Département d'Informatique

Filiere : IAAD

A ,U :2023-2024

Université Moulay Ismail faculté des Sciences Meknès

Département d'Informatique

TP N 3 : Spring Security

Module : Systèmes distribués

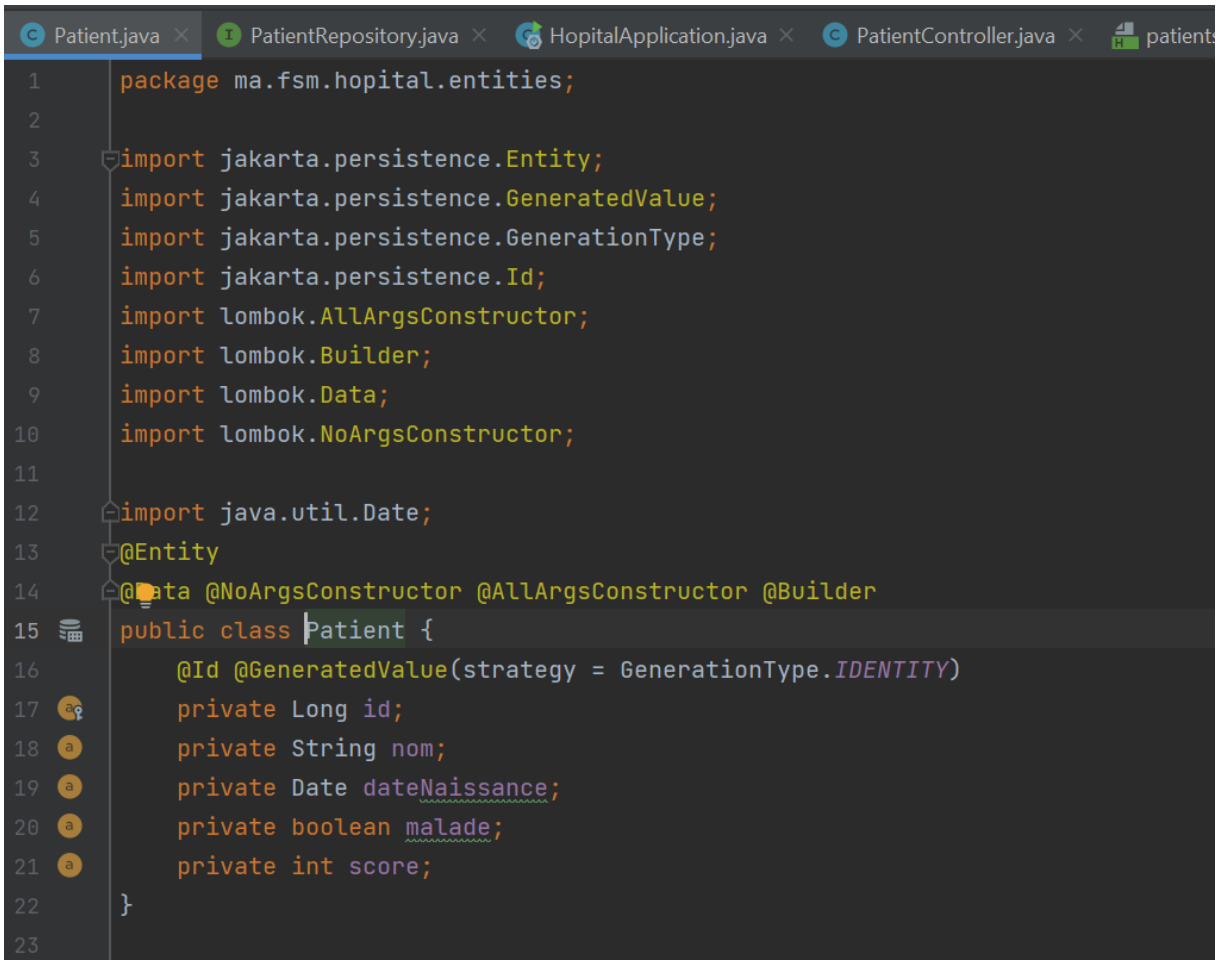
Réalisé par :

- ***Illa Meryeme***

Partie 1 :

Créer une application web JEE basée sur les dépendances Spring MVC, Thylemeaf 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



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

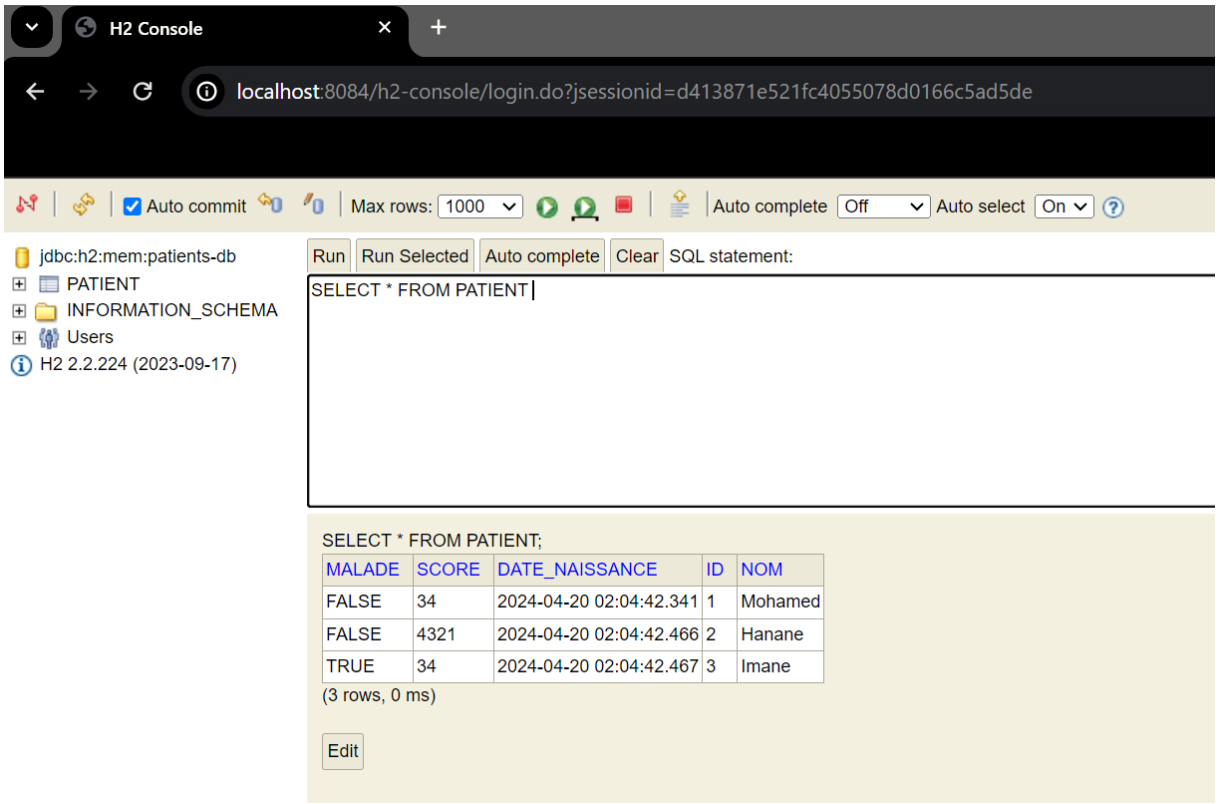
```java
private PatientRepository patientRepository;

public static void main(String[] args) {

    SpringApplication.run(HopitalApplication.class, args);
}

@Override
public void run(String... args) throws Exception {
    // En utilisant constructeur sans paramétre
    Patient patient= new Patient();
    patient.setId(null);
    patient.setNom("Mohamed");
    patient.setDateNaissance(new Date());
    patient.setMalade(false);
    patient.setScore(23);
    // En utilisant constructeur avec paramétre
    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
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));
```

