalem	ssaleim 55	2004-02-01	essadeq0701@gmail.com	
159324a9-9fcd-4618-8db3-a1cd1b91689d	Annie	Frier	1994-08-31	afrier2g@pen.io	
15b95dcd-d0b1-40bc-888a-1b2f7d0295bd	Clint	Rewbottom	1987-12-12	crewbottoml@stanford.edu	