

Département d'Informatique

Filiere : IAAD

A ,U :2023-2024



جامعة مولاي إسماعيل
 UNIVERSITÉ MOULAY ISMAÏL



كلية العلوم
FACULTÉ DES SCIENCES

Université Moulay Ismail faculté des Sciences Meknès

Département d'Informatique

TP N 3 : Spring Security

Module : Systèmes distribués

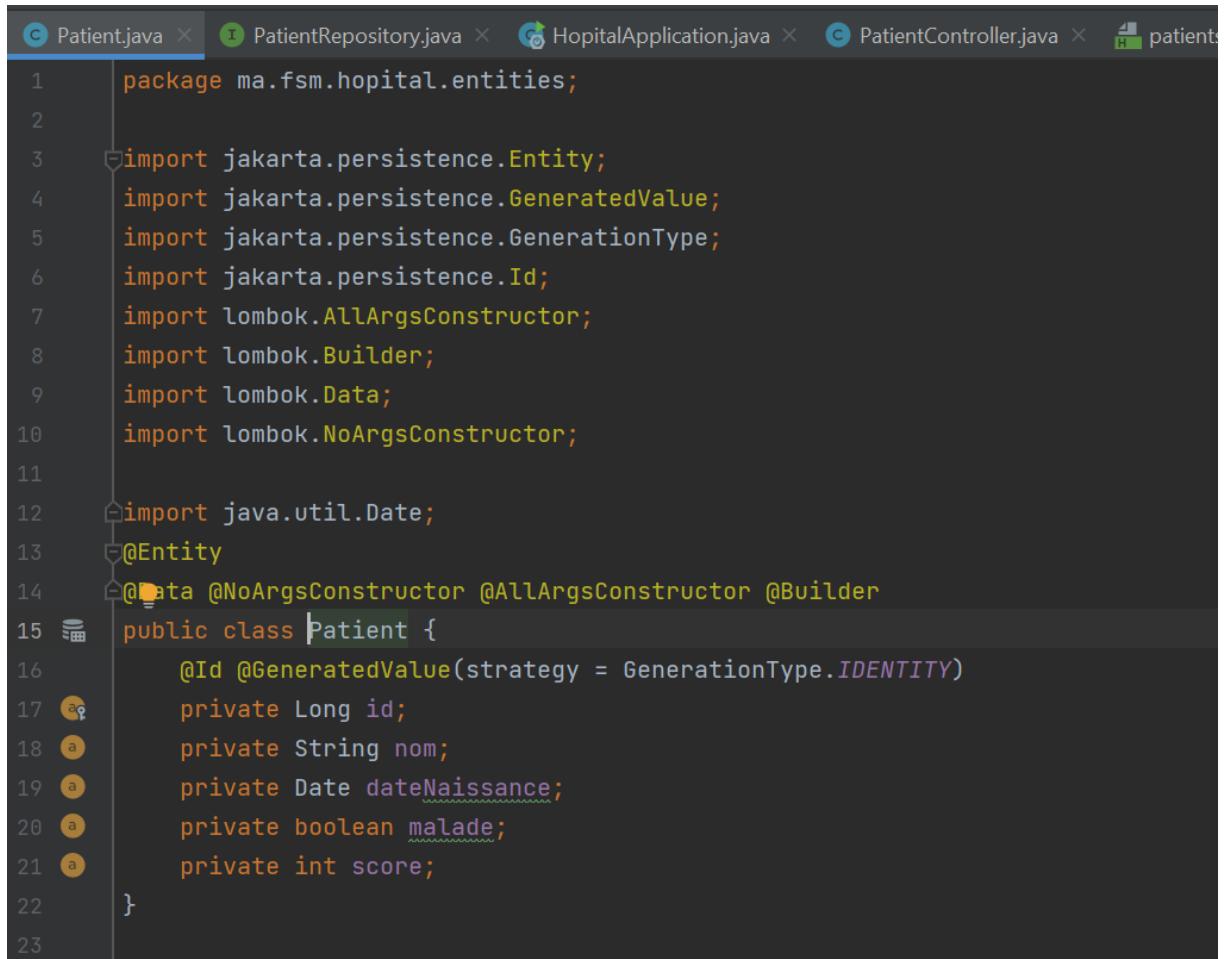
Réalisé par :

- *Illa Meryeme*

Partie 1 :

Créer une application web JEE basée sur les dépendances Spring MVC, Thymeleaf et Spring Data JPA qui permet de gérer les patients

On crée les packages usuelles entities, repositories et web avec leurs classes et interfaces et on utilise les notations @Data pour getters et setters, @NoArgsConstructor, @AllArgsConstructor, @Builder pour créer un patient et @Entity pour créer un entité jpa, @Id et @GeneratedValue pour l'identifiant auto incrément



```
1 package ma.fsm.hopital.entities;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7 import lombok.AllArgsConstructor;
8 import lombok.Builder;
9 import lombok.Data;
10 import lombok.NoArgsConstructor;
11
12 import java.util.Date;
13
14 @Entity
15 @Data @NoArgsConstructor@AllArgsConstructor@Builder
16 public class Patient {
17     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private Long id;
19     private String nom;
20     private Date dateNaissance;
21     private boolean malade;
22     private int score;
23 }
```

Et pour faire un traitement au démarrage le plus simple c'est d'implémenter l'interface CommandLineRunner, et redéfinir la méthode run.

Il y a 3 solutions pour créer un patient, on utilise constructeur sans paramètre, constructeur avec paramètre ou en utilisant Builder

```
private PatientRepository patientRepository;

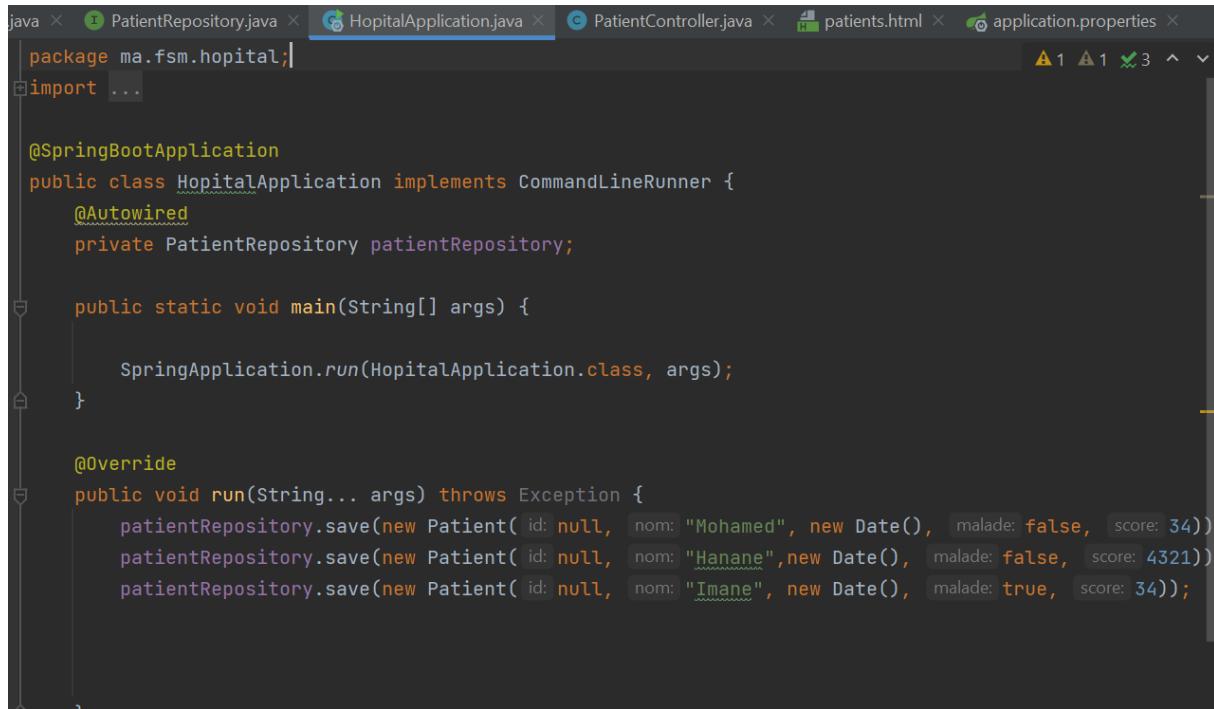
public static void main(String[] args) {
    SpringApplication.run(HopitalApplication.class, args);
}

@Override
public void run(String... args) throws Exception {
    // En utilisant constructeur sans paramètre
    Patient patient= new Patient();
    patient.setId(null);
    patient.setNom("Mohamed");
    patient.setDateNaissance(new Date());
    patient.setMalade(false);
    patient.setScore(23);
    // En utilisant constructeur avec paramètre
    Patient patient2= new Patient(id: null, nom: "Yassine",new Date(), malade: false, score: 123);
    //En utilisant Builder
    Patient patient3=Patient.builder()
        .nom("Imane").dateNaissance(new Date()).score(56).malade(true).build();
```

On va enregistrer les patients à l'aide de l'interface patientRepository

Et on définit le port de tomcat et la base de données h2 après on active la console de h2

```
spring.application.name=hopital
server.port=8084
spring.datasource.url=jdbc:h2:mem:patients-db
spring.h2.console.enabled=true
```

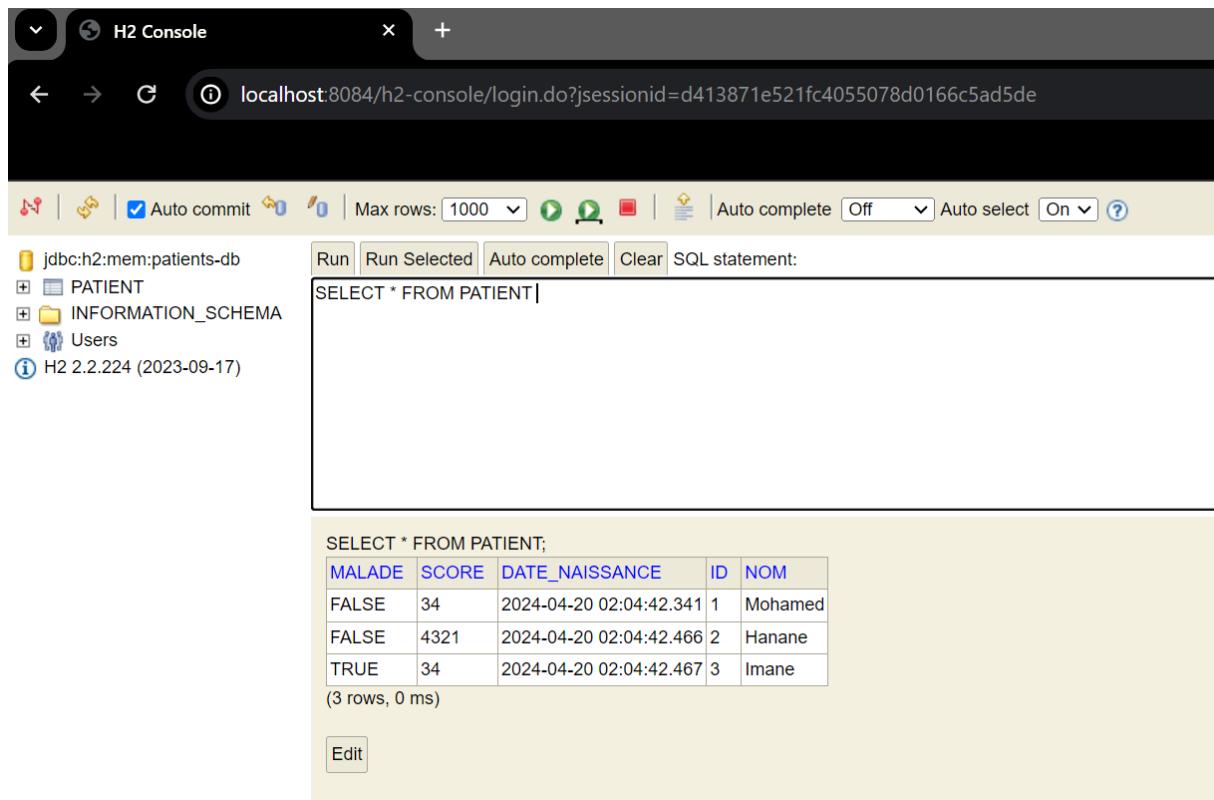


```
java -> PatientRepository.java <- HopitalApplication.java <- PatientController.java <- patients.html <- application.properties <-
package ma.fsm.hopital;
import ...;

@SpringBootApplication
public class HopitalApplication implements CommandLineRunner {
    @Autowired
    private PatientRepository patientRepository;

    public static void main(String[] args) {
        SpringApplication.run(HopitalApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        patientRepository.save(new Patient(id: null, nom: "Mohamed", new Date(), malade: false, score: 34));
        patientRepository.save(new Patient(id: null, nom: "Hanane", new Date(), malade: false, score: 4321));
        patientRepository.save(new Patient(id: null, nom: "Imane", new Date(), malade: true, score: 34));
    }
}
```