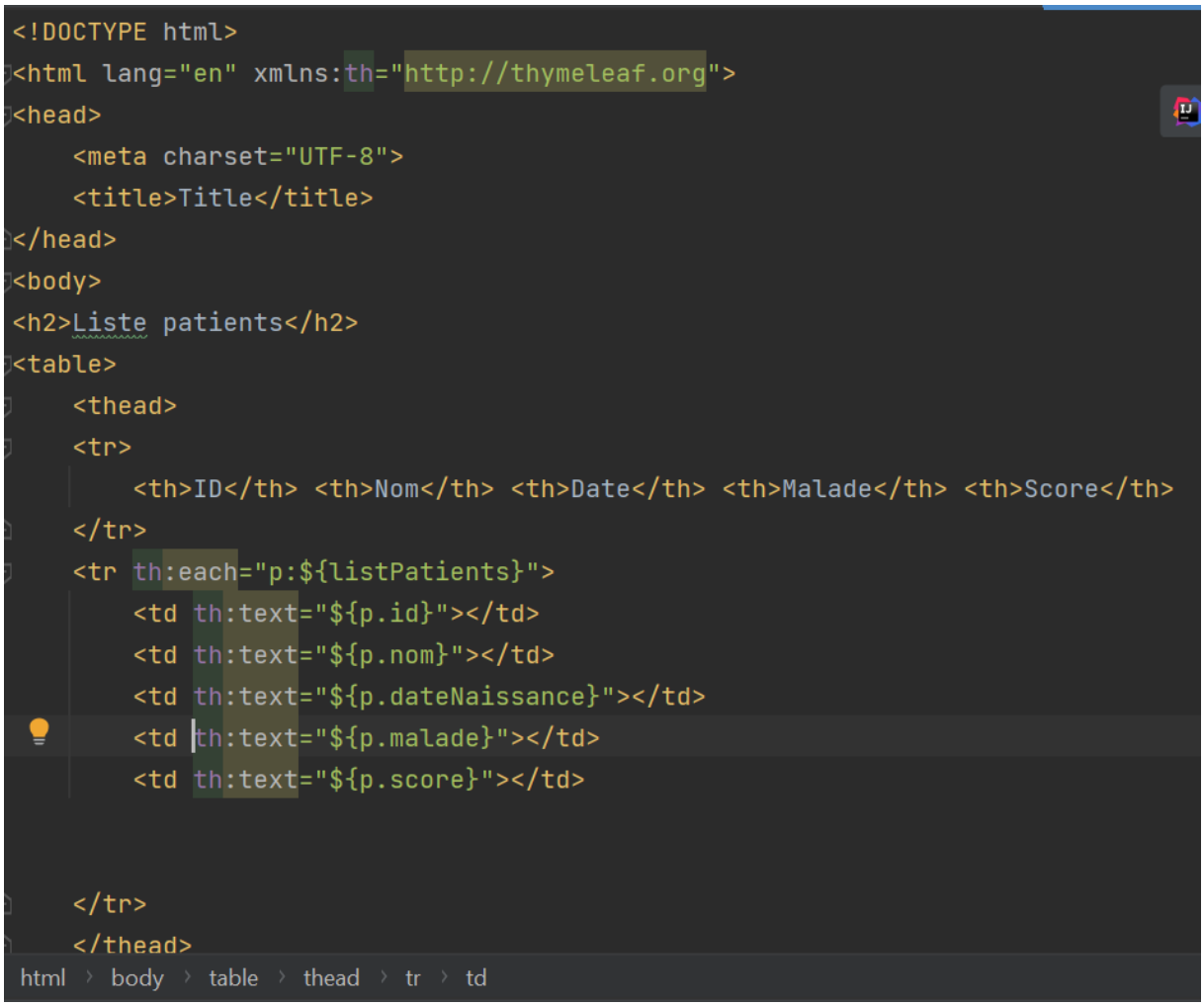


On va créer la classe PatientContrôleur et pour qu'il soit contrôleur on utilise la notation @Controller qui a besoin de faire appel a l'interface PatientRepository et injecter le constructeur par la notation @AllArgsConstructor

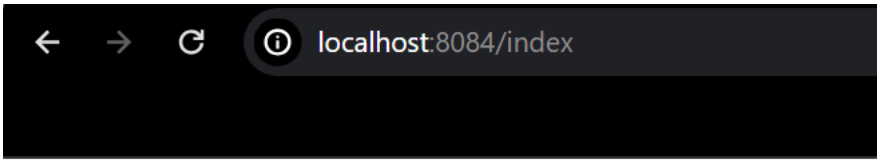
On crée une méthode index qui retourne une vue patients et on faire rappel à cette méthode par la notation @GetMapping. Dans Template, on va créer le file html « patient.html »

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



```java
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, Deployement cocher Build project Automaticaly.

On affiche la table bootstrap avec le card

