



Année : 2021/2022

Compte rendu

Application web JEE, Gestion d'un hopital

Filière :

**Méthodes Informatiques Appliquées à la Gestion
des Entreprises**

Realisée par Loubna Talha	Encadré par M. Youssfi Mohammed
------------------------------	------------------------------------

Sommaire:

Sujet

Annotations et mots clés

Dependances

Structure de projet

Les Entités

Déployer le data source

Couche DAO avec Spring data

Repositories

Couche Web

Controleurs

Vues

Authentification SPRING SECURITY

Sujet

L'objectif principal consiste à concevoir et réaliser une application web dynamique avec le Framework Spring qui permet la gestion d'un hôpital. Les données sont stockées dans une base de données MySQL

L'application se compose de trois couches :

La couche DAO qui est basée sur Spring Data, JPA, Hibernate et JDBC.

La couche Métier

La couche Web basée sur MVC coté Serveur en utilisant Thymeleaf.

La sécurité est basée sur Spring Security

Annotations et mots clés

Injection de dépendance	Permet d'implémenter le principe de l'inversion de contrôle	
	Inversion de contrôle	=> découpler les dépendances
JDBC (Java Data Base Connection)	API qui permet d'utiliser les bases de données relationnelles	
Mapping Objet Relationnel (ORM)	gère l'accès aux données	

Object-Relational Mapping	
Hibernate	Un ORM qui implémente la spécification JPA
JPA (Java Persistence API)	Un API qui repose essentiellement sur l'utilisation des annotations
Spring Data	Un module de Spring qui facilite l'utilisation de JPA
Spring Data JPA	Fait l'ORM basé sur JPA
Lombok	Permet de générer les getters et les setters
Spring Web	Spring mvc
H2 Database	SGBD à mémoire (les données sont perdus apres chaque redémarrage)
application.properties	fichier de configuration de l'application
Entities	Package qui contient les classes qui vont etre par la suite des tables dans la bd

@Data	annotation de Lombok => gener les getters et setters et constructeurs
@ Entity	=>annotation essentiel pour une classe pour devenir une entité JPA
@ID	Primary key =>annotation essentiel pour une classe pour devenir une entité JPA
repositories	Package qui contient des interfaces qui hérite de l'interface JpaRepository => permet d'utiliser JPA
@Query	Annotation qui dit a SpringData comment interprète la fonction =>Utilise HQL (Hibernate query language)
@Bean	Dit a Spring d'exécuter celle-ci au démarrage
BindingResult	informations sur les erreurs de validation
@Valid	Hibernate va utiliser les annotations de validations (comme @Min et @NotEmpty) avant d'exécuter les requêtes sql

@configuration	Dit a Spring que cette classe doit être instancier au premier lieu
----------------	--

Dependances

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
  <groupId>nz.net.ultraq.thymeleaf</groupId>
  <artifactId>thymeleaf-layout-dialect</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
</dependency>
<!-- https://mvnrepository.com/artifact/org.webjars/bootstrap -->
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>bootstrap</artifactId>
  <version>5.1.3</version>
</dependency>
```

```

<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>jquery</artifactId>
  <version>1.9.1</version>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- https://mvnrepository.com/artifact/org.thymeleaf.extras/thymeleaf-extras-springsecurity5 -->
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
<!--<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>-->

