





DEPARTEMENT MATHEMATIQUES ET INFORMATIQUE

Compte rendu 5 JEE

Filière:

« Génie du Logiciel et des Systèmes Informatiques Distribués » **GLSID**

Le 25/03/2022

Préparé par : Ahlam CHAKRANE

Année Universitaire: 2020-2021

Introduction

Ce TP a pour objectif de mettre en évidence l'utilisation des JSP et l'architecture Spring MVC coté client.

Dans ce TP, nous allons faire la gestion des patients, et nous allons traiter diverses actions notamment : l'ajout, la modification, la suppression, la recherche et la pagination.

Pour mettre en place notre application, nous devrons créer tout d'abord la classe Patient :

```
Patient.java X PatientRepository.java
                                                               package ma.enset.Seance6.entities;
  3⊖ import java.util.Date;
 4
 5 import javax.persistence.Entity;
 6 import javax.persistence.GeneratedValue;
 7 import javax.persistence.GenerationType;
 8 import javax.persistence.Id;
 9 import javax.persistence.Temporal;
 10 import javax.persistence.TemporalType;
 11
12 import lombok.AllArgsConstructor;
13 import lombok.Data;
14 import lombok.NoArgsConstructor;
15 @Entity
▲16 @Data @AllArgsConstructor @NoArgsConstructor
 17 public class Patient {
        @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
18⊖
19
        private Long id;
        private String nom;
 20
 21⊖ @Temporal(TemporalType.DATE)
22
        private Date dateNaissance;
        private boolean malade;
 23
        private int score;
 24
25 }
 26
```

Les annotations @NoArgsConstructor et @AllArgsConstructor permet de générer automatiquement des constructeurs sans et avec paramètres.

L'annotation @Id permet de déclarer de l'attribue est une clé primaire de la table de la base de données.

Ensuite nous avons créé un Repository qui hérite de JPARepository pour générer automatiquement les méthodes classiques qui permettent de gérer les entités JPA.

```
Patient.java X PatientRepository.java X Seance6Application.java application.properties

package ma.enset.Seance6.repositories;

import org.springframework.data.jpa.repository.JpaRepository;

import ma.enset.Seance6.entities.Patient;

public interface PatientRepository extends JpaRepository<Patient, Long>{
```

Dans la fonction main, nous avons ajouté des patients à stocker dans la base de données.

```
Patient.java
                PatientRepository.java
                                        application.properties
 1 package ma.enset.Seance6;
 3⊕ import java.util.Date;
12
13 @SpringBootApplication
14 public class Seance6Application {
15
        public static void main(String[] args) {
160
17
            SpringApplication.run(Seance6Application.class, args);
18
19⊝
        @Bean
        CommandLineRunner commandeLineRunner(PatientRepository patientRepository) {
20
21
            return args -> {
22
                patientRepository.save(new Patient(null, "ahlam", new Date(), true, 30));
23
                patientRepository.save(new Patient(null, "aicha", new Date(), false, 10));
24
                patientRepository.save(new Patient(null, "taoufik", new Date(), false, 20));
25
                patientRepository.save(new Patient(null, "anas", new Date(), true, 100));
26
                patientRepository.findAll().forEach(p->{
27
                    System.out.println(p.getNom());
28
                });
29
            };
30
31
32
```

Dans le fichier application.properties, nous avons déclarer la base de données à utiliser, et le port

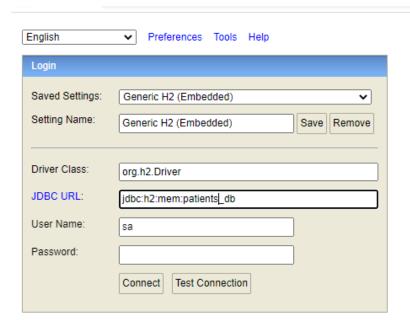
```
Patient.java PatientRepository.java Pseance6Application.java application.properties ×

1 spring.datasource.url=jdbc:h2:mem:patients_db
2 spring.h2.console.enabled=true
3 server.port=8082
```