```html
<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>
```

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

Liste Patients

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

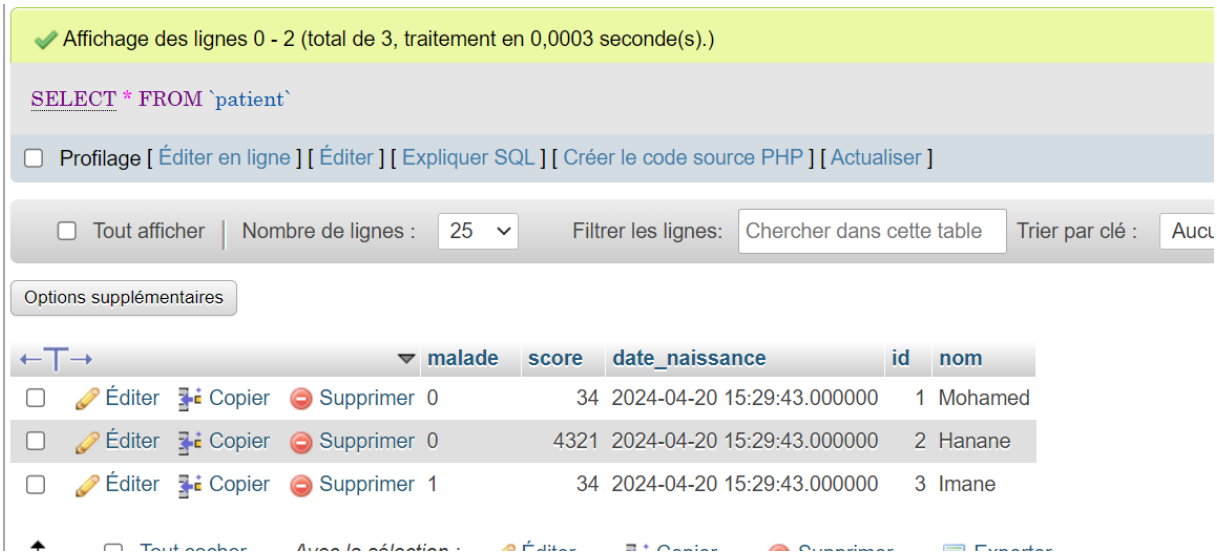
On veut basculer de la base de données h2 à MySQL pour faire la pagination.

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

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

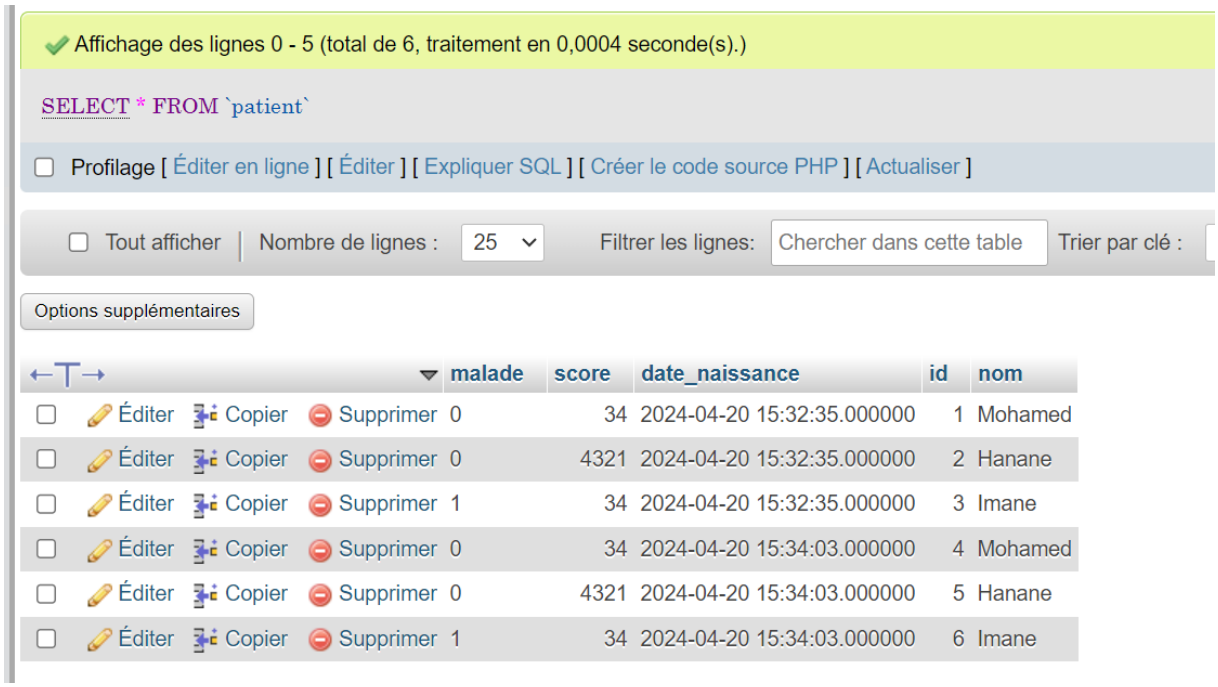
```
spring.application.name=hopital
server.port=8084
#spring.datasource.url=jdbc:h2:mem:patients-db
#spring.h2.console.enabled=true
spring.datasource.url=jdbc:mysql://localhost:3306/hopital?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=create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

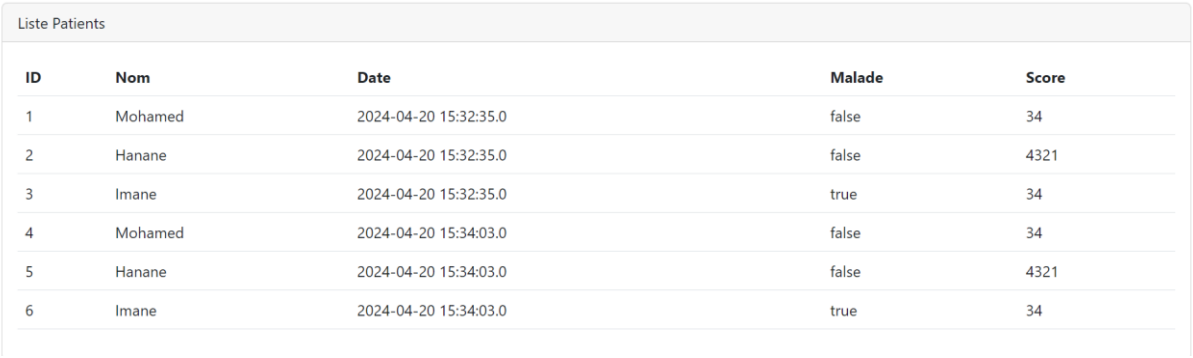
L'affichage :



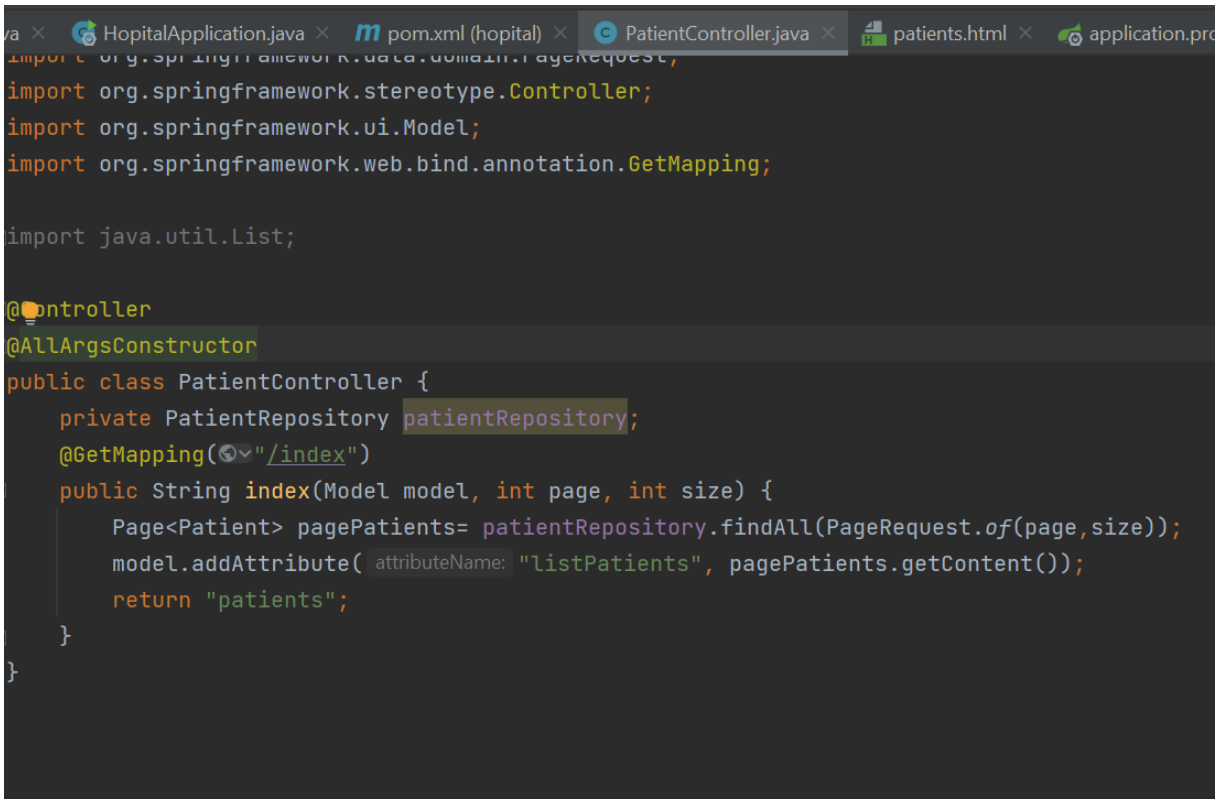
On redémarre l'application, elle nous rajouter les données car on n'a pas écrasé la base de données, et c'est fait exprès car on a besoin de plusieurs données pour qu'on puisse faire la pagination