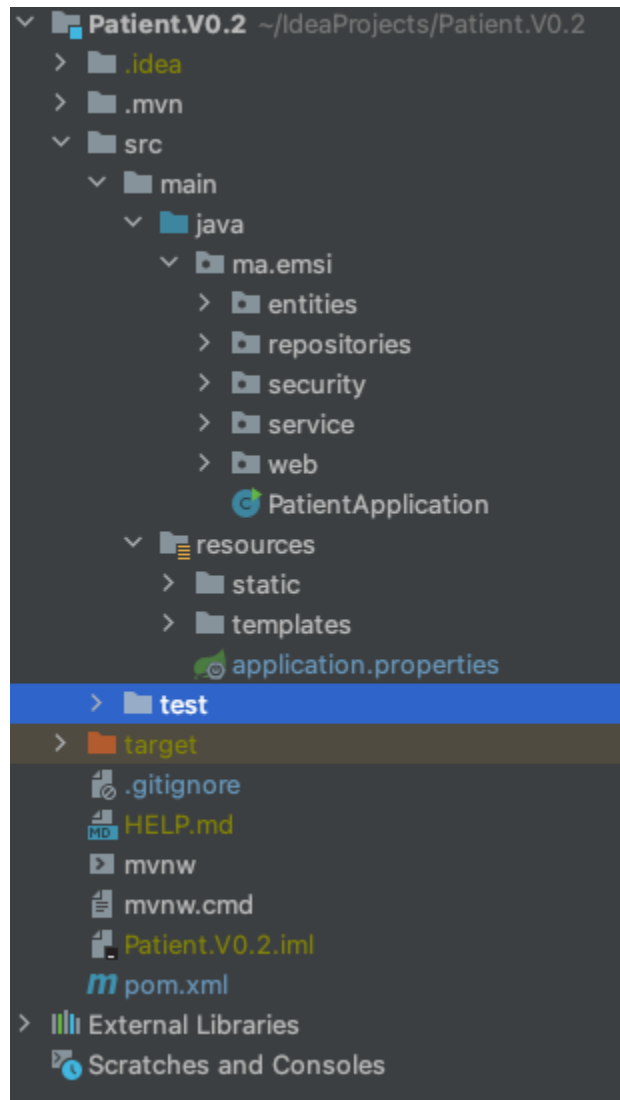
```

```

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>

```


Structure de projet



Les Entités

Dans l'application demandée on trouve tout 4 associations possible:

@OneToMany	@ManyToOne	@ManyToMany	@OneToOne
------------	------------	-------------	-----------

```
public class Patient {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
    @NotEmpty  
    @Size(min = 3,max = 10)  
    private String nom;  
    @Temporal(TemporalType.DATE)  
    @DateTimeFormat(pattern = "yyyy-MM-dd")  
    private Date dateNaissance;  
    private boolean malade;  
    private String adresse;  
    private String codePostal;  
    private String numeroTelephone;  
    private Titre titre;  
    @OneToMany(mappedBy = "patient" ,fetch=FetchType.LAZY)  
     private Collection<RendezVous> rendezVous;
```

```

@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Medecin {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
    private String specialite;
    @OneToMany(mappedBy = "medecin", fetch= FetchType.LAZY)
    @JsonProperty(access = JsonProperty.Access.READ_WRITE)
    private Collection<RendezVous> rendezVous;
}

```

```

@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class RendezVous {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date date;
    @Enumerated(EnumType.STRING)
    private StatusRDV statusRDV;
    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.READ_WRITE)
    private Patient patient;
    @ManyToOne
    private Medecin medecin;
    @OneToMany(mappedBy = "rendezVous")
    private Collection<Consultation> consultation;
    public enum StatusRDV{
        PENDING, CANCELED, DONE;
    }
}
}

```

```

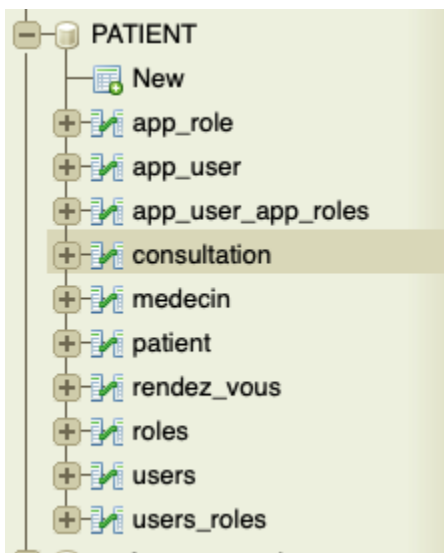
@Entity
@Data
@NoArgsConstructor @AllArgsConstructor
public class Consultation {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Temporal(TemporalType.DATE)
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date dateConsultation;
    private String rapport;
    @OneToOne
    @JsonProperty(access = JsonProperty.Access.READ_WRITE)
    private RendezVous rendezVous;
}

```

Déployer le data source

Le fichier de configuration 'application.properties'