H2 Console

localhost:8084/h2-console/login.do?jsessionid=d413871e521fc4055078d0166c5ad5de

Auto commit | Max rows: 1000 | Run | Run Selected | Auto complete | Clear | SQL statement: SELECT * FROM PATIENT

MALADE	SCORE	DATE_NAISSANCE	ID	NOM
FALSE	34	2024-04-20 02:04:42.341	1	Mohamed
FALSE	4321	2024-04-20 02:04:42.466	2	Hanane
TRUE	34	2024-04-20 02:04:42.467	3	Imane

(3 rows, 0 ms)

Edit

On va créer la classe PatientContrôleur et pour qu'il soit contrôleur on utilise la notation `@Controller` qui a besoin de faire appel à l'interface `PatientRepository` et injecter le constructeur par la notation `@AllArgsConstructor`

On crée une méthode index qui retourne une vue patients et on faire rappel à cette méthode par la notation `@GetMapping`. Dans Template, on va créer le file html « patient.html »

Pour afficher la liste des patients, on va créer une liste et on va le stocker dans le modèle et pour afficher la liste des patients on a besoin la librairie thymleaf

On va afficher la liste sous forme d'un tableau

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h2>Liste patients</h2>
    <table>
        <thead>
            <tr>
                <th>ID</th> <th>Nom</th> <th>Date</th> <th>Malade</th> <th>Score</th>
            </tr>
            <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>
            
```

html > body > table > thead > tr > td

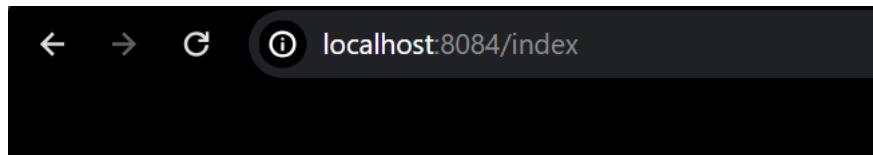
```

import ma.fsm.hopital.repository.PatientRepository;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import java.util.List;

@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping("/index")
    public String index(Model model) {
        List<Patient> patientList= patientRepository.findAll();
        model.addAttribute( attributeName: "listPatients", patientList);
        return "patients";
    }
}

```



Liste patients

ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-20 03:58:22.771	false	34
2	Hanane	2024-04-20 03:58:22.821	false	4321
3	Imane	2024-04-20 03:58:22.821	true	34

On va travailler avec bootstrap en ajoutant une dépendance

```

</dependency>
<!-- https://mvnrepository.com/artifact/org.webjars/bootstrap -->
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>5.3.3</version>
</dependency>

```

On affiche une table bootstrap

```

<meta charset="UTF-8">
<title>Title</title>
<link rel="stylesheet" href="/webjars/bootstrap/5.3.3/css/bootstrap.min.css">
<head>

```

Liste patients

ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-20 13:49:21.429	false	34
2	Hanane	2024-04-20 13:49:21.522	false	4321
3	Imane	2024-04-20 13:49:21.523	true	34

Pour nous pas redémarrer à chaque fois l'application, du moment qu'on a utilisé devtools on a deux choses pour les activer :

- On cherche Preferences -> Advanced Settings, et on s'assure que le premier choix dans Compiler est cocher.
- Dans Build, Exécution, Deployment cocher Build project Automaticaly.

On affiche la table bootstrap avec le card

```

<body>
    <div>
        <div class="card">
            <div class="card-header">Liste Patients</div>
            <div class="card-body">
                <table class="table">
                    <thead>
                        <tr>
                            <th>ID</th> <th>Nom</th> <th>Date</th> <th>Malade</th> <th>Score</th>
                        </tr>
                    <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>
                    </thead>
                </table>
            </div>
        </div>
    </div>

```

html > body > div > div.card > div.card-body > table.table > thead > tr > td

ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-20 14:34:59.637	false	34
2	Hanane	2024-04-20 14:34:59.685	false	4321
3	Imane	2024-04-20 14:34:59.686	true	34

On veut basculer de la base de données h2 à MySQL pour faire la pagination.

On doit télécharger les dépendances MySQL, et on change la base de données dans le fichier application.properties

```

        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>

```

```

spring.application.name=hôpital
server.port=8084
#spring.datasource.url=jdbc:h2:mem:patients-db
#spring.h2.console.enabled=true
spring.datasource.url=jdbc:mysql://localhost:3306/hôpital?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect

```

L'affichage :

Affichage des lignes 0 - 2 (total de 3, traitement en 0,0003 seconde(s).)

`SELECT * FROM `patient``

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Crée le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans cette table | Trier par clé : Aucun

Options supplémentaires

	malade	score	date_naissance	id	nom
<input type="checkbox"/>	Éditer	Copier	Supprimer	0	34 2024-04-20 15:29:43.000000 1 Mohamed
<input type="checkbox"/>	Éditer	Copier	Supprimer	0	4321 2024-04-20 15:29:43.000000 2 Hanane
<input type="checkbox"/>	Éditer	Copier	Supprimer	1	34 2024-04-20 15:29:43.000000 3 Imane

↑ □ Tout masquer Avec la sélection : □ Éditer □ Copier □ Supprimer □ Fournir

On redémarre l'application, elle nous rajoute les données car on n'a pas écrasé la base de données, et c'est fait exprès car on a besoin de plusieurs données pour qu'on puisse faire la pagination

```

spring.application.name=hôpital
server.port=8084
#spring.datasource.url=jdbc:h2:mem:patients-db
#spring.h2.console.enabled=true
spring.datasource.url=jdbc:mysql://localhost:3306/hôpital?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect

```

✓ Affichage des lignes 0 - 5 (total de 6, traitement en 0,0004 seconde(s).)

SELECT * FROM `patient`

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 ▾ Filtrer les lignes: Chercher dans cette table Trier par clé :

Options supplémentaires

	←	→	▼	malade	score	date_naissance	id	nom
<input type="checkbox"/>		Éditer		Copier		Supprimer	0	34 2024-04-20 15:32:35.000000 1 Mohamed
<input type="checkbox"/>		Éditer		Copier		Supprimer	0	4321 2024-04-20 15:32:35.000000 2 Hanane
<input type="checkbox"/>		Éditer		Copier		Supprimer	1	34 2024-04-20 15:32:35.000000 3 Imane
<input type="checkbox"/>		Éditer		Copier		Supprimer	0	34 2024-04-20 15:34:03.000000 4 Mohamed
<input type="checkbox"/>		Éditer		Copier		Supprimer	0	4321 2024-04-20 15:34:03.000000 5 Hanane
<input type="checkbox"/>		Éditer		Copier		Supprimer	1	34 2024-04-20 15:34:03.000000 6 Imane

Liste Patients

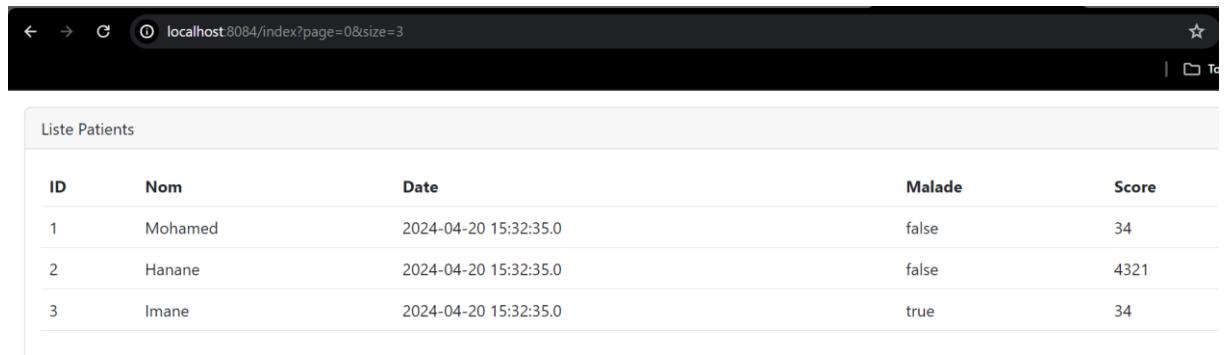
ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-20 15:32:35.0	false	34
2	Hanane	2024-04-20 15:32:35.0	false	4321
3	Imane	2024-04-20 15:32:35.0	true	34
4	Mohamed	2024-04-20 15:34:03.0	false	34
5	Hanane	2024-04-20 15:34:03.0	false	4321
6	Imane	2024-04-20 15:34:03.0	true	34

On fait la pagination :

```
ya × HopitalApplication.java × pom.xml (hopital) × PatientController.java × patients.html × application.pro
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import java.util.List;

@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping("/index")
    public String index(Model model, int page, int size) {
        Page<Patient> pagePatients= patientRepository.findAll(PageRequest.of(page,size));
        model.addAttribute( attributeName: "listPatients", pagePatients.getContent());
        return "patients";
    }
}
```



ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-20 15:32:35.0	false	34
2	Hanane	2024-04-20 15:32:35.0	false	4321
3	Imane	2024-04-20 15:32:35.0	true	34

On utilise la notation `@RequestParam` pour faire la correspondance entre les paramètres qu'on créer dans l'url et celui qu'on a déclaré.

Et pour faire la pagination on a stocké le nombre de pages dans un model

The screenshot shows a Java IDE with several tabs open. The active tab is PatientController.java, which contains the following code:

```
import java.util.List;

@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping("/index")
    public String index(Model model, @RequestParam(name = "page", defaultValue = "0") int p, @RequestParam(name = "size", defaultValue = "4") int s) {
        Page<Patient> pagePatients = patientRepository.findAll(PageRequest.of(p, s));
        model.addAttribute("listPatients", pagePatients.getContent());
        model.addAttribute("pages", new int[pagePatients.getTotalPages()]);
        return "patients";
    }
}
```

On a ajouté des boutons pour se déplacer d'une page vers l'autre dans le fichier patients.html sous forme d'une liste ul

The screenshot shows a Java IDE with patients.html selected. The file contains the following HTML code:

```
<tr>
    <th>ID</th> <th>Nom</th> <th>Date</th> <th>Malade</th> <th>Score</th>
</tr>
<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>
</thead>
</table>


- 

```

Dans l'affichage, on bascule d'une page vers l'autre

← → ⌂ localhost:8084/index?page=5

Liste Patients					
ID	Nom	Date	Malade	Score	
21	Imane	2024-04-20 16:09:51.0	true	34	
22	Mohamed	2024-04-20 16:11:15.0	false	34	
23	Hanane	2024-04-20 16:11:15.0	false	4321	
24	Imane	2024-04-20 16:11:15.0	true	34	

0 1 2 3 4 5 6

On veut que le bouton de la page courante soit colorié, alors on va stocker dans le model la page courante

```
Application.java × pom.xml (hopital) × PatientController.java × patients.html × application.properties ×
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.List;

@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping("/index")
    public String index(Model model, @RequestParam(name = "page", defaultValue = "0")
        int p, @RequestParam(name = "size", defaultValue = "4") int s) {
        Page<Patient> pagePatients= patientRepository.findAll(PageRequest.of(p,s));
        model.addAttribute( attributeName: "listPatients", pagePatients.getContent());
        model.addAttribute( attributeName: "pages", new int[pagePatients.getTotalPages()]);
        model.addAttribute( attributeName: "currentPage",p);
        return "patients";
    }
}
```

```


| ID | Nom     | Date                  | Malade | Score |
|----|---------|-----------------------|--------|-------|
| 17 | Hanane  | 2024-04-20 16:08:20.0 | false  | 4321  |
| 18 | Imane   | 2024-04-20 16:08:20.0 | true   | 34    |
| 19 | Mohamed | 2024-04-20 16:09:51.0 | false  | 34    |
| 20 | Hanane  | 2024-04-20 16:09:51.0 | false  | 4321  |



- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

```

Liste Patients

ID	Nom	Date	Malade	Score
17	Hanane	2024-04-20 16:08:20.0	false	4321
18	Imane	2024-04-20 16:08:20.0	true	34
19	Mohamed	2024-04-20 16:09:51.0	false	34
20	Hanane	2024-04-20 16:09:51.0	false	4321

0 1 2 3 4 5 6 7 8

On a effectué la pagination donc on va ajouter une partie de la recherche des patients

On va créer un formulaire

The screenshot shows a code editor with several tabs at the top: PatientRepository.java, HopitalApplication.java, pom.xml (hopital), PatientController.java, and patients.html. The patients.html tab is active, displaying an HTML template for a patient list page. The template includes a header with a link to a stylesheet, a search form with a keyword input and a 'Chercher' button, and a table with columns for ID, Nom, Date, Malade, and Score. The table rows are generated by a Thymeleaf loop. The code editor's navigation bar at the bottom shows the path: html > body > div.p-3 > div.card > div.card-body > form.