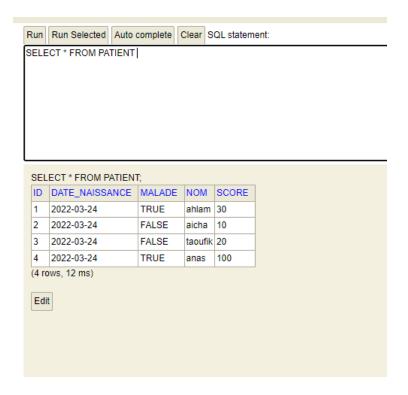
Apres avoir tourner l'application, les données ont été stocker dans notre base de données qui porte le nom « patinets_db » :

```
ahlam
aicha
taoufik
anas
2022-03-24 09:04:27.824 INFO 4716 --- [nio-8082-exec-2022-03-24 09:04:27.824 INFO 4716 --- [nio-8082-exec-2022-03
```

Alors si on accède à la base de données H2:



On trouve les enregistrements suivants :



Pour basculer vers MySQL, il faut changer les données du fichier application.properties :

```
patients.html
Patient.java
               application.... X  PatientCont...
                                                                                      addPat
 1 #spring.datasource.url=jdbc:h2:mem:patients db
 2 #spring.h2.console.enabled=true
 3 #server.port=8082
 4 spring.datasource.url=jdbc:mysql://localhost:3306/patients db?createDatabaseIfNotExist=true
 5 spring.datasource.username=root
 6 spring.datasource.password=
 7 server.port=8082
 8 spring.jpa.hibernate.ddl-auto = update
 9 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDBDialect
10 spring.jpa.show = true
11 spring thymeleaf.cache= false
```

Pour faire la gestion des routes, nous avons créé une classe controleur pour faire le traitement nécéssaire lors de l'appel des routes :

La première route :

Lorsqu'on tape la route /index, on se dirige dans la page patients.html que dans laquelle on affiche 8 patients dans chaque page, puisqu'on utilise la pagination.

Pour se faire, on récupère la liste des patients, et on la stocke dans une variable appelée ListPatients, qu'on la mit dans la variable model de type Model.

On passe également le nombre de page pour faire l'affichage des patients selon ce nombre, la page courante, et le mot recherché s'il existe dans la barre de recherche pour ne le pas perdre si l'utilisateur passe d'une page à une autre.

La deuxième route :

```
@GetMapping("/delete")
public String delete(Long id, String keyword, int page) {
    patientRepository.deleteById(id);
    return "redirect:/index?page="+page+"&keyword="+keyword;
}
```

Lorsque l'utilisateur click sur le bouton de suppression, il sera dirigé vers cette fonction, que dans la quel on supprime le patient qui a l'id correspondant avec celui du patient sélectionné, et finalement on redirige l'utilisateur vers la page index, tout en gardant la page courante et le mot recherché s'il existe.

La troisième route :

```
@GetMapping("/add")
public String add() {
    return "addPatient";
}
```

Lorsque le client click sur le bouton d'ajout, il sera envoyé vers la vue addPatient.html

La quatrième route :

```
@PostMapping("/save")
public String save(Model model, Patient patient) {
    patientRepository.save(patient);
    return "redirect:/index";
}
```

Lorsque l'utilisateur click sur le bouton save pour enregistrer un nouvel utilisateur, il sera dirigé vers cette route pour sauvegarder le patient ajouté, puis le redirigé vers la vue index.html

La cinquième route :

```
@GetMapping("/update")
public String update(Model model, Long id) {
    Patient patient=patientRepository.findById(id).get();
    model.addAttribute("patient",patient);
    return "updatePatient";
}
```

Si l'utilisateur souhaite effectuer des modifications sur un patient, il va être diriger vers la route update. Un patient qui portera le même id que celui sélectionné sera sélectionner de la base de données, puis stocker dans un objet patient, qu'on stocke dans la variable model de type model. Tous cela pour récupérer ce patient dans la vue updatePatient.html, pour afficher les informations correspondantes à ce patient.