```
#spring.datasource.url=jdbc:h2:mem:patientweb
#spring.h2.console.enabled=True
spring.datasource.url=jdbc:mysql://localhost:3302/PATIENT?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
spring.jpa.show-sql=true
server.port=8086
spring.thymeleaf.cache=false
spring.jpa.open-in-view=false
spring.main.allow-circular-references=true
```



Couche DAO avec Spring data

1. On a créer un package Repository
2. On a déclaré pour chaque entité une interface EntityRepository qui hérite de l'interface générique JpaRepository
3. On a déclaré les signatures des méthodes en respectant les règles de l'écriture pour récupérer les données sans avoir besoin de les implémenter grâce à Spring Boot

Repositories

Les interfaces Repositories

```
@Transactional
//patientDAO
//heriter d'une interface generique
public interface PatientRepository extends JpaRepository<Patient, Long> {

    Page<Patient> findByNomContains(String nom, Pageable pageable);

}
```

```
@Transactional
public interface MedecinRepository extends JpaRepository<Medecin, Long> {

    Page<Medecin> findByNomContains(String nom, Pageable pageable);

}
```

```
@Transactional
public interface RendezVousRepository extends JpaRepository<RendezVous, Long> {

    Page<RendezVous> findByDate(Date date, Pageable pageable);

}
```

```
@Transactional
public interface ConsultationRepository extends JpaRepository<Consultation, Long> {

}
```

```
public interface AppRoleRepository extends JpaRepository<AppRole, Long> {

    AppRole findByRoleName(String roleName);

}
```

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

Couche Web

<div><div>H</div><div>Patients ▾ Medecins ▾ Rendez-vous ▾ Consultations ▾</div><div>loubna ▾</div></div>				
LISTE DES PATIENTS				
<div><div>chercher un patient</div><div>Rechercher</div></div>				
Id	Nom	Malade	Supprimer	Modifier
1	jackson	true	Supprimer	Modifier
2	aloe	true	Supprimer	Modifier
3	mery	true	Supprimer	Modifier
4	saly	false	Supprimer	Modifier
5	rtyu	true	Supprimer	Modifier
<div><div>0</div><div>1</div></div>				

Nom

Email

Specialite

Save

Nom

Date de naissance

2022-04-30

Malade ☐

Save

Date

2022-04-30

- ☐ PENDING
☐ CANCELED
☐ DONE

select patient

select medecin

Save

Patients ▾ Medecins ▾ Rendez-vous ▾ Consultations ▾

Date de Consultation

2022-04-30

Rapport

select rendez-vous

Save

LISTE DES MEDECINS					
<input type="text" value="chercher un medecin"/> <input type="button" value="Rechercher"/>					
Id	Nom	specialite	email	Action	
1	norman1	dentist	norman@gmail.com	<input type="button" value="Supprimer"/>	<input type="button" value="Modifier"/>
2	kai	orthodontist	kai@gmail.com	<input type="button" value="Supprimer"/>	<input type="button" value="Modifier"/>
3	exo	dentiste	exo@gmail.com	<input type="button" value="Supprimer"/>	<input type="button" value="Modifier"/>
4	travis	gynécologue	travis@gmail.com	<input type="button" value="Supprimer"/>	<input type="button" value="Modifier"/>
5	khai	gynécologue	khai@gmail.com	<input type="button" value="Supprimer"/>	<input type="button" value="Modifier"/>
<input type="button" value="0"/> <input type="button" value="1"/>					

LISTE DES CONSULTATIONS					
Id	date de consultation	Rapport	Numero de rendez-vous	Action	
2	2022-04-28	RSTX	4	<input type="button" value="Supprimer"/>	<input type="button" value="Modifier"/>
3	2022-04-09	azerty	2	<input type="button" value="Supprimer"/>	<input type="button" value="Modifier"/>
5	2022-04-22	rtyuiop	4	<input type="button" value="Supprimer"/>	<input type="button" value="Modifier"/>
6	2022-04-06	ghjk	4	<input type="button" value="Supprimer"/>	<input type="button" value="Modifier"/>
<input type="button" value="0"/>					

LISTE DES RENDEZ VOUS					
Id	Date	Status	Patient	Medecin	Action
1	2022-04-06	CANCELED	aloe	kai	<input type="button" value="Annuler"/> <input type="button" value="Modifier"/>
2	2022-04-29	PENDING	aloe	norman1	<input type="button" value="Annuler"/> <input type="button" value="Modifier"/>
3	2022-04-05	CANCELED	aloe	kai	<input type="button" value="Annuler"/> <input type="button" value="Modifier"/>
4	2022-04-19	DONE	jackson	kai	<input type="button" value="Annuler"/> <input type="button" value="Modifier"/>
5	2022-04-12	DONE	jackson	norman1	<input type="button" value="Annuler"/> <input type="button" value="Modifier"/>
<input type="button" value="0"/> <input type="button" value="1"/>					

Controleurs

Une classe qui gère les requêtes http

La route '/user/index est liée à la méthode patients() elle va être appelée lorsqu'une requête de type GET est envoyée

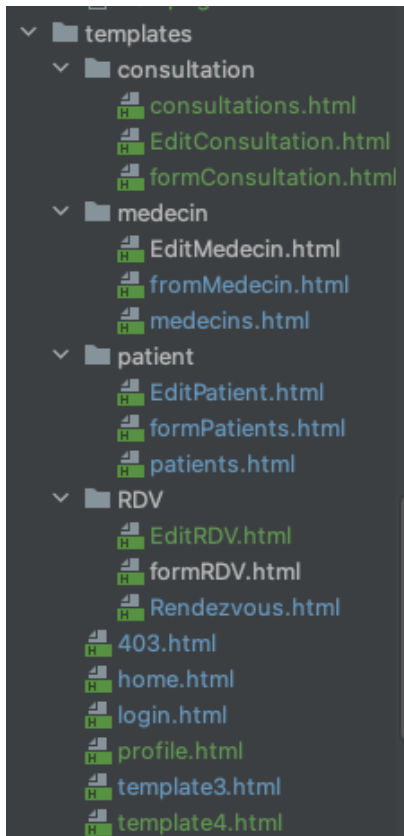
```

