Spring MVC

Gestion des Patients

AZEHAF Issam
II-BDCC

Introduction

Ce TP consiste à faire une application de gestion des patients, l'application va nous permettre de ajouter,supprimer,modifier,rechercher … des patients.
Dans cette application on va utiliser Spring MVC avec le moteur de recherche Thymleaf

Demonstration

Commencons par créer notre classe de patients

```java
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Patient {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    @Temporal(TemporalType.DATE)
    private Date dateNaissance;
    private boolean malade;
    private int score;

}
```

Figure 1 : Classe Patient

Après on va créer notre couche Dao en créons une interface PatientRapository qui hérite de l'interface JPARepository

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

}
```

Figure 2 : PatientRepository

Et après on va tester on ajoutons des données dans la table patient

```java
@Bean
CommandLineRunner start(PatientRepository patientRepository){
    return args -> {
        patientRepository.save(new Patient( id: null, nom: "Azehaf",new Date(), malade: false, score: 18));
        patientRepository.save(new Patient( id: null, nom: "mbarako",new Date(), malade: true, score: 12));
        patientRepository.save(new Patient( id: null, nom: "elamri",new Date(), malade: true, score: 20));
        patientRepository.save(new Patient( id: null, nom: "syah",new Date(), malade: false, score: 14));

        patientRepository.findAll().forEach(p->{
            System.out.println(p.getNom());
        });
```

Figure 3 : Ajout des données

SELECT * FROM PATIENT;

ID	DATE_NAISSANCE	MALADE	NOM	SCORE
1	2022-04-01	FALSE	Azehaf	18
2	2022-04-01	TRUE	mbarako	12
3	2022-04-01	TRUE	elamri	20
4	2022-04-01	FALSE	syah	14

(4 rows, 7 ms)

Figure 4 : Table patient

Passant affichons ces données dans une page web pour faire cela il nous faut un controller.
Au premier lieu ce controller va avoir une seul methode qui retourne une page HTML en initialisons le model avec la listes despatient la route est /index

```java
@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;

    @GetMapping(path = "/index")
    public String patients(Model model){
        List<Patient> patients = patientRepository.findAll();
        model.addAttribute( attributeName: "listPatients",patients);
        return "patients";
    }
}
```

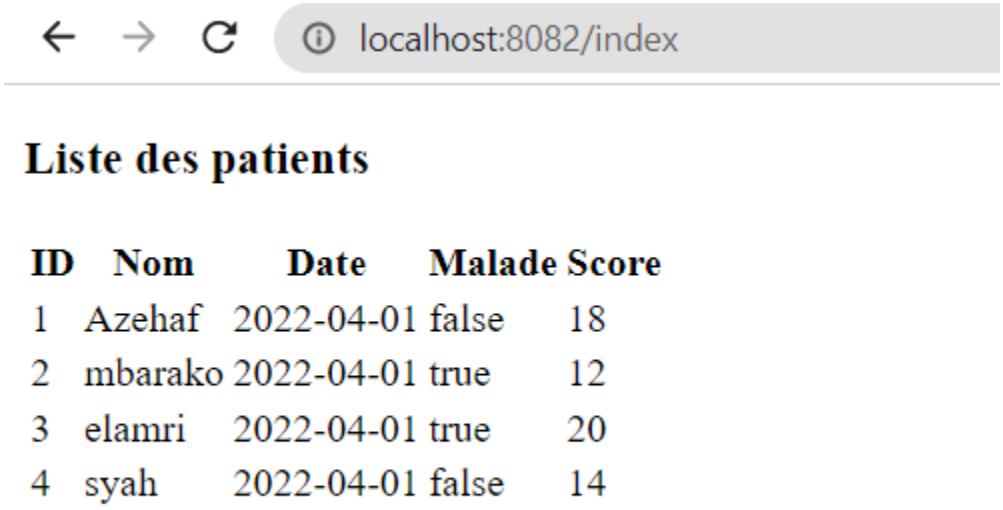
Figure 5 : Controller

Passant affichons ces donnée dans la page html puisque on utilise Thymeleaf on doit avoir ca

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

```
<tbody>
    <tr th:each="p:${listPatients}">
        <td th:text="${p.id}"></td>
        <td th:text="${p.nom}"></td>
        <td th:text="${p.dateNaissance}"></td>
        <td th:text="${p.malade}"></td>
        <td th:text="${p.score}"></td>
    </tr>
</tbody>
```

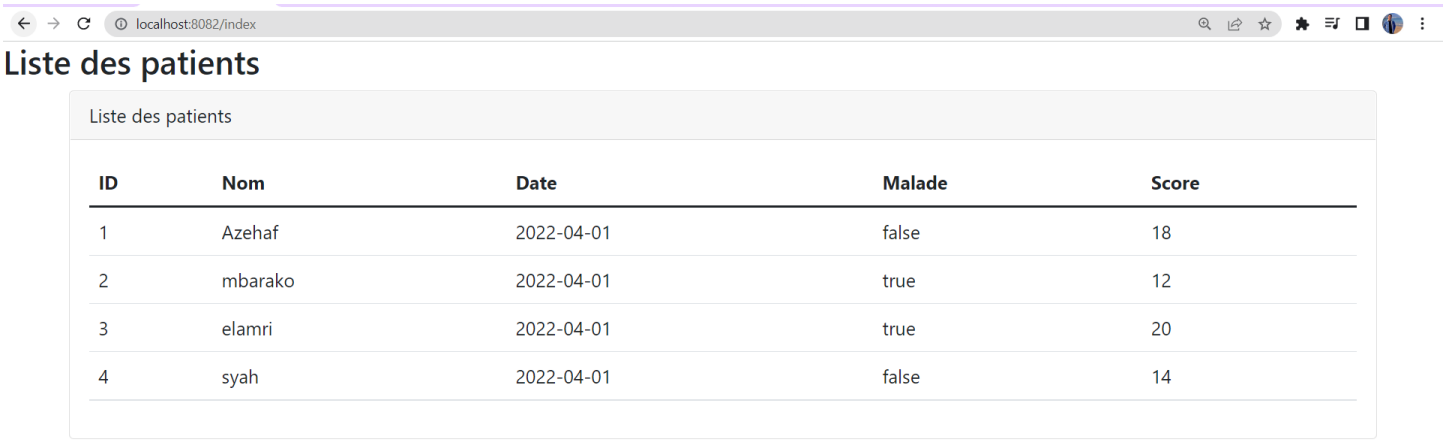
Et voici les données sont affiché dans la page web /index



On veut bien integrer du bootstrap dans notre page, pour cela in nous faut une dependance.