La sixième route :

```
@GetMapping("/")
public String home() {
    return "redirect:/index";
}
/**/
```

Si l'utilisateur ne précise pas de route il sera alors dirigé vers la vue index.html

Alors pour afficher une liste de patients voici la vue patients.html :

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>title</title>
   <link rel="stylesheet" href="/webjars/bootstrap/5.1.3/css/bootstrap.min.css">
   < link rel="stylesheet" href="/webjars/bootstrap-icons/1.8.1/font/bootstrap-icons.css">
</head>
<body>
<div class="container mt-4">
   <div class="card">
      <div class="card-header">
      <u>Liste des</u> patients
      <a class="btn btn-success" th:href="@{add}">
         <i class="icon-eye">+</i>
      </a>
      </div>
      <div class="card-body">
         <form method="get" th:action="@{index}">
            <label>Key word</label>
            <input type="text" name="keyword" th:value="${keyword}">
            <button type="submit" class="btn btn-primary">Search</button>
         </form>
         <thead>
         ID
            Nom
            Date
            Malade
            Score
            Action
         </thead>
       tnieacn= p:s{Listratients} >

            </a>
              <a class="btn btn-warning" th:href="@{update(id=${p.id}))}">
              <i class="bi bi-pen">update</i></a>
            th:text="${status.index}"
          th:href="@{index(page=${status.index}, keyword=${keyword})}">
          </a>
      </div>
    </div>
 </div>
 </body>
 </html>
```

```
, keyword=${keyword}, page=${currentPage})}">
```

Pour ajouter un patient voici la vue addPatient.html :

```
1<!DOCTYPE html>
2 <html lang="en" xmlns:th="http://www.thymeleaf.org">
3 <head>
     <meta charset="UTF-8">
     <title>New patient</title>
     k rel="stylesheet" href="/webjars/bootstrap/5.1.3/css/bootstrap.min.css">
7 </head>
B <body>