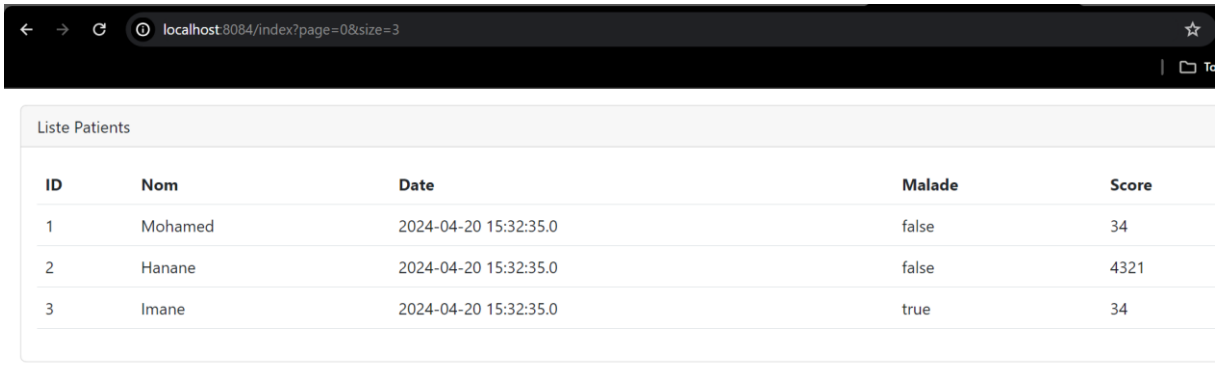
```
spring.application.name=hopital
server.port=8084
#spring.datasource.url=jdbc:h2:mem:patients-db
#spring.h2.console.enabled=true
spring.datasource.url=jdbc:mysql://localhost:3306/hopital?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update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





On fait la pagination :





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

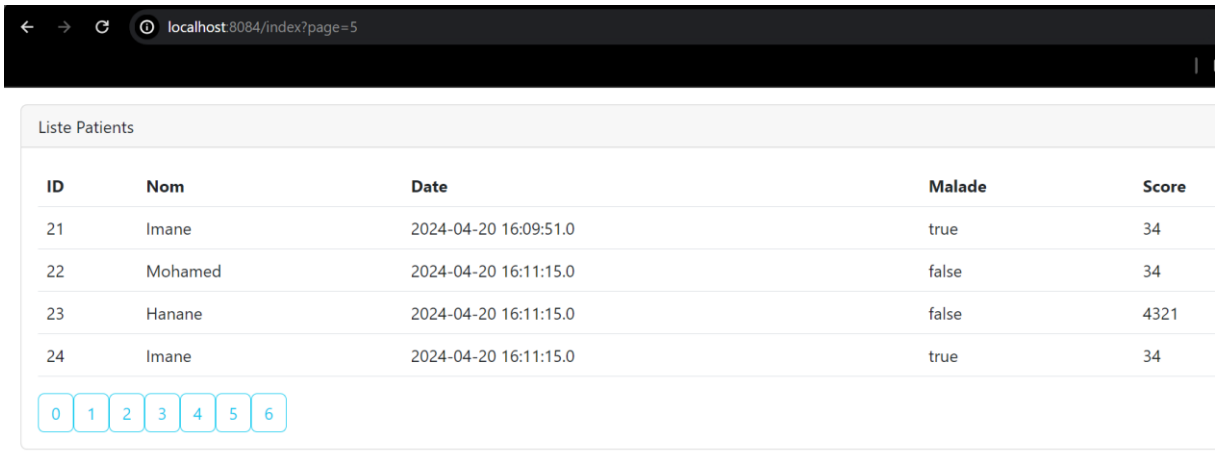
Et pour faire la pagination on a stocké le nombre de pages dans un model

```java
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()]);
        return "patients";
    }
}
```

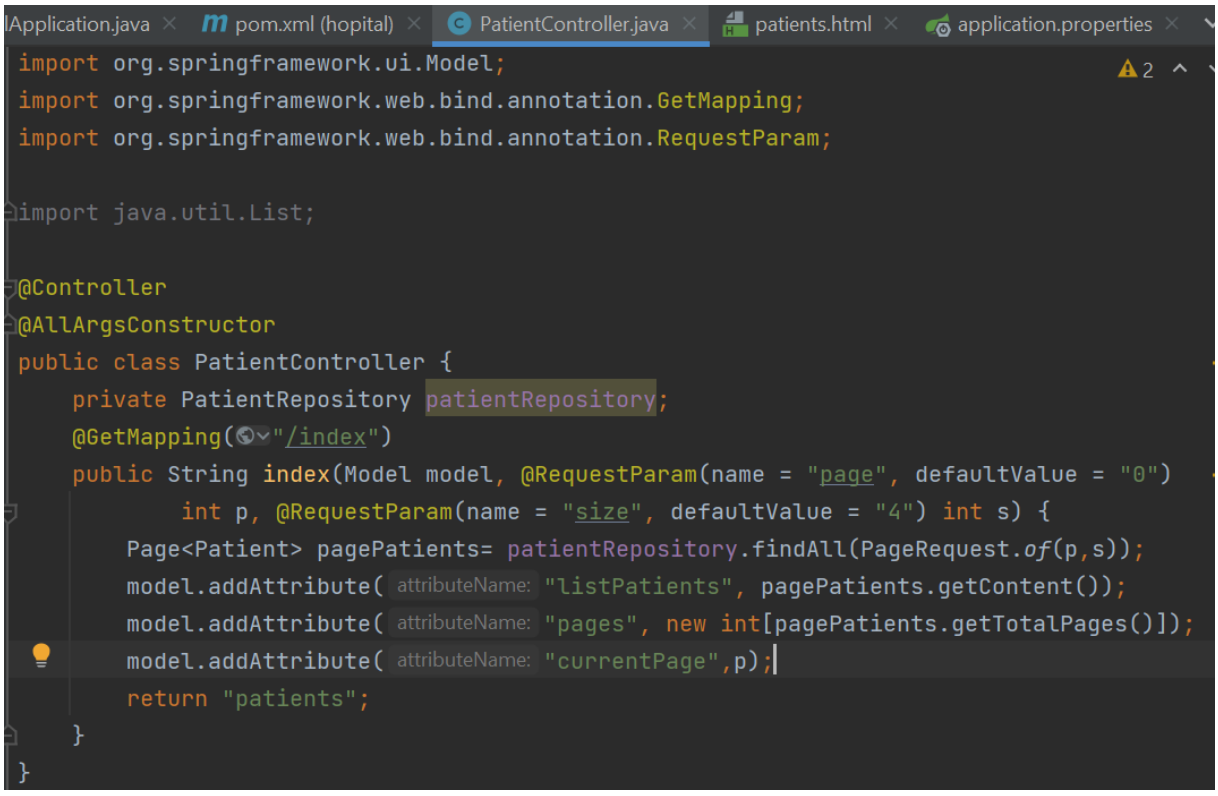
On a ajouté des boutons pour se déplacer d'une page vers l'autre dans le fichier patients.html sous forme d'une liste ul

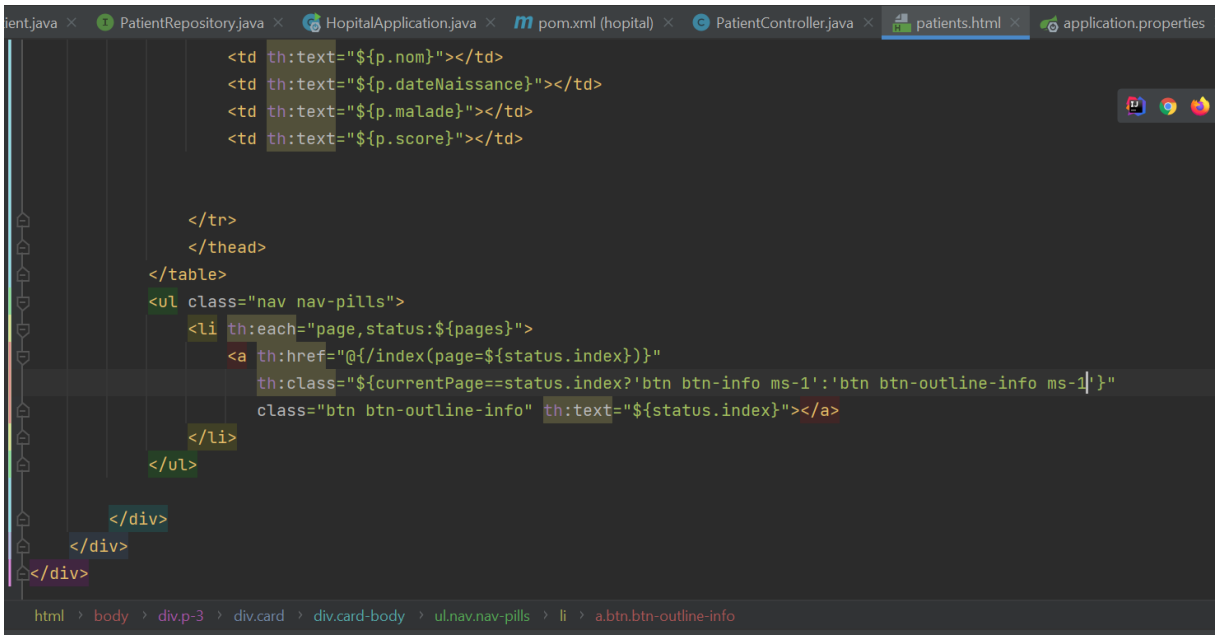
```html
<tr>
    <th>ID</th> <th>Nom</th> <th>Date</th> <th>Malade</th> <th>Score</th>
</tr>
<tr th:each="p:${listPatients}">
    <td th:text="${p.id}"></td>
    <td th:text="${p.nom}"></td>
    <td th:text="${p.dateNaissance}"></td>
    <td th:text="${p.malade}"></td>
    <td th:text="${p.score}"></td>