```xml
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>5.1.3</version>
</dependency>
```

```xml
<link rel="stylesheet" href="webjars/bootstrap/5.1.3/css/bootstrap.min.css">
```



Après je veux bien basculer vers une base de donnée mysql pour cela

```properties
#spring.datasource.url=jdbc:h2:mem:patients_db
spring.datasource.url=jdbc:mysql://localhost:3306/PAT_DB?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=8082
spring.h2.console.enabled=true
```

+ Options

	id	date_naissance	malade	nom	score
Éditer Copier Supprimer	1	2022-04-02	0	Azehaf	18
Éditer Copier Supprimer	2	2022-04-02	1	mbarako	12
Éditer Copier Supprimer	3	2022-04-02	1	elamri	20
Éditer Copier Supprimer	4	2022-04-02	0	syah	14
Éditer Copier Supprimer	5	2022-04-02	0	Azehaf	18
Éditer Copier Supprimer	6	2022-04-02	1	mbarako	12
Éditer Copier Supprimer	7	2022-04-02	1	elamri	20
Éditer Copier Supprimer	8	2022-04-02	0	syah	14
Éditer Copier Supprimer	9	2022-04-02	0	Azehaf	18
Éditer Copier Supprimer	10	2022-04-02	1	mbarako	12
Éditer Copier Supprimer	11	2022-04-02	1	elamri	20

Console de requêtes SQL

Après on va faire la pagination des données.

```
@GetMapping(path = "/index")
public String patients(Model model,
                       @RequestParam(name = "page", defaultValue = "0") int page,
                       @RequestParam(name = "size",defaultValue = "5") int size){
    Page<Patient> pagePatients = patientRepository.findAll(PageRequest.of(page,size));
    model.addAttribute( attributeName: "listPatients",pagePatients.getContent());
    model.addAttribute( attributeName: "pages",new int[pagePatients.getTotalPages()]);
    model.addAttribute( attributeName: "currentPage",page);
    return "patients";
}
```

```
<ul class="nav nav-pills">
    <li th:each="page,status:${pages}">
        <a th:class="${status.index==currentPage?'btn btn-primary ms-1':'btn btn-outline-primary ms-1'}"
           th:text="${status.index}"
           th:href="@{index(page=${status.index})}"></a>
    </li>
</ul>
```

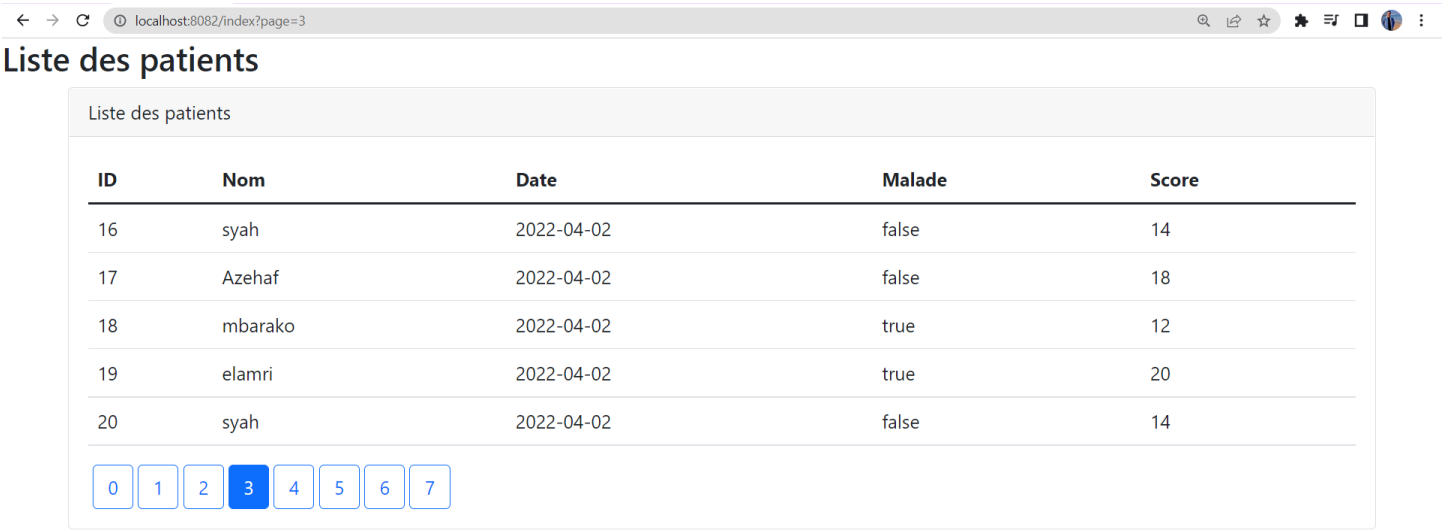


Figure 6 : Pagination

Ajoutons un input et des bouttons pour qu'on puisse faire des opérations sur le patient.