```
<link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css" />
</head>
<body>
<div class="p-3">
    <div class="card">
        <div class="card-header">Liste Patients</div>
        <div class="card-body">
            <form method="get" th:action="@{index}">
                <label>Keyword:</label>
                <input type="text" name="keyword">
                <button type="submit" class="btn btn-info">Chercher</button>
            </form>
            <table class="table">
                <thead>
                    <tr>
                        <th>ID</th> <th>Nom</th> <th>Date</th> <th>Malade</th> <th>Score</th>
                    </tr>
                </thead>
                <tbody>
                    <tr th:each="p:${listPatients}">
                        <td th:text="${p.id}"></td>
                        <td th:text="${p.nom}"></td>
                        <td th:text="${p.dateNaissance}"></td>
                </tbody>
            </table>
        </div>
    </div>
</div>
```

The screenshot shows a web browser window with the URL localhost:8084/index?keyword=. The page title is "Liste Patients". It features a search form with a "Keyword:" input field and a "Chercher" button. Below the form is a table with columns: ID, Nom, Date, Malade, and Score. The table contains four rows of data. At the bottom of the table is a pagination control with buttons labeled 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10, where the "0" button is highlighted.

ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-20 15:32:35.0	false	34
2	Hanane	2024-04-20 15:32:35.0	false	4321
3	Imane	2024-04-20 15:32:35.0	true	34
4	Mohamed	2024-04-20 15:34:03.0	false	34

Pour faire la recherche on ajoute une méthode dans l'interface PatientRepository et on utilise le paramètre pageable pour faire la pagination

```

Patient.java × PatientRepository.java × HopitalApplication.java × pom.xml (hopital) × PatientController.java
package ma.fsm.hopital.repository;

import ma.fsm.hopital.entities.Patient;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

public interface PatientRepository extends JpaRepository<Patient, Long> {
    Page<Patient> findByNomContains(String keyword, Pageable pageable);
    @Query("select p from Patient p where p.nom like :x")
    Page<Patient> chercher(@Param("x") String keyword, Pageable pageable);

}

```

```

@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping("/index")
    public String index(Model model, @RequestParam(name = "page", defaultValue = "0")
        int p, @RequestParam(name = "size", defaultValue = "4") int s,
        @RequestParam(name="keyword", defaultValue="")String kw) {
        Page<Patient> pagePatients= patientRepository.findByNomContains(kw,PageRequest.of(p,s));
        model.addAttribute( attributeName: "listPatients", pagePatients.getContent());
        model.addAttribute( attributeName: "pages", new int[pagePatients.getTotalPages()]);
        model.addAttribute( attributeName: "currentPage",p);
        return "patients";
    }
}

```



Liste Patients					
Keyword:		Chercher			
ID	Nom	Date	Malade	Score	
3	Imane	2024-04-20 15:32:35.0	true	34	
6	Imane	2024-04-20 15:34:03.0	true	34	
9	Imane	2024-04-20 15:47:23.0	true	34	
12	Imane	2024-04-20 15:54:23.0	true	34	
0 1 2 3					

On veut que la valeur rechercher reste dans la zone de texte, alors ce qu'on fait et de stocker dans le model le keyword, car c'est lui qu'on va transmis dans la zone de texte

```
<div class="p-3">
    <div class="card">
        <div class="card-header">Liste Patients</div>
        <div class="card-body">
            <form method="get" th:action="@{index}">
                <label>Keyword:</label>
                <input type="text" name="keyword" th:value="${keyword}">
                <button type="submit" class="btn btn-info">Chercher</button>
            </form>
            <table class="table">
                <thead>
                    <tr>
```

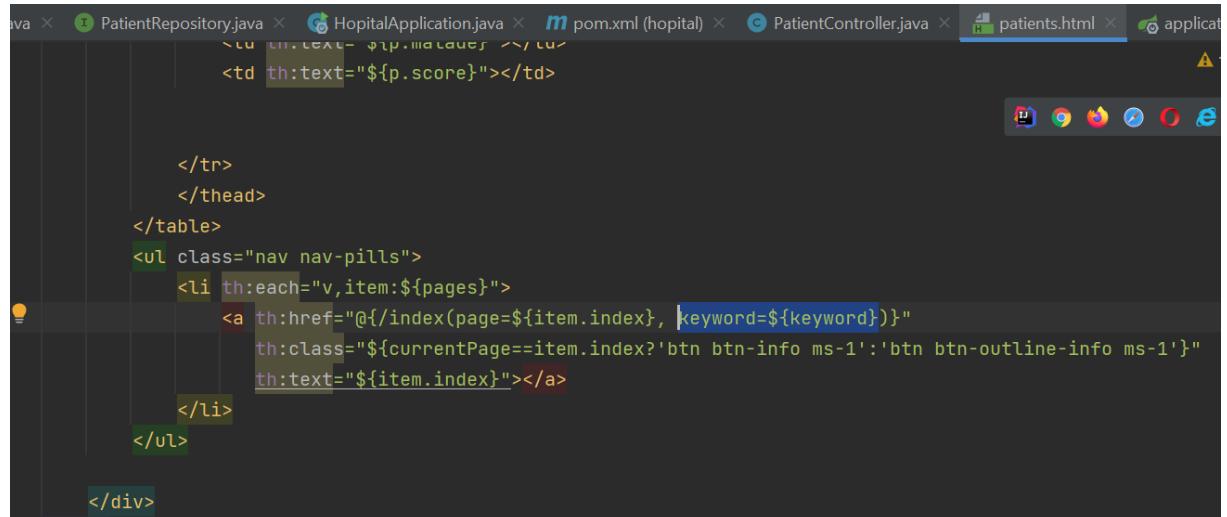
```
@Controller
@AllArgsConstructor
public class PatientController {
    private PatientRepository patientRepository;
    @GetMapping("/index")
    public String index(Model model, @RequestParam(name = "page", defaultValue = "0") int p, @RequestParam(name = "size", defaultValue = "4") int s, @RequestParam(name = "keyword", defaultValue = "") String kw) {
        Page<Patient> pagePatients = patientRepository.findByNomContains(kw, PageRequest.of(p, s));
        model.addAttribute("listPatients", pagePatients.getContent());
        model.addAttribute("pages", new int[pagePatients.getTotalPages()]);
        model.addAttribute("currentPage", p);
        model.addAttribute("keyword", kw);
        return "patients";
    }
}
```



Liste Patients					
ID	Nom	Date	Malade	Score	
3	Imane	2024-04-20 15:32:35.0	true	34	
6	Imane	2024-04-20 15:34:03.0	true	34	
9	Imane	2024-04-20 15:47:23.0	true	34	
12	Imane	2024-04-20 15:54:23.0	true	34	

0 1 2 3

Lorsqu'on appuie sur un bouton d'une autre page il revient à afficher tous les patients car le keyword par défaut c'est vide et non pas celles recherchées, si on ajoute le keyword dans le lien ça va marcher



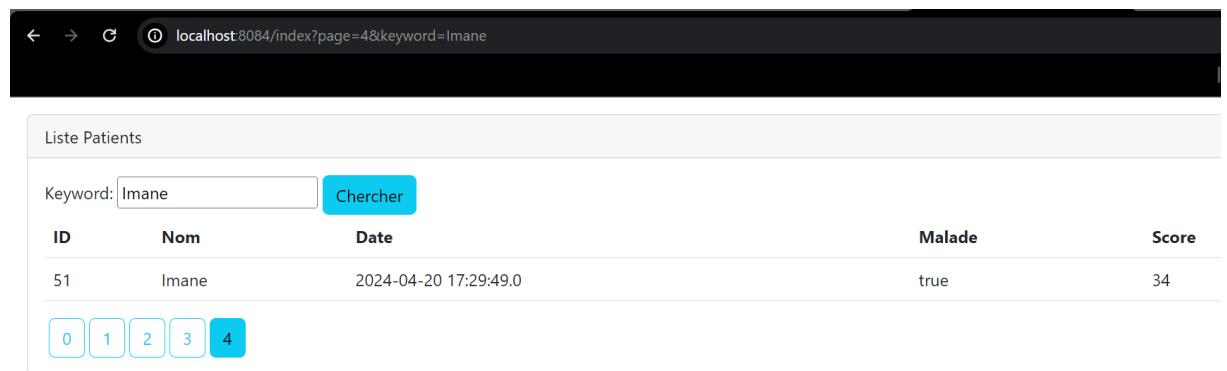
```


| ID | Nom   | Date                  | Malade | Score |
|----|-------|-----------------------|--------|-------|
| 51 | Imane | 2024-04-20 17:29:49.0 | true   | 34    |

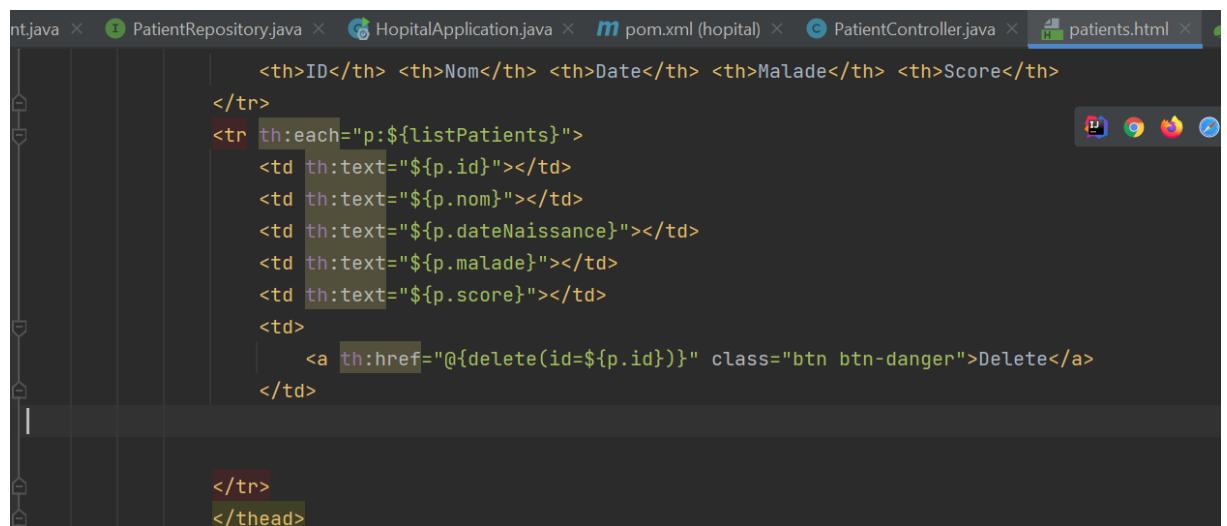


-


```



On veut supprimer un patient, tout d'abord on va créer le bouton delete, puis on ajoute la fonction delete dans le contrôleur



```


| ID | Nom   | Date                  | Malade | Score |
|----|-------|-----------------------|--------|-------|
| 51 | Imane | 2024-04-20 17:29:49.0 | true   | 34    |


```

```

    Page<Patient> pagePatients= patientRepository.findByNomContains(kw,PageRequest.of(p,s))
    model.addAttribute( attributeName: "listPatients", pagePatients.getContent());
    model.addAttribute( attributeName: "pages", new int[pagePatients.getTotalPages()]);
    model.addAttribute( attributeName: "currentPage",p);
    model.addAttribute( attributeName: "keyword", kw);
    return "patients";
}
@GetMapping("/delete")
public String delete(Long id) {
    patientRepository.deleteById(id);
    return "redirect:/index";
}
}

```

On a supprimé les deux premiers patients

ID	Nom	Date	Malade	Score										
3	Imane	2024-04-20 15:32:35.0	true	34	<button>Delete</button>									
4	Mohamed	2024-04-20 15:34:03.0	false	34	<button>Delete</button>									
5	Hanane	2024-04-20 15:34:03.0	false	4321	<button>Delete</button>									
6	Imane	2024-04-20 15:34:03.0	true	34	<button>Delete</button>									
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

On affiche un message de confirmation qui permet de confirmer la suppression

```

<td th:text="${p.score}"></td>
<td>
    <a onclick="return confirm('Etes vous sure?')" th:href="@{delete(id=${p.id})}" class="btn btn-danger">Delete</a>
</td>

```

Liste Patients

Keyword: Chercher

ID	Nom	Date	Malade	Score	
3	Imane	2024-04-20 15:32:35.0	true	34	<button>Delete</button>
5	Hanane	2024-04-20 15:34:03.0	false	4321	<button>Delete</button>
6	Imane	2024-04-20 15:34:03.0	true	34	<button>Delete</button>
7	Mohamed	2024-04-20 15:47:23.0	false	34	<button>Delete</button>

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Lorsqu'on supprime un patient, on perd le mot clé et on a résoudre le problème de la même manière qu'on a fait précédemment