</tr>
</thead>
</table>
<ul class="nav nav-pills">
    <li th:each="page,status:${pages}">
        <a th:href="@{/index(page=${status.index})}" class="btn btn-outline-info" th:text="${status.index}"></a>
    </li>
</ul>

</div>
</div>
</div>
```

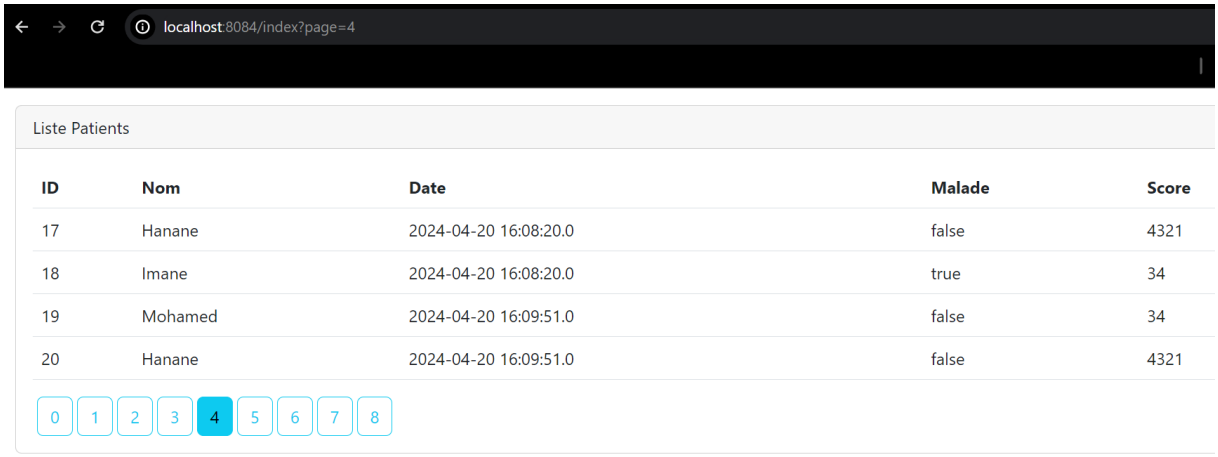
Dans l'affichage, on bascule d'une page vers l'autre



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

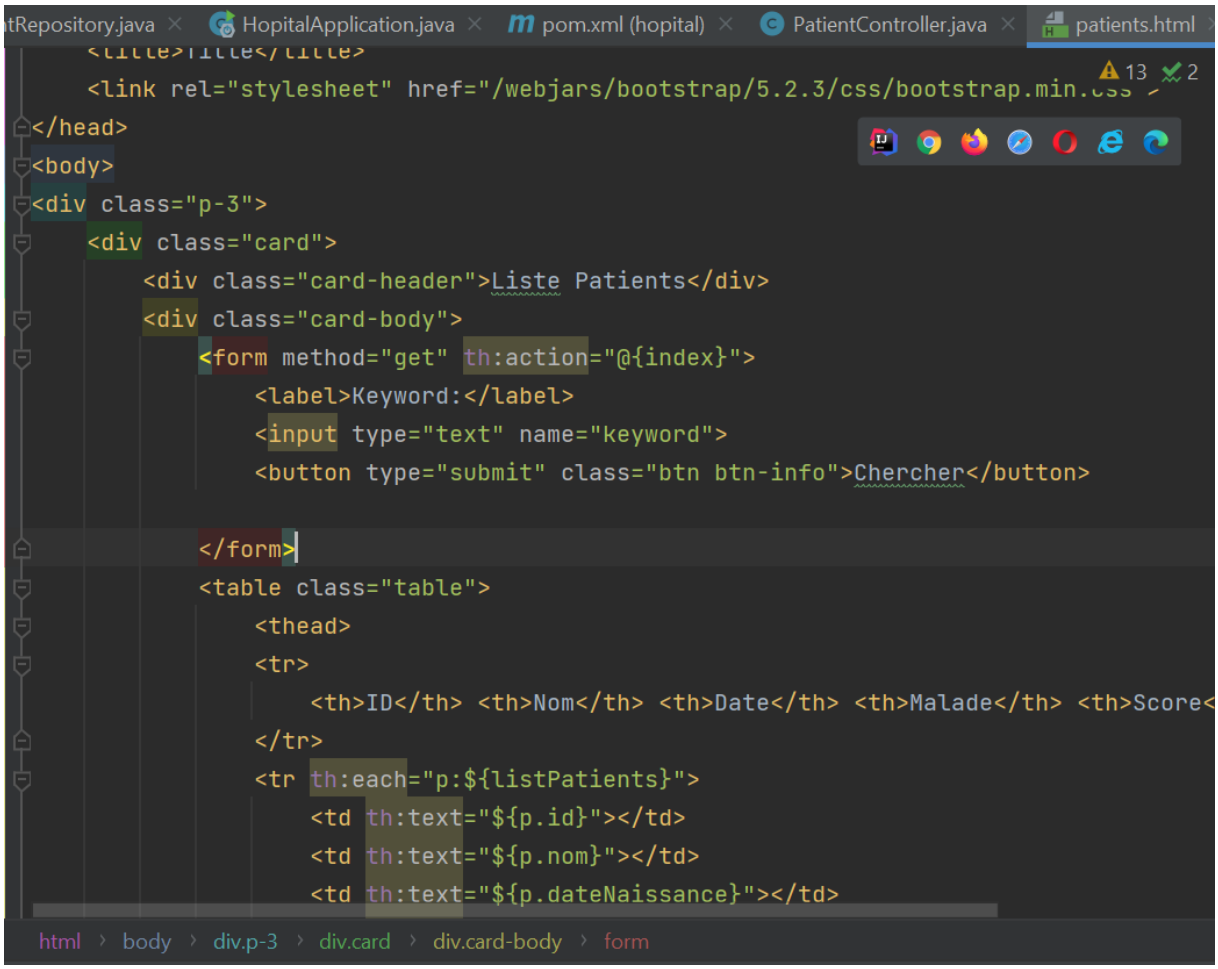