3 <center>

     <div class="col-md-6">
         <div class="panel panel-default">
              <div class="panel-heading">New Patient</div>
              <div class="panel-body">
                  <form th:action="@{save}" method="post">
                      <div class="form-group">
                          <label class="control-label">Nom:</label>
                          <input class="form-control" type="text" name="nom">
                      </div>
                      <div class="form-group">
                          <label class="control-label">Date Naissance:</label>
                          <input class="form-control" type="date" name="dateNaissance" value="12/12/2022" required>
                      </div>
                      <div class="form-check">
                        <input class="form-check-input" type="checkbox" name="malade" id="malade"</pre>
                        th:checked="${flag} ? 'checked'">
                        <label class="form-check-label" for="malade">
                          Malade
                        </label>
                      </div>
                      <div class="form-group">
                          <label class="control-label">Score:</label>
<input class="form-control" type="number" name="score" value="0">
                      <button type="submit" class="btn btn-success">Save</button>
                  </form>
              </div>
         </div>
    </div>
3</center>
</body>
           Writable
                                                   38 - 11 - 1311
```

Et pour modifier un patient voici la vue updatePatient.html

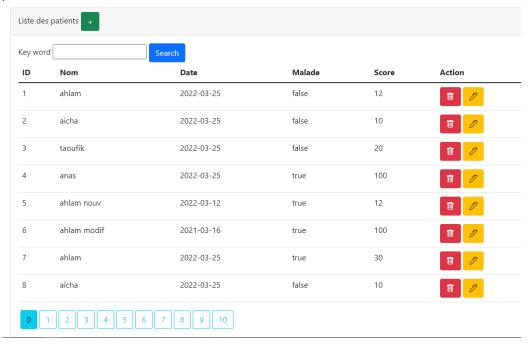
```
8 <body>
9 <center>
              <div class="col-md-6">
     <div class="panel panel-default">
                                    </div>
                                                           Malade
</label>
                                                            </div>
                                                           $\langle \text{3.50.0000} \text{
\text{div class="form-group"} \\
\text{\langle class="control-label">Score:\text{\label} \\
\text{\langle class="form-control" type="number" name="score" th:value="${patient.score}">
\text{\langle class="form-control" type="number" name="score" th:value="score" th:value="score" type="number" name="score" th:value="score" type="number" name="score" type="number" name="number" name="score" type="number" name="number" name=
                                                            <button type="submit" class="btn btn-success">Save</button>
                                                </form>
                                     </div>
                          </div>
2 </div>
3 </center>
4 </body>
```

Alors

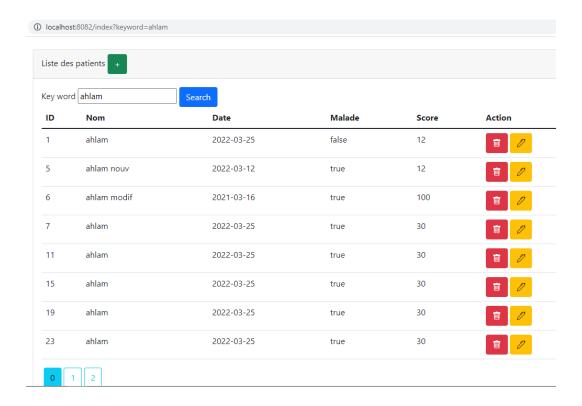
notre

application a pris la forme suivante :

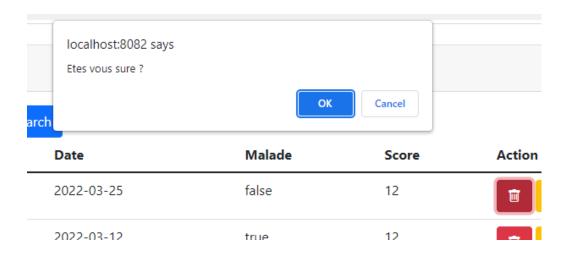
La liste des patients :



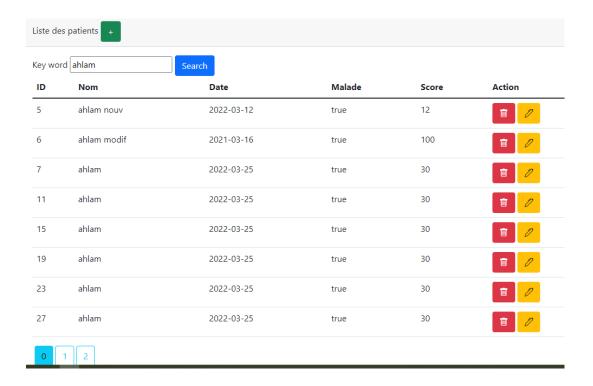
Rechercher un patient :



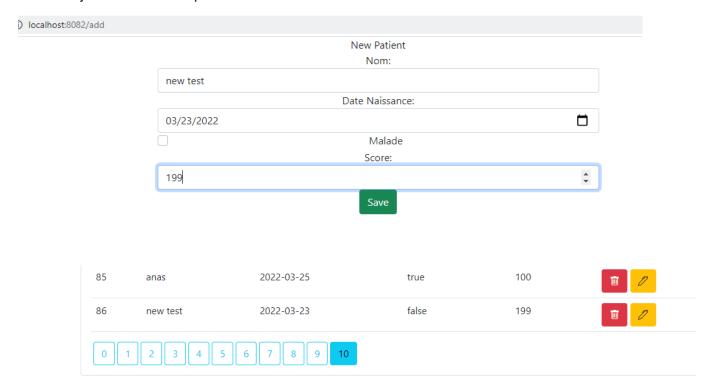
Supprimer le patient dont l'id est 1 :



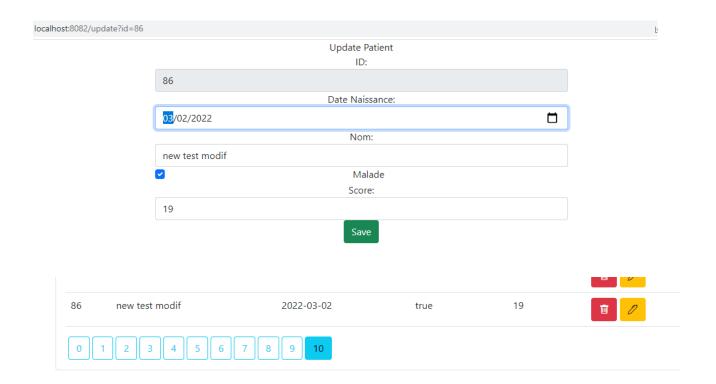
Liste des patients après suppression :



Ajouter un nouveau patient :



Modifier un patient :



Conclusion:

Grace à ce TP, nous avons pu comprendre l'architecture Spring MVC, et comment créer des JSP. Nous avons pu mieux développer nos compétences en programmation JEE, ainsi que nous avons pu améliorer notre connaissance de gérer des routes.