```

<th>ID</th> <th>Nom</th> <th>Date</th> <th>Malade</th> <th>Score</th>
</tr>
<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>
    <td>
        <a onclick="javascript:return confirm('Etes vous sure?')"
            th:href="@{delete(id=${p.id}, keyword=${keyword}, page=${currentPage})}" class="btn btn-danger">Delete</a>
    </td>
</tr>
</thead>
</table>

```

html > body > div.p-3 > div.card > div.card-body > table.table > thead > tr > td > a.btn.btn-danger

```

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

```

On a supprimé le patient de l'identifiant 33 et de keyword Imane de la page 2

The screenshot shows a web browser window with the URL `localhost:8084/index?page=2&keyword=Imane`. A modal dialog box is displayed in the center, asking "Etes vous sur?" (Are you sure?). The dialog has two buttons: "OK" and "Annuler" (Cancel). The background page shows a table titled "Liste Patients" with columns: ID, Nom, Date, Malade, and Score. There are four rows of data, each with a "Delete" button. The first row's data is: ID 27, Nom Imane, Date 2024-04-20 16:13:50.0, Malade true, Score 34. The last row's data is: ID 36, Nom Imane, Date 2024-04-20 16:45:19.0, Malade true, Score 34. Below the table is a navigation bar with buttons 0 through 5.

ID	Nom	Date	Malade	Score
27	Imane	2024-04-20 16:13:50.0	true	34
30	Imane	2024-04-20 16:25:30.0	true	34
33	Imane	2024-04-20 16:30:34.0	true	34
36	Imane	2024-04-20 16:45:19.0	true	34

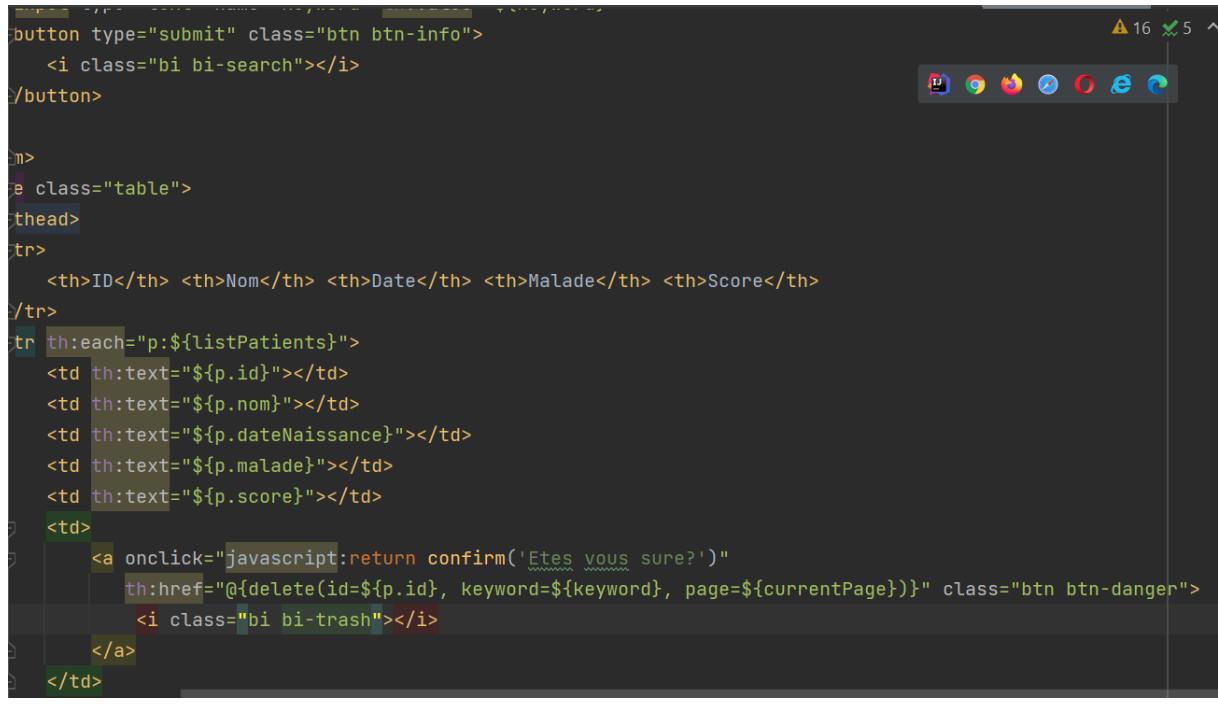
This screenshot shows the same web application after the confirmation dialog was closed. The table now only contains three rows of data, corresponding to the rows where the "Delete" button was not clicked in the previous screenshot. The last row's data is: ID 39, Nom Imane, Date 2024-04-20 16:45:54.0, Malade true, Score 34.

ID	Nom	Date	Malade	Score
27	Imane	2024-04-20 16:13:50.0	true	34
30	Imane	2024-04-20 16:25:30.0	true	34
36	Imane	2024-04-20 16:45:19.0	true	34

On améliore notre application par des icônes, tout d'abord on ajoute la dépendance bootstrap-icons

```
    </dependencies>
<!-- https://mvnrepository.com/artifact/org.webjars.npm/bootstrap-icons -->
<dependency>
    <groupId>org.webjars.npm</groupId>
    <artifactId>bootstrap-icons</artifactId>
    <version>1.10.3</version>
</dependency>
```

Au lieu d'écrire chercher et delete en utilisant leur icons

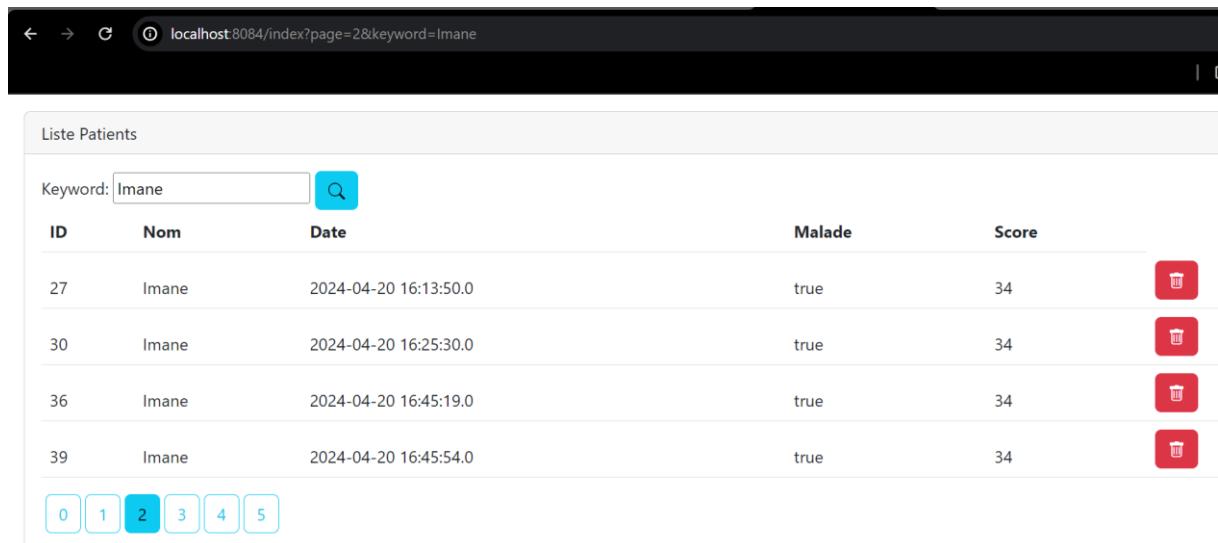


```

button type="submit" class="btn btn-info">
    <i class="bi bi-search"></i>


<table>
    <thead>
        <tr>
            <th>ID</th> <th>Nom</th> <th>Date</th> <th>Malade</th> <th>Score</th>
        </tr>
    </thead>
    <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>
            <td>
                <a onclick="javascript:return confirm('Etes vous sure?')"
                    th:href="@{delete(id=${p.id}, keyword=${keyword}, page=${currentPage})}" class="btn btn-danger">
                    <i class="bi bi-trash"></i>
                </a>
            </td>
        </tr>
    </tbody>

```



ID	Nom	Date	Malade	Score
27	Imane	2024-04-20 16:13:50.0	true	34
30	Imane	2024-04-20 16:25:30.0	true	34
36	Imane	2024-04-20 16:45:19.0	true	34
39	Imane	2024-04-20 16:45:54.0	true	34

0 1 2 3 4 5

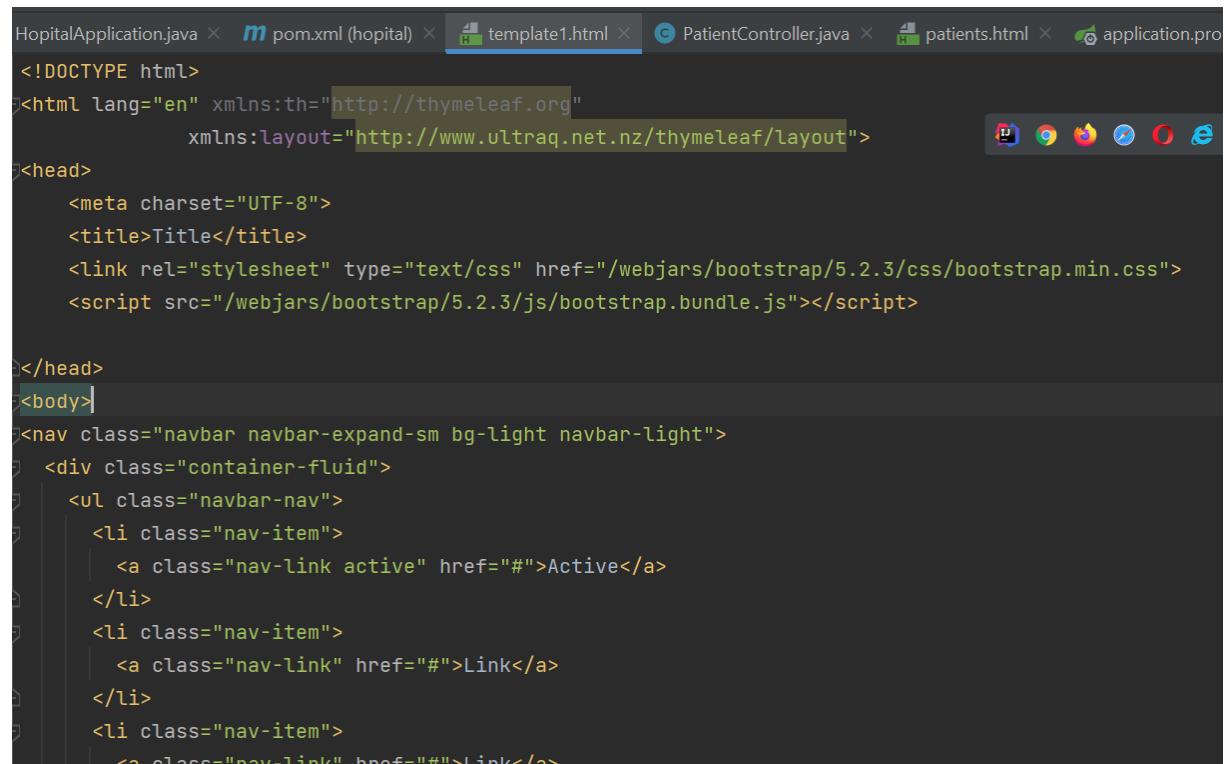
Partie 2 :

Dans cette partie, on va créer une page template et faire la validation des formulaires

On ajoute les dépendances de thymleaf layout dialect pour avoir un header fixe dans toutes les pages sans faire copier-coller à chaque fois

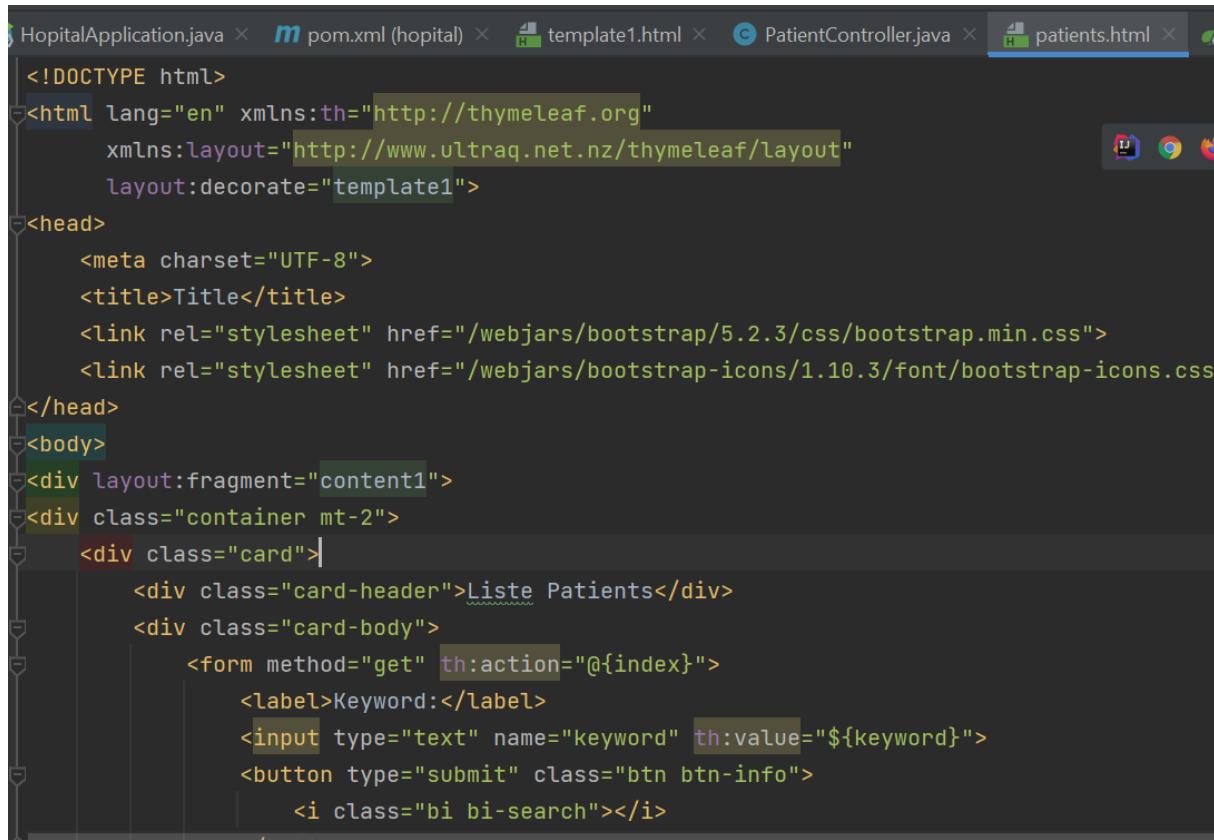
```
</dependency>
<!-- https://mvnrepository.com/artifact/nz.net.ultraq.thymeleaf/thymeleaf-layout-dialect -->
<dependency>
    <groupId>nz.net.ultraq.thymeleaf</groupId>
    <artifactId>thymeleaf-layout-dialect</artifactId>
    <version>3.1.0</version>
</dependency>
```

On crée une page template1.html et on ajoute un navbar de bootstrap 5 et navbar with dropdown puis on l'ajoute dans la vue patients.html, c'est comme on dit décorer la vue avec template1, et on ajoute aussi le content1

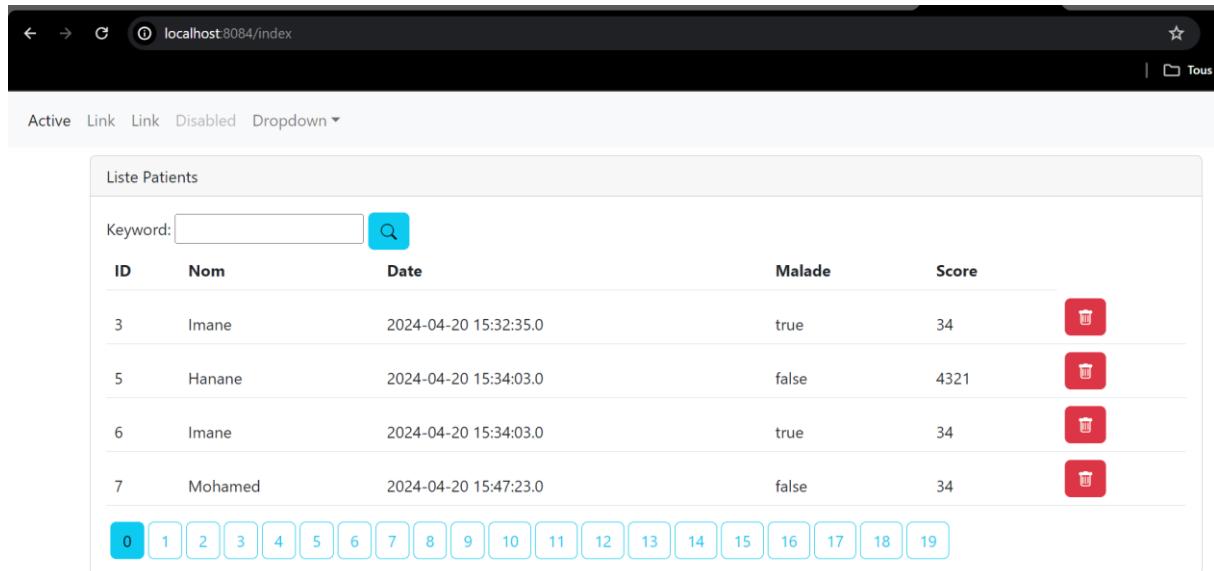