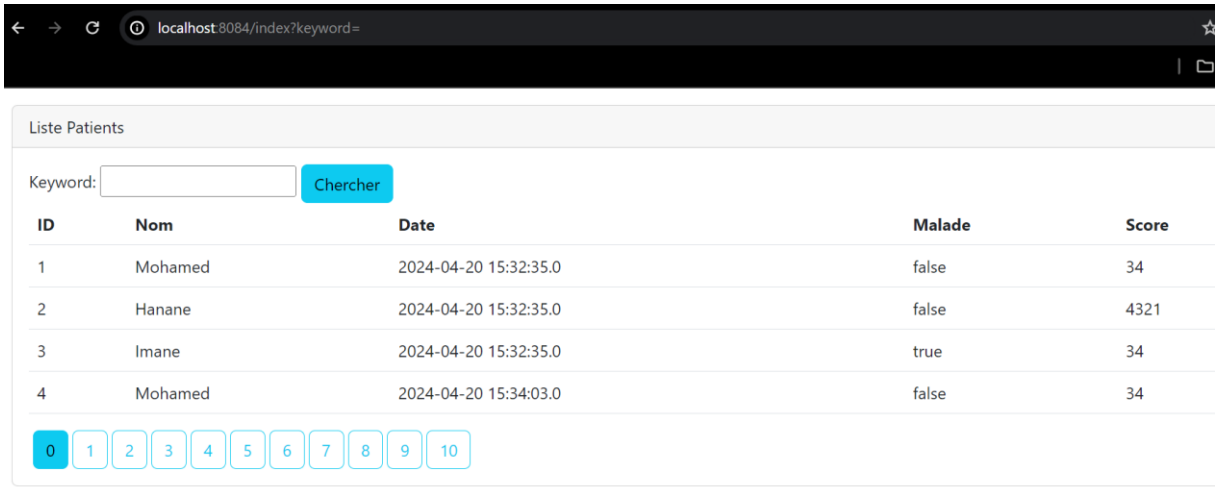




On a effectué la pagination donc on va ajouter une partie de la recherche des patients

On va créer un formulaire

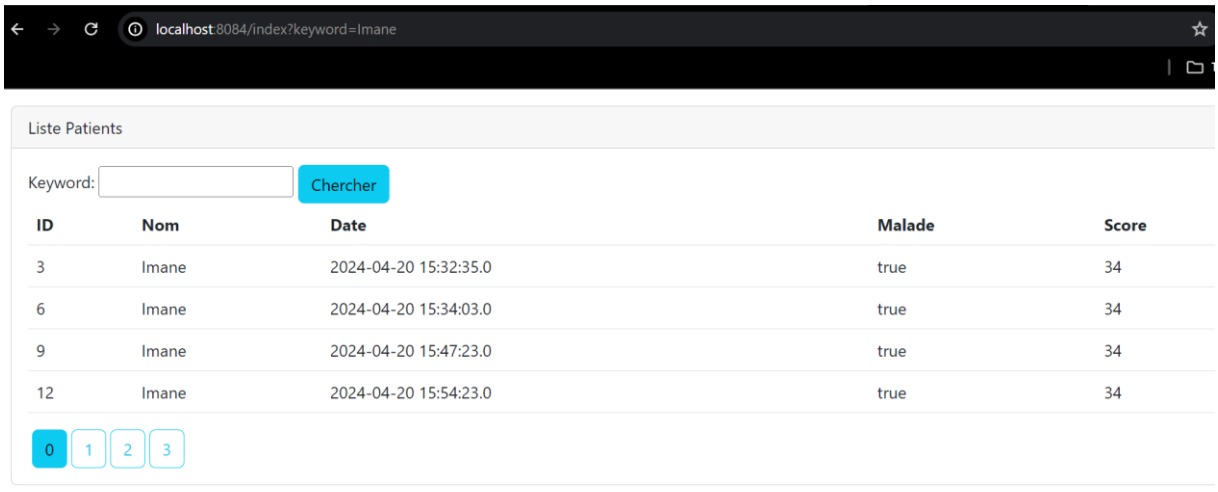




Pour faire la recherche on ajoute une méthode dans l'interface PatientRepository et on utilise le paramètre pageable pour faire la pagination

```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);