//classe qui va gérer les requetes http
@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping(path= "/user/index") //route
    // fct patients() appelée => si une requête de type GET est envoyée sur GetMapping
    public String patients(Model model,
        @RequestParam(name= "page", defaultValue = "0") int page, // parametre d'url : requete
        @RequestParam(name= "size", defaultValue = "5") int size,
        @RequestParam(name="keyword", defaultValue = "") String keyword){
        Page<Patient> pagePatients = patientRepository.findByNomContains(keyword, PageRequest.of(page,size));//
        //stocker la liste dans le model
        //model creer des variables, permet de recuperer les données aupres de la vue
        // je veux les patients de la page 0 et size 5
        model.addAttribute( attributeName: "listpatients", pagePatients.getContent()); // getcontent donne la liste
        model.addAttribute( attributeName: "pages", new int[pagePatients.getTotalPages()]);
        model.addAttribute( attributeName: "currentPage", page);
        model.addAttribute( attributeName: "keyword",keyword);
        return "patient/patients";// nom de la vue
    }
}

```

Vues

On a travaillé avec le moteur de template thymeleaf



On ajout le dialect thyemeleaf

```
<!DOCTYPE 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">
<head>
```

Exemple d'une vue

```
<body style="background-image:url('https://previews.123rf.com/images/winnievinzence/winnievinzence1704/winnievinzence17040689.jpg');">
<div layout:fragment="content1">
    <div class="col-md-6 offset-3">
        <form method="post" th:action="@{/admin/save(page=${page},keyword=${keyword})}">
            <div>
                <label for="nom">Nom</label>
                <input id="nom" class="form-control" type="text" name="nom" th:value="${patient.nom}">
                <span class="text-danger" th:errors="${patient.nom}"></span>
            </div>
            <div>
                <label>Date de naissance</label>
                <input class="form-control" type="date" name="dateNaissance" th:value="${patient.dateNaissance}">
                <span class="text-danger" th:errors="${patient.dateNaissance}"></span>
            </div>
            <div>
                <label>Malade</label>
                <input type="checkbox" name="malade" th:checked="${patient.malade}">
                <span class="text-danger" th:errors="${patient.malade}"></span>
            </div>

            <button type="submit" class="btn btn-primary">Save</button>

        </form>
    </div>

```

La methode save() reçoit les paramètres suivant: un objet de type Patient, quel page, sa taille et le keyword de la recherche

Authentification SPRING SECURITY

Interface SecurityService qui contient la déclaration des signatures des méthodes qui concerne les utilisateurs et des rôles:

```
//Interface => declarer les signatures des méthodes pour les traitements des utilisateurs et ses roles
public interface SecurityService {
    AppUser saveNewUser(String username, String password, String verifyPassword); //permet d'ajouter un nv user
    //pour le creer on aura besoin des 3 param ^
    AppRole saveNewRole(String roleName, String description); //creer une role
    void addRoleToUser(String username, String roleName); //associer le role a un user
    AppUser loadUserByUsername(String username); //chercher un utilisateur
    void removeRoleFromUser(String username, String roleName); // supprimer un role d'une user
}
```

La classe suivant est une classe qui va servir à la configuration de Spring Security, elle hérite de la classe WebSecurityConfigurerAdapter

```
@Configuration
// pour dire que c'est une classe de configuration
//chaque classe qui utilise l'annotation config va etre instancier du 1er lieu
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired
    private DataSource dataSource;
    @Autowired
    private UserDetailsService userDetailsService;
    @Override
    //configure=>la methode configure avec AuthenticationManagerBuilder va servir pour preciser quel strategie
    //que vous voulez utiliser pour que spring security va chercher l'utilisateur
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
```

Contient La méthode suivante, configurée avec AuthenticationManagerBuilder qui va servir pour préciser quel stratégie que vous voulez utiliser afin que spring security l'utilise pour chercher l'utilisateur

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    PasswordEncoder passwordEncoder= PasswordEncoder(); //permet de creer un objet BCryptPasswordEncoder
    /*...*/
    /*...*/
    quand l'utilisateur va entrer son username et mdp spring sec va faire appel a l'obj userDetailsService qu
    auth.userDetailsService(userDetailsService);
}
}
```

La méthode suivante prend comme paramètre HttpSecurity et va servir a spécifier les droits d'accès

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

    // pour specifier les droits d'accès
    // http.formLogin(); // demander a spring security => utiliser un formulaire d'authentification
    http.formLogin().loginPage("/login");
    // pour utiliser votre formulaire on ajout:
    //http.formLogin().loginPage("/login");
    //qlq soit les ressources utilisé dans l'app nécessite une auth
    //http.authorizeRequests().antMatchers("/delete/**", "/edit/**", "/save/**", "/formpatients/**").hasRe
    // ^ tt nécessite un role admin
    //http.authorizeRequests().antMatchers("/index/**").hasRole("user");
    http.authorizeRequests().antMatchers( ...antPatterns: "/").permitAll();
    http.authorizeRequests().antMatchers( ...antPatterns: "/admin/**").hasAnyAuthority( ...authorities: "ADMIN");
    http.authorizeRequests().antMatchers( ...antPatterns: "/user/**").hasAnyAuthority( ...authorities: "USER");
    http.authorizeRequests().antMatchers( ...antPatterns: "/resources/**" ,"/webjars/**", "/login").permitAll()
    //autoriser les ressources static
    http.authorizeRequests().anyRequest().authenticated();
    http.exceptionHandling().accessDeniedPage("/403");
}
}
```

La méthode suivante retourne un PasswordEncoder qui va servir a encoder les mots de passes

```
@Bean // au démarrage creer moi un pwdencoder
//il va utiliser bcrypt
PasswordEncoder PasswordEncoder() { return new BCryptPasswordEncoder(); }
```

Le résultat est un mot de passe difficile déchiffrer

		user_id	active	password	username
<input type="checkbox"/>	 Edit  Copy  Delete	8d8fc53c-574c-464c-883c-5356550db01a	1	\$2a\$10\$ZdQv0rgvHJ9cyf3Ql1nJM.b0jy2G4Ym5c77fXgB3HmL...	testuser1
<input type="checkbox"/>	 Edit  Copy  Delete	9c198e69-9c93-408e-b398-6d30b7b05c2c	1	\$2a\$10\$ifAugVRRX9sCE2YLOU1Bpe4B2enaOGQzEg0ekWRbQkY...	loubna
<input type="checkbox"/>	 Edit  Copy  Delete	db83379b-91b7-4aad-8e90-27efc10437cb	1	\$2a\$10\$kir6JgKgVerPwzP9FXhV1OjAEJt2gXHGajnFS4j/3TV...	testuser2