```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" type="text/css" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
    <script src="/webjars/bootstrap/5.2.3/js/bootstrap.bundle.js"></script>
</head>
<body>
<nav class="navbar navbar-expand-sm bg-light navbar-light">
    <div class="container-fluid">
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link active" href="#">Active</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Link</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Link</a>
            </li>
        </ul>
    </div>
</nav>

```



```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="template1">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
    <link rel="stylesheet" href="/webjars/bootstrap-icons/1.10.3/font/bootstrap-icons.css">
</head>
<body>
    <div layout:fragment="content1">
        <div class="container mt-2">
            <div class="card">
                <div class="card-header">Liste Patients</div>
                <div class="card-body">
                    <form method="get" th:action="@{index}">
                        <label>Keyword:</label>
                        <input type="text" name="keyword" th:value="${keyword}">
                        <button type="submit" class="btn btn-info">
                            <i class="bi bi-search"></i>
                        </button>
                    </form>
                </div>
            </div>
        </div>
    </div>
</body>
```



localhost:8084/index

Active Link Link Disabled Dropdown ▾

Liste Patients

Keyword: 

ID	Nom	Date	Malade	Score	Action
3	Imane	2024-04-20 15:32:35.0	true	34	
5	Hanane	2024-04-20 15:34:03.0	false	4321	
6	Imane	2024-04-20 15:34:03.0	true	34	
7	Mohamed	2024-04-20 15:47:23.0	false	34	

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

On va changer l'arrière-plan du header et les éléments pour qu'on puisse faire ajouter un nouveau patient, chercher un patient, et logout

```

    </li>
<li class="nav-item">
    <a class="nav-link" href="#">Link</a>
</li>
<li class="nav-item">
    <a class="nav-link disabled" href="#">Disabled</a>
</li>
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Patients</a>
    <ul class="dropdown-menu">
        <li><a class="dropdown-item" th:href="@{formPatients}">Nouveau</a></li>
        <li><a class="dropdown-item" href="@{index}">Chercher</a></li>
    </ul>
</li>
</ul>
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">[Username]</a>
    <ul class="dropdown-menu">
        <li><a class="dropdown-item" th:href="@{formPatients}">Logout</a></li>
    </ul>
</li>

```

The screenshot shows a web browser window with the URL `localhost:8084/index`. The navigation bar includes links for Home, Link, Disabled, Patients (with a dropdown menu), and [Username] (with a dropdown menu). Below the navigation bar is a search form with a keyword input field and a search button. The main content area displays a table titled "Liste Patients" with the following data:

ID	Nom	Date	Malade	Score	Action
3	Imane	2024-04-20 15:32:35.0	true	34	
5	Hanane	2024-04-20 15:34:03.0	false	4321	
6	Imane	2024-04-20 15:34:03.0	true	34	
7	Mohamed	2024-04-20 15:47:23.0	false	34	

Below the table is a page navigation bar with buttons numbered 0 through 24.

ID	Nom	Date	Malade	Score
3	Imane	2024-04-20 15:32:35.0	true	34
5	Hanane	2024-04-20 15:34:03.0	false	4321
6	Imane	2024-04-20 15:34:03.0	true	34
7	Mohamed	2024-04-20 15:47:23.0	false	34

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25

On ajoute une méthode formPatients qui permet d'ajouter un patient dans la classe contrôleur puis on va créer un fichier HTML que l'on appelle formPatients.html, pour générer la vue du formulaire d'ajout

```

        model.addAttribute( attributeName: "listPatients", pagePatients.getContent());
        model.addAttribute( attributeName: "pages", new int[pagePatients.getTotalPages()]);
        model.addAttribute( attributeName: "currentPage",p);
        model.addAttribute( attributeName: "keyword", kw);
        return "patients";
    }

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

    @GetMapping("/patients")
    @ResponseBody
    public List<Patient> listPatients() {
        return patientRepository.findAll();
    }

    @GetMapping("/formPatients")
    public String formPatient(Model model) {
        model.addAttribute( attributeName: "patient", new Patient());
        return "formPatients";
    }
}

```

A screenshot of a code editor displaying a Java Thymeleaf template file. The code defines a form for a patient with fields for Nom, Date Naissance, Malade (checkbox), and Score. The Nom field has a value of \${patient.nom} and an error message of \${patient.nom}. The Date Naissance field is a date input with a value of \${patient.dateNaissance} and an error message of \${patient.dateNaissance}. The Malade field is a checkbox with a checked value of \${patient.malade} and an error message of \${patient.malade}. The Score field has a value of \${patient.score}.

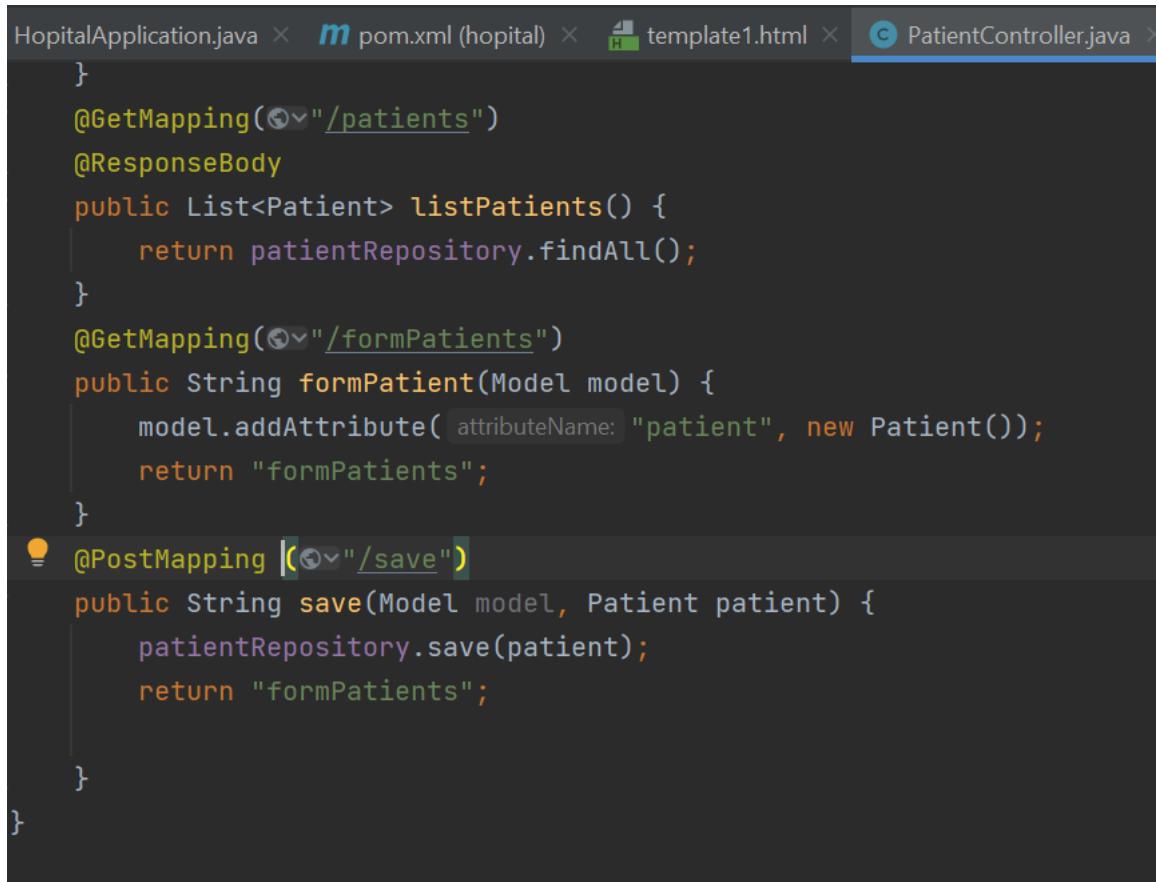
```
<div class="col-md-6 offset-3">
<form method="post" th:action="@{save}">
    <div>
        <label for="nom">Nom</label>
        <input id="nom" class="form-control" type="text" name="nom" th:value="${patient.nom}" th:errors="${patient.nom}">
    </div>
    <div>
        <label>Date Naissance</label>
        <input class="form-control" type="date" name="nom" th:value="${patient.dateNaissance}" th:errors="${patient.dateNaissance}">
    </div>
    <div>
        <label>Malade</label>
        <input type="checkbox" name="malade" th:checked="${patient.malade}">
        <span th:errors="${patient.malade}"></span>
    </div>
    <div>
        <label>Score</label>
        <input class="form-control" type="text" name="malade" th:value="${patient.score}">
    </div>

```

A screenshot of a web browser displaying a patient form. The URL is localhost:8084/formPatients. The form contains fields for Nom, Date Naissance (with a date input and calendar icon), Malade (checkbox), and Score (with a numeric input). A Save button is at the bottom.

Nom
Date Naissance
Malade
Score
Save

On crée une méthode save() qui permet d'ajouter un patient dans la classe contrôleur



The screenshot shows a Java code editor with the file `PatientController.java` open. The code defines a controller for managing patients. It includes three methods: `listPatients()`, `formPatient(Model model)`, and `save(Model model, Patient patient)`. The `listPatients()` method returns a list of patients from the repository. The `formPatient()` method adds a new patient model to the view. The `save()` method saves the updated patient back to the repository.