}
```

```java
@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";
    }
}
```

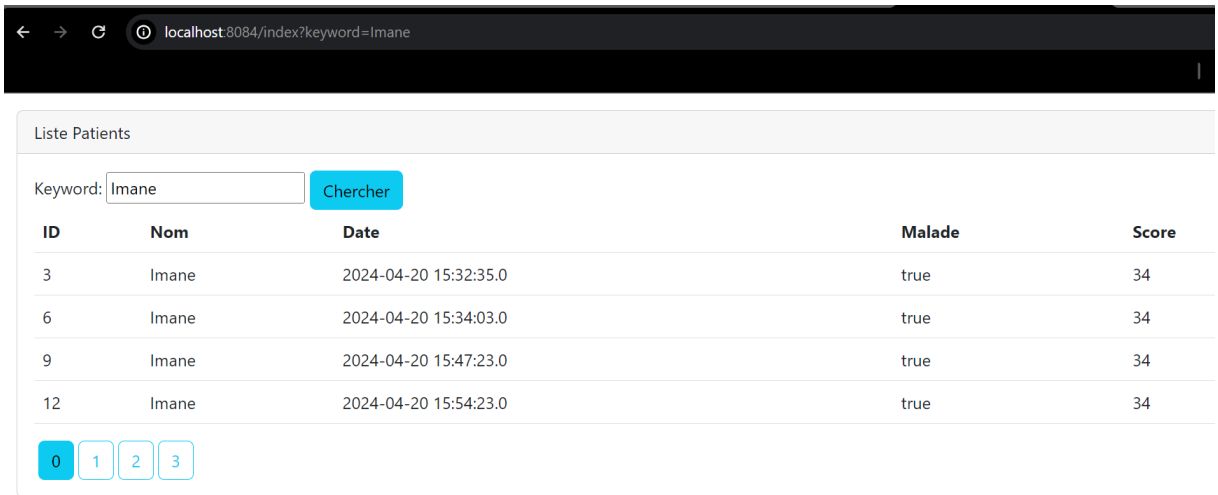


On veut que la valeur rechercher reste dans la zone de texte, alors ce qu'on fait et de stocker dans le model le keyword, car c'est lui qu'on va transmis dans la zone de texte

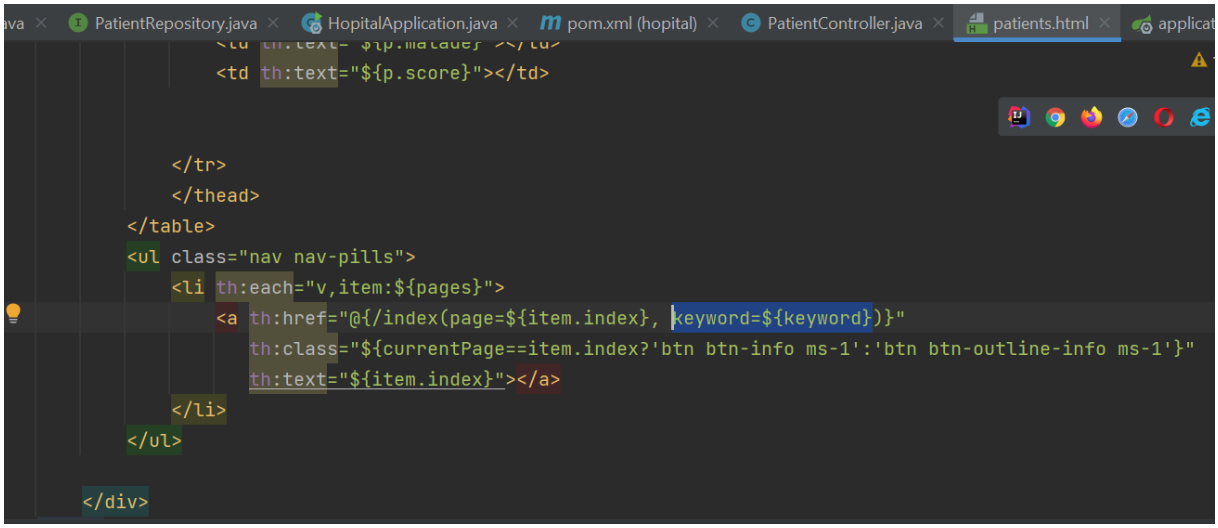
```
<div class="p-3">
    <div class="card">
        <div class="card-header">Liste Patients</div>
        <div class="card-body">
            <form method="get" th:action="@{index}">
                <label>Keyword:</label>
                <input type="text" name="keyword" th:value="${keyword}">
   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

            </form>
            <table class="table">
                <thead>
                <tr>
```

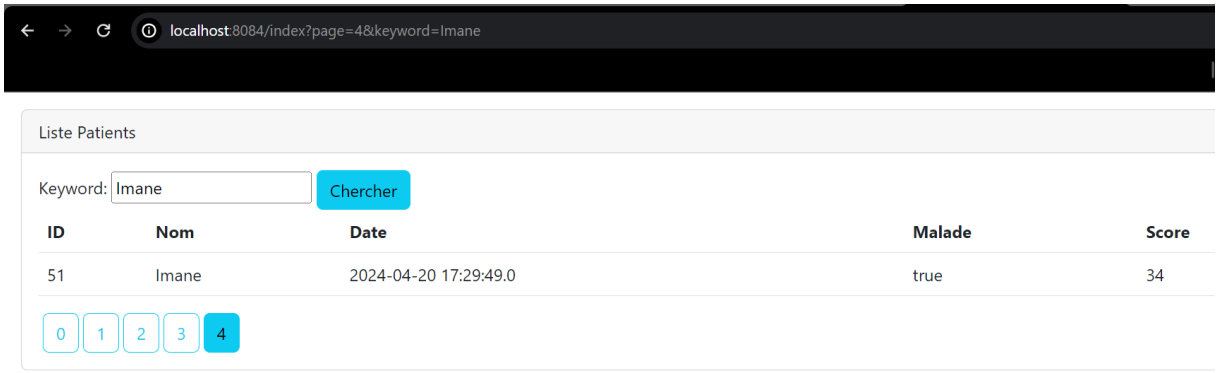
va | PatientRepository.java | HopitalApplication.java | pom.xml (hopital) | PatientController.java | patients.html