```
 HopitalApplication.java × pom.xml (hopital) × template1.html × PatientController.java >
}
@GetMapping("/patients")
@ResponseBody
public List<Patient> listPatients() {
    return patientRepository.findAll();
}

@GetMapping("/formPatients")
public String formPatient(Model model) {
    model.addAttribute(attributeName: "patient", new Patient());
    return "formPatients";
}

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

On utilise la notation `@DateTimeFormat` pour spécifier format des dates

```

import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.format.annotation.DateTimeFormat;

import java.util.Date;
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Patient {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    @Temporal(TemporalType.DATE)
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date dateNaissance;
    private boolean malade;
    private int score;
}

```

On ajoute un patient

localhost:8084/formPatients

Home Link Link Disabled Patients ▾

Nom	Mansouri
Date Naissance	05/12/2023 <input type="button" value="Calendar"/>
Malade	<input checked="" type="checkbox"/>
Score	34

Liste Patients																									
Keyword:	<input type="text"/>	<input type="button" value="Search"/>																							
ID	Nom	Date	Malade	Score																					
141	Hanane	2024-04-21	false	4321																					
142	Imane	2024-04-21	true	34																					
144	Mansouri	2023-12-05	true	34																					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	
25	26	27	28	29	30	31	32	33																	

On fait la validation en ajoutant la dépendance spring boot validation dans pom.xml

```
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-validation -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
    <version>2.6.5</version>
</dependency>
```

On ajoute des notations à la classe Patient commençant par `@NotEmpty` c'est-à-dire qu'on n'accepte pas que ce champ soit vide, `@Size` pour définir le minimum et la maximum des caractères saisies et `@DecimalMin` pour qu'on dire par exemple qu'on n'accepte pas un score inférieur de la valeur cité

On ajoute la notation `@Valid` pour qu'on dire à spring mvc une fois que tu stocke les données dans un patient et faire la validation, et si jamais il y a des erreurs qui va nous sauter la connexion des erreurs dans une connexion de type `BindingResult`

```
    @GetMapping("/formPatients")
    public String formPatient(Model model) {
        model.addAttribute("patient", new Patient());
        return "formPatients";
    }
    @PostMapping (path = "/save")
    public String save(Model model, @Valid Patient patient, BindingResult bindingResult) {
        if(bindingResult.hasErrors()) return "formPatients";
        patientRepository.save(patient);
        return "redirect:/formPatients";
    }
}
```

```
import org.springframework.format.annotation.DateTimeFormat;

import java.util.Date;
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Patient {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotEmpty
    @Size(min = 4, max = 40)
    private String nom;
    @Temporal(TemporalType.DATE)
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date dateNaissance;
    private boolean malade;
    @DecimalMin("100")
    private int score;
}
```

localhost:8084/save

Home Link Link Disabled Patients ▾

Nom

ne doit pas être vide
la taille doit être comprise entre 4 et 40

Date Naissance

jj/mm/aaaa

Malade

Score

0

doit être supérieur ou égal à 100

Save

On met les messages en rouge

localhost:8084/save

Home Link Link Disabled Patients ▾

Nom

ne doit pas être vide
la taille doit être comprise entre 4 et 40

Date Naissance

jj/mm/aaaa

Malade

Score

0

doit être supérieur ou égal à 100

Save

On ajoute un bouton Edit

```

</td>
<td>
<a href="@{editPatient(id=${p.id})}" class="btn btn-success">
    Edit
</a>
</td>
```

ID	Nom	Date	Malade	Score		
3	Imane	2024-04-20	true	34		
5	Hanane	2024-04-20	false	4321		
6	Imane	2024-04-20	true	34		
7	Mohamed	2024-04-20	false	34		

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37

On ajoute la méthode editPatient au contrôleur pour modifier les informations d'un patient

```

    @GetMapping("/formPatients")
    public String formPatient(Model model) {
        model.addAttribute("patient", new Patient());
        return "formPatients";
    }
    @PostMapping (path = "/save")
    public String save(Model model, @Valid Patient patient, BindingResult bindingResult) {
        if(bindingResult.hasErrors()) return "formPatients";
        patientRepository.save(patient);
        return "redirect:/formPatients";

    }
    @GetMapping("/editPatients")
    public String editPatient(Model model, Long id) {
        Patient patient=patientRepository.findById(id).orElse( other: null);
        if(patient==null) throw new RuntimeException("Patient introuvable");
        model.addAttribute("patient");
        return "editPatients";
    }
}

```

On va créer un fichier HTML editPatients.html

```
<link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
<link rel="stylesheet" href="/webjars/bootstrap-icons/1.10.3/font/bootstrap-icons.css">

```

A 17 ^

```
</head>
<body>
<div layout:fragment="content1">
<div class="col-md-6 offset-3">
<form method="post" th:action="@{save}">
<div>
    <label>ID</label>
    <input type="text" name="id" th:value="${patient.id}">
    <span class="text-danger" th:errors="${patient.id}"></span>
</div>
<div>
    <label for="nom">Nom</label>
    <input id="nom" type="text" name="nom" th:value="${patient.nom}">
    <span class="text-danger" th:errors="${patient.nom}"></span>
</div>
<div>
    <label>Date Naissance</label>
    <input type="date" name="dateNaissance" th:value="${patient.dateNaissance}">
    <span class="text-danger" th:errors="${patient.dateNaissance}"></span>
</div>

```

On essaie la modification de la ligne 2 avec l'id 5

localhost:8084/editPatients?id=5

Home Link Link Disabled Patients ▾

ID
5

Nom
Hanane

Date Naissance
20/04/2024

Malade

Score
4321

Save

localhost:8084/editPatients?id=5

Home Link Link Disabled Patients ▾

ID	5
Nom	Salma
Date Naissance	20/04/2024
Malade	<input type="checkbox"/>
Score	120
<input type="button" value="Save"/>	

localhost:8084/index

Home Link Link Disabled Patients ▾ [User]

Liste Patients

ID	Nom	Date	Malade	Score																																																				
3	Imane	2024-04-20	true	34																																																				
5	Salma	2024-04-20	false	120																																																				
6	Imane	2024-04-20	true	34																																																				
7	Mohamed	2024-04-20	false	34																																																				
<table border="1"> <tr> <td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td> </tr> <tr> <td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td><td>32</td><td>33</td><td>34</td><td>35</td><td>36</td><td>37</td><td>38</td><td>39</td><td>40</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td> </tr> </table>							0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40									
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24																																
25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40																																									

On perd le keyword après qu'on effectue la modification, pour le résoudre on fait un petit changement dans patients.html et on revient au contrôleur pour stocker dans le model le keyword et la page

```

<td>
  <a href="@{editPatients(id=${p.id}, keyword=${keyword}, page=${currentPage})}" class="btn btn-success">
    Edit
  </a>
</td>

```

```

    @GetMapping("/editPatients")
    public String editPatient(Model model, Long id, String keyword, int page) {
        Patient patient=patientRepository.findById(id).orElse( other: null);
        if(patient==null) throw new RuntimeException("Patient introuvable");
        model.addAttribute( attributeName: "patient", patient);
        model.addAttribute( attributeName: "page",page);
        model.addAttribute( attributeName: "keyword", keyword);
        return "editPatients";
    }
}

```

On ajoute les paramètres page et keyword dans save au niveau de th:action et on les ajouter dans la méthode save dans le contrôleur

```

<div layout:fragment="content1">
    <div class="col-md-6 offset-3">
        <form method="post" th:action="@{save(page=${page},keyword=${keyword})}">
            <div>
                <label>ID</label>
                <input class="form-control" type="text" name="id" th:value="${patient.id}">
            </div>
        </form>
    </div>

```

```

    @PostMapping (path = "/save")
    public String save(Model model, @Valid Patient patient, BindingResult bindingResult,
                       @RequestParam(defaultValue = "0") int page,
                       @RequestParam(defaultValue = "") String keyword) {
        if(bindingResult.hasErrors()) return "formPatients";
        patientRepository.save(patient);

        return "redirect:/index?page="+page+"&keyword="+keyword;
    }
}

```

On modifie la ligne de l'id 19 dans la page 4

ID	Nom	Date	Malade	Score		
17	Hanane	2024-04-21	false	4321		
18	Imane	2024-04-21	true	349		
19	Mohamed	2024-04-21	false	344		
20	Hanane	2024-04-21	false	4321		



Liste Patients					
ID	Nom	Date	Malade	Score	
17	Hanane	2024-04-21	false	4321	
18	Imane	2024-04-21	true	349	
19	Mohamed	2024-04-21	true	400	
20	Hanane	2024-04-21	false	4321	

Keyword:

0 1 2 3 4 5 6

On cache la zone de texte de l'id et on affiche sa valeur mais on ne peut pas le modifier

```
<div>
    <label>ID :</label>
    <label th:text="${patient.id}"></label>
    <input class="form-control" type="hidden" name="id" th:value="${patient.id}">
    <span class="text-danger" th:errors="${patient.id}"></span>
</div>
```

ID : 1

Nom

Mohamed

Date Naissance

21/04/2024

Malade

Score

344

Save

Partie 3 :

Sécurité avec Spring security

On va travailler sur même projet que celui de la deuxième partie

Tout d'abord, on va ajouter la dépendance security

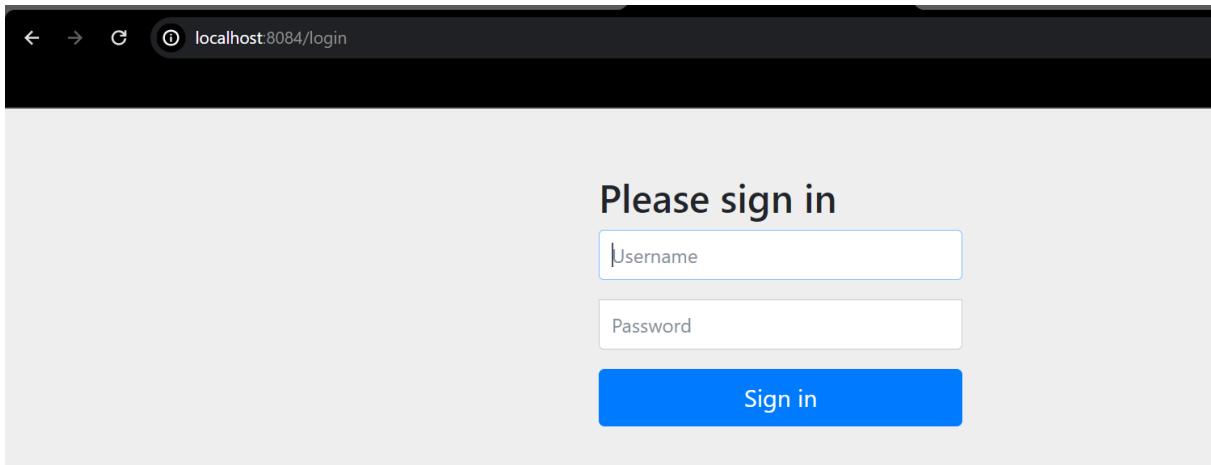
```

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

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>

```

On démarre l'application et quand on accède, on trouve qu'on a une configuration par défaut



Pour accéder à l'application on utilise un Username « user », et un mot de passe qui est généré

```
Console  Endpoints
2024-04-21T21:30:09.541+02:00  INFO 19512 --- [hopital] [ restartedMain] o.s.d.j.r.query.QueryEnhancerFactory      : Hibernate is in classpath; If
2024-04-21T21:30:09.900+02:00  WARN 19512 --- [hopital] [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is en
2024-04-21T21:30:10.301+02:00  WARN 19512 --- [hopital] [ restartedMain] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: 14dbe900-11e6-46b7-8dbd-4efe73838f6b

This generated password is for development use only. Your security configuration must be updated before running your application in production.

2024-04-21T21:30:10.414+02:00  INFO 19512 --- [hopital] [ restartedMain] o.s.s.web.DefaultSecurityFilterChain      : Will secure any request with
2024-04-21T21:30:10.453+02:00  INFO 19512 --- [hopital] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer      : LiveReload server is running
2024-04-21T21:30:10.483+02:00  INFO 19512 --- [hopital] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat started on port 8084 (
2024-04-21T21:30:10.492+02:00  INFO 19512 --- [hopital] [ restartedMain] m.a.fsm.hopital.HopitalApplication      : Started HopitalApplication in
2024-04-21T21:30:24.965+02:00  INFO 19512 --- [hopital] [nio-8084-exec-1] o.a.c.c.C.[Tomcat].[localhost].[]      : Initializing Spring Dispatche
2024-04-21T21:30:24.965+02:00  INFO 19512 --- [hopital] [nio-8084-exec-1] o.s.web.servlet.DispatcherServlet      : Initializing Servlet 'dispatcher
```

ID	Nom	Date	Malade	Score		
1	Mohamed	2024-04-21	false	344		
2	Hanane	2024-04-21	false	4321		
3	Imane	2024-04-21	true	349		
4	Mohamed	2024-04-21	false	344		

0 1 2 3 4 5 6 7 8

On doit personnaliser la configuration de sécurité de l'application.

Pour ce faire on va créer une classe SecurityConfig, dont laquelle on travaille avec deux annotations @Configuration et @EnableWebSecurity.

On utilise également l'annotation @Bean pour que notre méthode s'exécute au démarrage.

```
package ma.fsm.hopital.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        return httpSecurity.build();
    }
}
```

L'accès est refusé



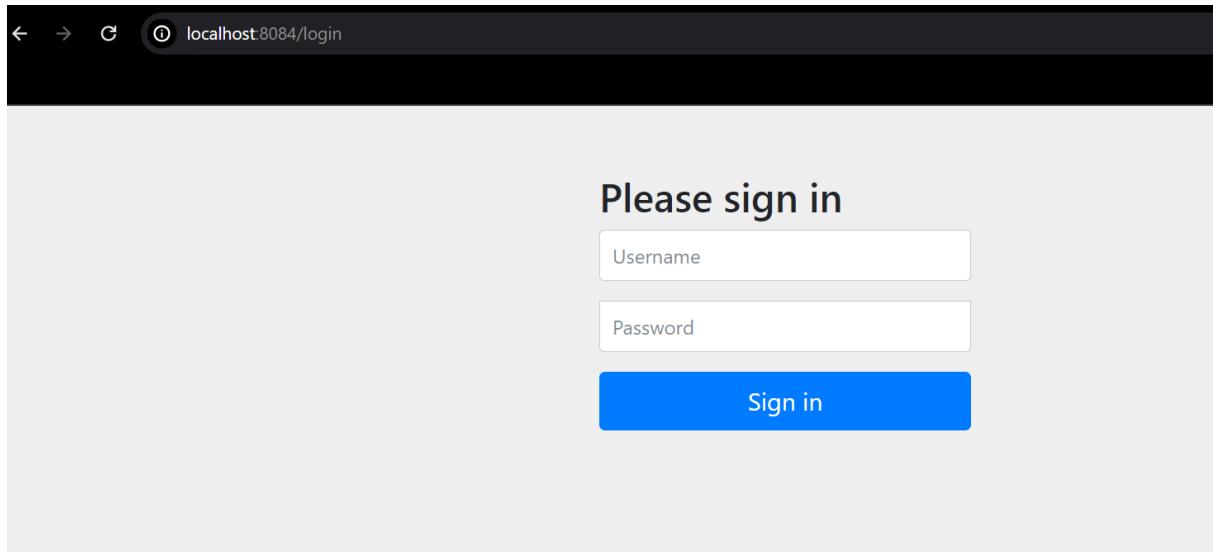
L'accès à localhost a été refusé

Vous n'êtes pas autorisé à consulter cette page.

HTTP ERROR 403

On va demander d'ajouter un formulaire d'authentification en ajoutant formLogin()

Et le formulaire de l'authentification s'affiche



On doit créer des autorisations pour les utilisateurs, pour cela on utilise un objet de type `InMemoryAuthentication`, qui permet de préciser au mémoire les utilisateurs qui ont le droit d'accéder à l'application.

On utilise `{noop}` pour dire à spring security ne pas utiliser un password encoder, il va juste comparer le mot de passe tel qu'il est.

```
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public InMemoryUserDetailsManager inMemoryUserDetailsManager() {
        return new InMemoryUserDetailsManager(
            User.withUsername("user1").password("{noop}1234").roles("USER").build(),
            User.withUsername("user1").password("{noop}1234").roles("USER").build(),
            User.withUsername("admin").password("{noop}1234").roles("USER", "ADMIN").build()
        );
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        return httpSecurity.build();
    }
}
```

L'affichage :

localhost:8084/login

Please sign in

Sign in

localhost:8084/index?continue

Liste Patients					
ID	Nom	Date	Malade	Score	
1	Mohamed	2024-04-21	false	344	
2	Hanane	2024-04-21	false	4321	
3	Imane	2024-04-21	true	349	
4	Mohamed	2024-04-21	false	344	

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Pour travailler avec spring security on utilise un password encoder BCrypt

```
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Autowired
    private PasswordEncoder passwordEncoder;

    @Bean
    public InMemoryUserDetailsManager inMemoryUserDetailsManager() {
        return new InMemoryUserDetailsManager(
            User.withUsername("user1").password(passwordEncoder.encode("1234")).roles("USER").build(),
            User.withUsername("user2").password(passwordEncoder.encode("1234")).roles("USER").build(),
            User.withUsername("admin").password(passwordEncoder.encode("1234")).roles("USER", "ADMIN").build()
        );
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        return httpSecurity.build();
    }
}
```

```

Config.java x HopitalApplication.java x
  ▾ public static void main(String[] args) {
    SpringApplication.run(HopitalApplication.class, args);
  }

  ▾ @Override
  ▾ public void run(String... args) throws Exception {
    patientRepository.save(new Patient( id: null, nom: "Mohamed", new Date(), malade: false, score: 344));
    patientRepository.save(new Patient( id: null, nom: "Hanane", new Date(), malade: false, score: 4321));
    patientRepository.save(new Patient( id: null, nom: "Imane", new Date(), malade: true, score: 349));
  }
  ▾ @Bean
  ▾ PasswordEncoder passwordEncoder(){
    return new BCryptPasswordEncoder();
  }
}

```

On affiche le nom du user connecté au lieu de [User], et on ajoute une opération du logout.

Pour afficher l'utilisateur authentifié, on est besoin d'ajouter une autre dépendance thymeleaf extras

```

<dependency>
  <optional>true</optional>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- https://mvnrepository.com/artifact/org.thymeleaf.extras/thymeleaf-extras-springsecurity6 -->
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>
  <version>3.1.0.M1</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
</dependencies>

<build>

```

On demande maspring security via thymeleaf d'afficher le nom de l'utilisateur qui est authentifié

```

    <a class="nav-link active" href="#">Home</a>
</li>
<li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Patients<span class="visually-hidden dropdown-toggle" data-bs-toggle="dropdown">

```

localhost:8084/login

Please sign in

Sign in

localhost:8084/index?continue

Liste Patients				
ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-21	false	344
2	Hanane	2024-04-21	false	4321
3	Imane	2024-04-21	true	349
4	Mohamed	2024-04-21	false	344

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Pour se déconnecter, on passe par un autre bouton logout

```

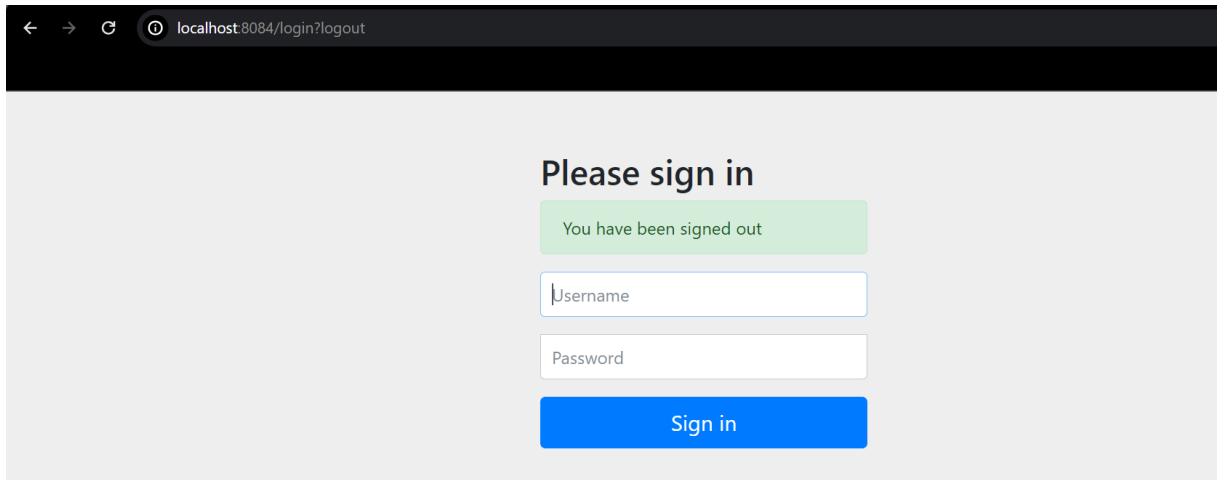
</ul>
<ul class ="navbar-nav">
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" th:text="${#authentication.name}">
        </a>
        <ul class="dropdown-menu">
            <li>
                <form method="post" th:action="@{/logout}">
                    <button class="dropdown-item" type="submit">Logout</button>
                </form>
            </li>
            <li><a class="dropdown-item" href="#">Profile</a></li>
        </ul>
    </li>

</ul>
</div>
</nav>
<section layout:fragment="content1"></section>

```

ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-21	false	344
2	Hanane	2024-04-21	false	4321
3	Imane	2024-04-21	true	349
4	Mohamed	2024-04-21	false	344

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17



On ajoute un méthode Home() pour rediriger vers la page index

```

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

```

Les rôles admin et user sont totalement déférents, c'est pour cela on vérifier les droits d'accès et on va gérer les autorisations et on a changé les chemins dans les templates, avec deux façons :

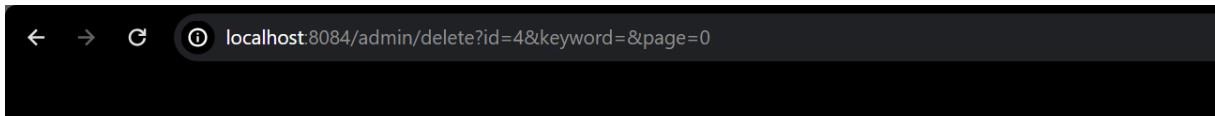
```

public class SecurityConfig {
    @Autowired
    private PasswordEncoder passwordEncoder;

    @Bean
    public InMemoryUserDetailsManager inMemoryUserDetailsManager() {
        return new InMemoryUserDetailsManager(
            User.withUsername("user1").password(passwordEncoder.encode("1234")).roles("USER").build(),
            User.withUsername("user2").password(passwordEncoder.encode("1234")).roles("USER").build(),
            User.withUsername("admin").password(passwordEncoder.encode("1234")).roles("USER", "ADMIN").build()
        );
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
        httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/user/**").hasRole("USER");
        httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/admin/**").hasRole("ADMIN");
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        return httpSecurity.build();
    }
}

```

On est connecté en tant qu'user, et lorsqu'on veut supprimer ou modifier un patient une erreur 403 se génère, car on n'a pas donner à l'utilisateur le droit de faire ses actions



Whitelabel Error Page

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

Tue Apr 23 01:56:41 CEST 2024
There was an unexpected error (type=Forbidden, status=403).
Forbidden

On authentifié en tant qu'admin et on trouve qu'on a le droit de modifier, supprimer, etc.

The screenshot shows an administrative form for patient data entry. The URL in the browser is `localhost:8084/admin/formPatients`. The form fields are:

- Nom: A text input field.
- Date Naissance: A date input field with the placeholder "jj/mm/aaaa".
- Malade: A checkbox labeled "Malade".
- Score: A text input field containing the value "0".
- Save: A blue "Save" button.

Si on se connecte en tant que admin, on peut faire tous les actions.

On règle les droits d'accès, et rien à afficher à l'utilisateur les actions qu'il le peut pas faire.

Alors on va retirer les boutons pour l'utilisateur :

A screenshot of a code editor showing a table row template. The template includes columns for patient ID, name, birth date, status, score, and two buttons for delete and edit operations. The delete button uses a confirmation dialog and a href pointing to /admin/delete. The edit button uses a href pointing to /admin/editPatients.

```
</tr>
<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>
    <td th:if="#{authorization.expression('hasRole(''ADMIN''))}">
        <a onclick="return confirm('Etes vous sure?')" class="btn btn-danger">
            Delete
        </a>
    </td>
    <td th:if="#{authorization.expression('hasRole(''ADMIN''))}">
        <a href="@{/admin/editPatients(id=${p.id}, keyword=${keyword}, page=${currentPage})}">Edit
    </td>

```

A screenshot of a code editor showing a navigation bar template. It includes a home link and a dropdown menu for patients. The dropdown menu has a link to /user/index if the user is an admin. The dropdown menu also includes a link to /user/index for searching.

```
</nav>
<body>
<nav class="navbar navbar-expand-sm bg-dark navbar-dark">
    <div class="container-fluid">
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link active" th:href="@{/user/index}">Home</a>
            </li>
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Patients</a>
                <ul class="dropdown-menu">
                    <li th:if="#{authorization.expression('hasRole(''ADMIN''))}"><a class="dropdown-item" th:href="@{/user/index}">Chercher</a></li>
                </ul>
            </li>
        </ul>
        <ul class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" th:text="#{authentication.name}">
            </a>
            <ul class="dropdown-menu">
                <li>
```

Voilà les boutons sont retirés lorsqu'on se connecte en tant que user

localhost:8084/user/index

Home Patients ▾ user1 ▾

Chercher

Keyword:

ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-21	false	344
2	Hanane	2024-04-21	false	4321
3	Imane	2024-04-21	true	349
4	Mohamed	2024-04-21	false	344

localhost:8084/user/index

Home Patients ▾

Nouveau

Chercher

Keyword:

ID	Nom	Date	Malade	Score	Actions
1	Mohamed	2024-04-21	false	344	
2	Hanane	2024-04-21	false	4321	
3	Imane	2024-04-21	true	349	
4	Mohamed	2024-04-21	false	344	

Pour qu'on affiche un message à l'utilisateur comme quoi vous n'avez pas le droit de faire cette fonctionnalité, on va ajouter une configuration :

```
public class SecurityConfig {
    @Autowired
    private PasswordEncoder passwordEncoder;

    @Bean
    public InMemoryUserDetailsManager inMemoryUserDetailsManager() {
        return new InMemoryUserDetailsManager(
            User.withUsername("user1").password(passwordEncoder.encode("1234")).roles("USER"),
            User.withUsername("user2").password(passwordEncoder.encode("1234")).roles("USER"),
            User.withUsername("admin").password(passwordEncoder.encode("1234")).roles("ADMIN")
        );
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin();
        httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/user/**").hasRole("USER");
        httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/admin/**").hasRole("ADMIN");
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        httpSecurity.exceptionHandling().accessDeniedPage("/notAuthorized");
        return httpSecurity.build();
    }
}
```

Et on ajoute un autre contrôleur SecurityController

```
package ma.fsm.hopital.web;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class SecurityController {

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

Et aussi une vue notAuthorized.html :

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="template1">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
        <link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
        <link rel="stylesheet" href="/webjars/bootstrap-icons/1.10.3/font/bootstrap-icons.css">
    </head>
    <body>
        <div layout:fragment="content1">
            <div class="alert alert-danger m-3">
                <h2>Not Authorized</h2>
            </div>
        </div>
    </body>
</html>

```



Page user1 :

The screenshot shows a browser window with the URL `localhost:8084/user/index?continue`. The page displays a table titled "Liste Patients" with columns: ID, Nom, Date, Malade, and Score. The table contains four rows of data. Below the table is a pagination bar with numbered buttons from 0 to 23. The browser interface includes a navigation bar with back, forward, and search icons, and a user profile section on the right.

ID	Nom	Date	Malade	Score
1	Mohamed	2024-04-21	false	344
2	Hanane	2024-04-21	false	4321
3	Imane	2024-04-21	true	349
4	Mohamed	2024-04-21	false	344

Page admin :

ID	Nom	Date	Malade	Score		
1	Mohamed	2024-04-21	false	344		
2	Hanane	2024-04-21	false	4321		
3	Imane	2024-04-21	true	349		
4	Mohamed	2024-04-21	false	344		

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

On ajoute encore une vue login.html

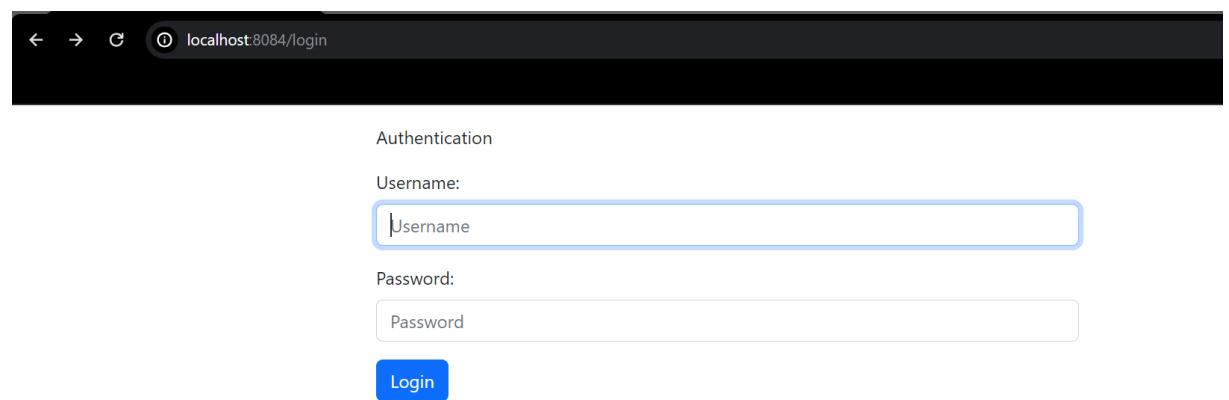
```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Authentification</title>
    <link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
    <link rel="stylesheet" href="/webjars/bootstrap-icons/1.10.3/font/bootstrap-icons.css">
</head>
<body>
    <div class="row mt-3">
        <div class="col-md-6 offset-3">
            <div class="card-header">Authentication</div>
            <div class="card-body">
                <form method="post" th:action="@{/login}">
                    <div class="mb-3 mt-3">
                        <label for="username" class="form-label">Username: </label>
                        <input type="text" class="form-control" id="username" placeholder="Username" name="username" th:value="#{user.username}">
                    </div>
                    <div class="mb-3 mt-3">
                        <label for="password" class="form-label">Password: </label>
                        <input type="password" class="form-control" id="password" placeholder="Password" name="password" th:value="#{user.password}">
                    </div>
                    <button type="submit" class="btn btn-primary">Login</button>
                </form>
            </div>
        </div>
    </div>
</body>

```

```
        return new InMemoryUserDetailsManager(
            User.withUsername("user1").password(passwordEncoder.encode("1234")).roles("USER")
            User.withUsername("user2").password(passwordEncoder.encode("1234")).roles("USER")
            User.withUsername("admin").password(passwordEncoder.encode("1234")).roles("USER")
        );
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin().loginPage("/Login").permitAll();
        httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/webjars/**", "/h2-console/**").permitAll();
        httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/user/**").hasRole("USER");
        httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/admin/**").hasRole("ADMIN");
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        httpSecurity.exceptionHandling().accessDeniedPage("/notAuthorized");
        return httpSecurity.build();
    }
}
```



On veut ajouter une case à cocher Remember me

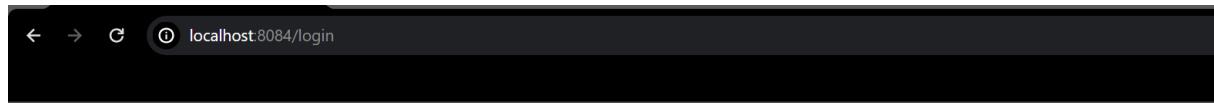
A screenshot of a code editor displaying a template for a login form. The code uses Thymeleaf syntax, indicated by the `th:action` attribute. The form includes fields for username and password, a remember-me checkbox, and a login button. The code is well-structured with appropriate class and id attributes for styling.

```
<div class="card-body">
    <form method="post" th:action="@{/login}">
        <div class="mb-3 mt-3">
            <label for="username" class="form-label">Username: </label>
            <input type="text" class="form-control" id="username" placeholder="Username" name="username" th:value=" ${username}">
        </div>
        <div class="mb-3 mt-3">
            <label for="password" class="form-label">Password: </label>
            <input type="password" class="form-control" id="password" placeholder="Password" name="password" th:value=" ${password}">
        </div>
        <div class="form-check mb-3">
            <label class="form-check-label">
                <input class="form-check-input" type="checkbox" name="remember-me"> Remember me
            </label>
        </div>
        <button type="submit" class="btn btn-primary">Login</button>
    </form>
</div>
</div>
```

A screenshot of a code editor displaying Java configuration code. It defines two beans: `InMemoryUserDetailsManager` and `SecurityFilterChain`. The `InMemoryUserDetailsManager` bean creates an in-memory user details manager with three users: `user1`, `user2`, and `admin`, each with the role `USER` and raw password `1234`. The `SecurityFilterChain` bean configures a security filter chain using `HttpSecurity`. It enables form-based login for the `/login` endpoint, remembers the user, and applies various authorization rules for different paths like `/webjars/**`, `/h2-console/**`, `/user/**`, and `/admin/**`. It also handles access denied pages.

```
@Bean
public InMemoryUserDetailsManager inMemoryUserDetailsManager() {
    return new InMemoryUserDetailsManager(
        User.withUsername("user1").password(passwordEncoder.encode("1234")).roles("USER"),
        User.withUsername("user2").password(passwordEncoder.encode("1234")).roles("USER"),
        User.withUsername("admin").password(passwordEncoder.encode("1234")).roles("USER")
    );
}

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.formLogin().loginPage("/login").permitAll();
    httpSecurity.rememberMe();
    httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/webjars/**", "/h2-console/**").permittedBy();
    httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/user/**").hasRole("USER");
    httpSecurity.authorizeHttpRequests().requestMatchers(...patterns: "/admin/**").hasRole("ADMIN");
    httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
    httpSecurity.exceptionHandling().accessDeniedPage("/notAuthorized");
    return httpSecurity.build();
}
```



Authentication

Username:

admin

Password:

Remember me

Login

ID	Nom	Date	Malade	Score		
1	Mohamed	2024-04-21	false	344		
2	Hanane	2024-04-21	false	4321		
3	Imane	2024-04-21	true	349		
4	Mohamed	2024-04-21	false	344		

Keyword:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28

Maintenant on passe pour stocker les données dans la base de données au lieu dans une mémoire.

On protège les données avec l'annotation `@EnableMethodSecurity`, et `@PreAuthorize`.

```

import org.springframework.context.annotation.Configuration;
import org.springframework.core.userdetails.User;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
public class SecurityConfig {
    @Autowired
    private PasswordEncoder passwordEncoder;

    @Bean
    public InMemoryUserDetailsManager inMemoryUserDetailsManager() {
        return new InMemoryUserDetailsManager(
            User.withUsername("user1").password(passwordEncoder.encode("1234")).roles("ROLE_USER"),
            User.withUsername("user2").password(passwordEncoder.encode("1234")).roles("ROLE_USER"),
            User.withUsername("admin").password(passwordEncoder.encode("1234")).roles("ROLE_ADMIN")
        );
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity
            .authorizeHttpRequests(auth -> auth
                .anyRequest().authenticated()
            )
            .formLogin(configurer -> configurer
                .loginPage("/login")
                .usernameParameter("username")
                .passwordParameter("password")
                .successHandler(successHandler -> successHandler.defaultSuccessUrl("/"))
            )
            .logout(configurer -> configurer.logoutUrl("/logout"))
        );
        return httpSecurity.build();
    }
}

```

```

        return "patients";
    }

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

    @GetMapping("/patients")
    @ResponseBody
    public List<Patient> listPatients() { return patientRepository.findAll(); }

    @GetMapping("/admin/formPatients")
    public String formPatient(Model model) {
        model.addAttribute("patient", new Patient());
        return "formPatients";
    }

    @PostMapping(path = "/admin/save")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public String save(Model model, @Valid Patient patient, BindingResult bindingResult,
                      @RequestParam(defaultValue = "0") int page,
                      @RequestParam(defaultValue = "") String keyword) {
        if(bindingResult.hasErrors()) return "formPatients";
        patientRepository.save(patient);
        return "redirect:/user/index?page=" + page + "&keyword=" + keyword;
    }
}

```