```
@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);
        model.addAttribute( attributeName: "keyword", kw);
        return "patients";
    }
}
```

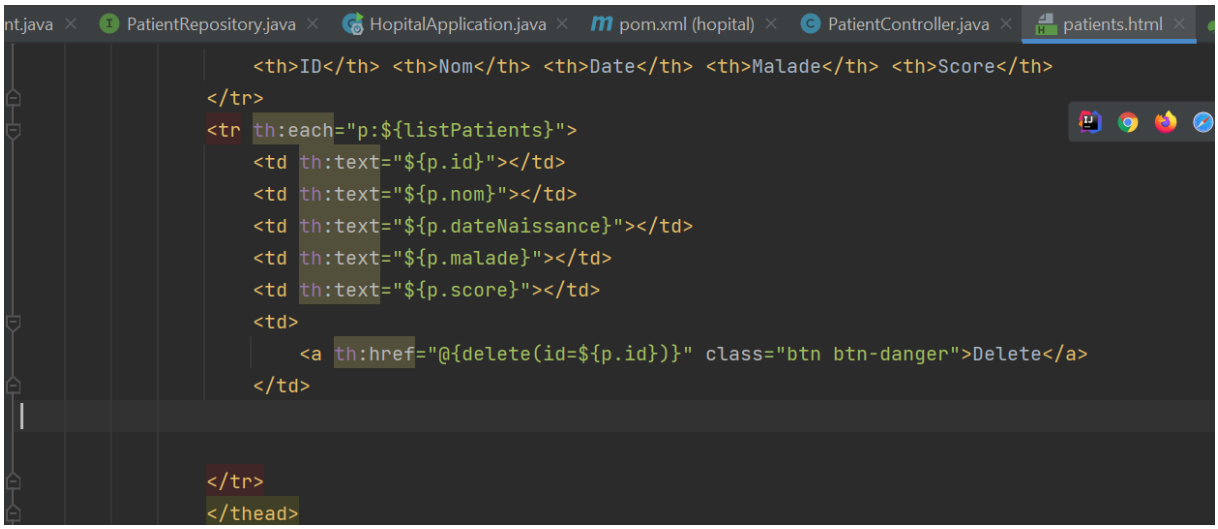


Lorsqu'on appuyer sur un bouton d'une autre page il revient d'afficher tous les patients car le keyword par défaut c'est vide et non pas celles recherchés, si on ajoute le keyword dans le lien ça va marcher



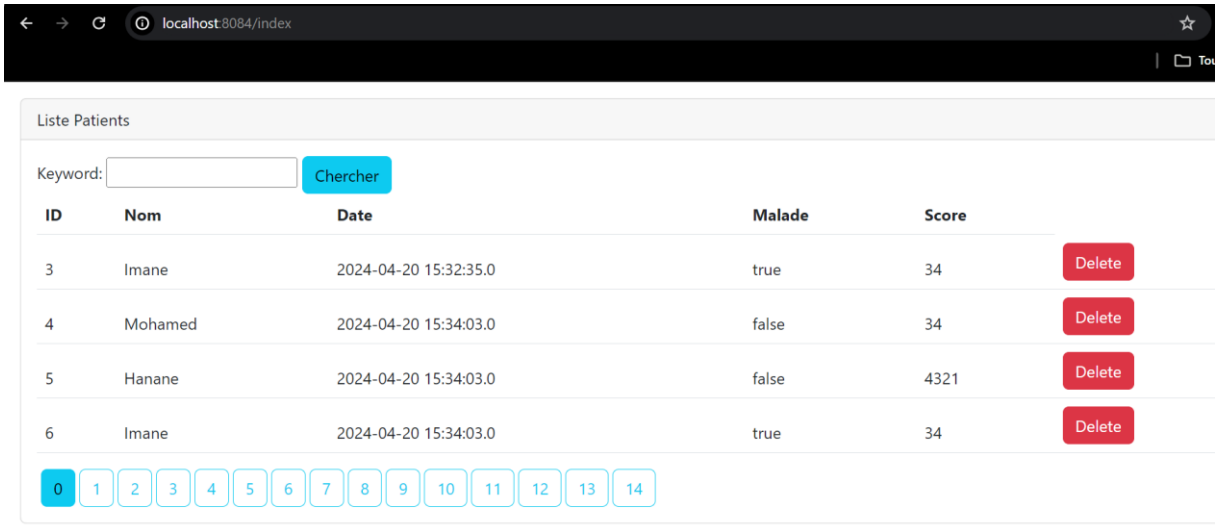


On veut supprimer un patient, tout d'abord on va créer le bouton delete, puis on ajoute la fonction delete dans le contrôleur



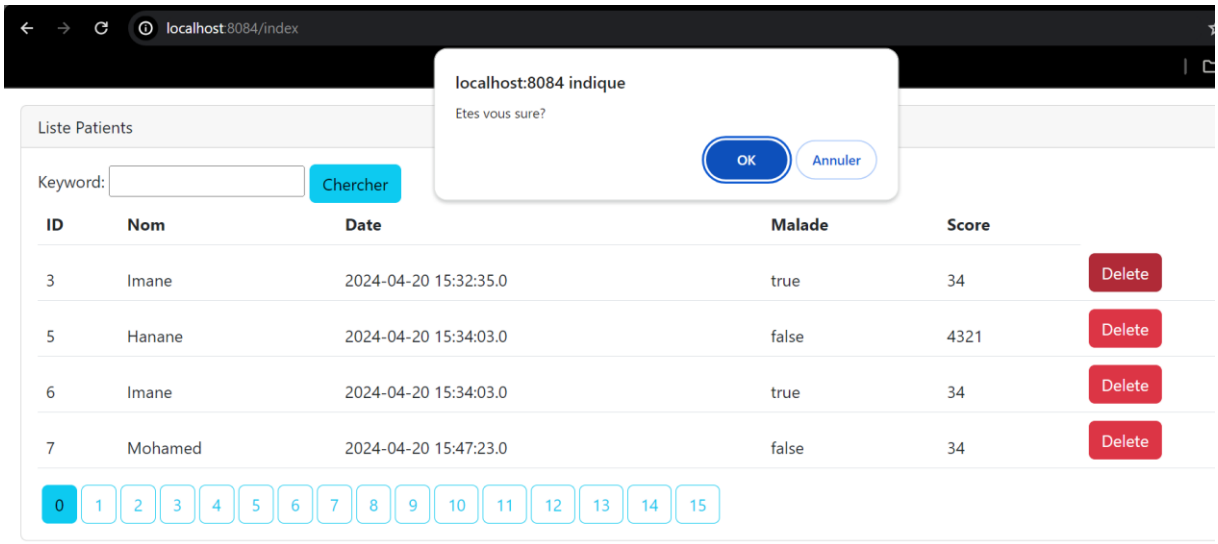
```java
        Page<Patient> pagePatients= patientRepository.findByNomContains(kw,PageRequest.of(p,s));
        model.addAttribute( attributeName: "listPatients", pagePatients.getContent());
        model.addAttribute( attributeName: "pages", new int[pagePatients.getTotalPages()]);
        model.addAttribute( 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

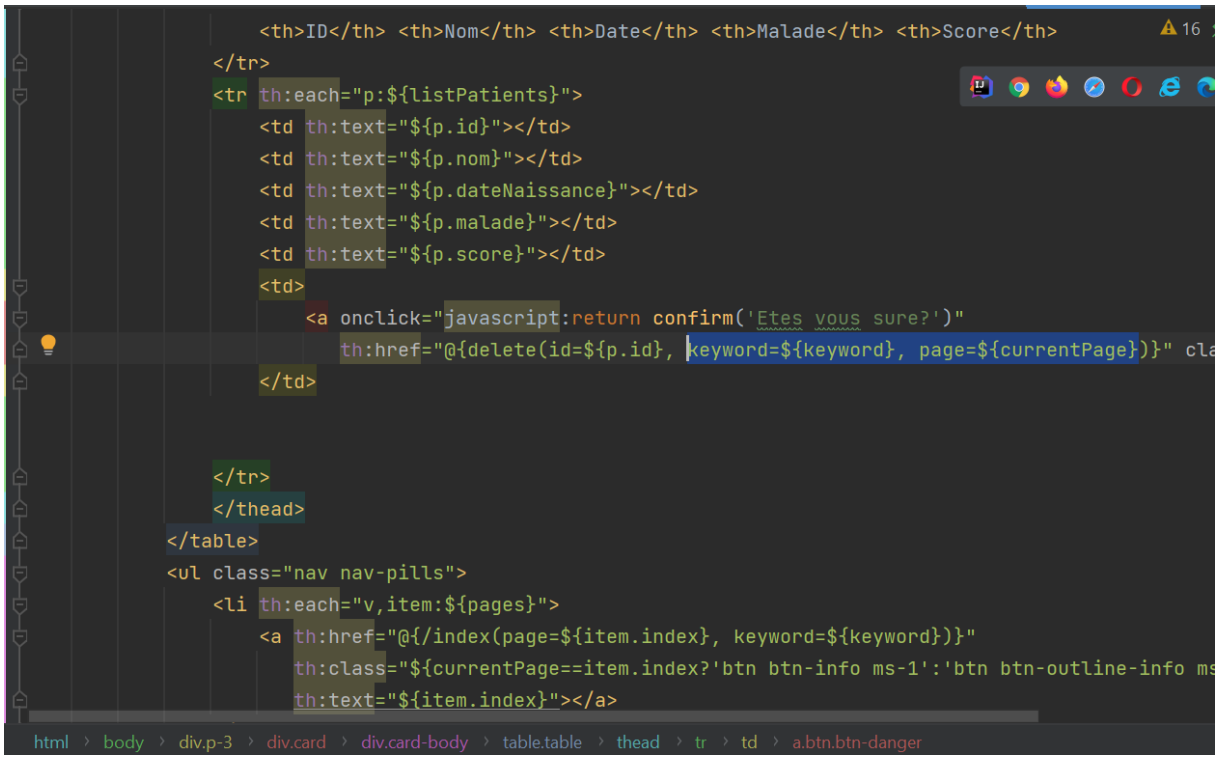


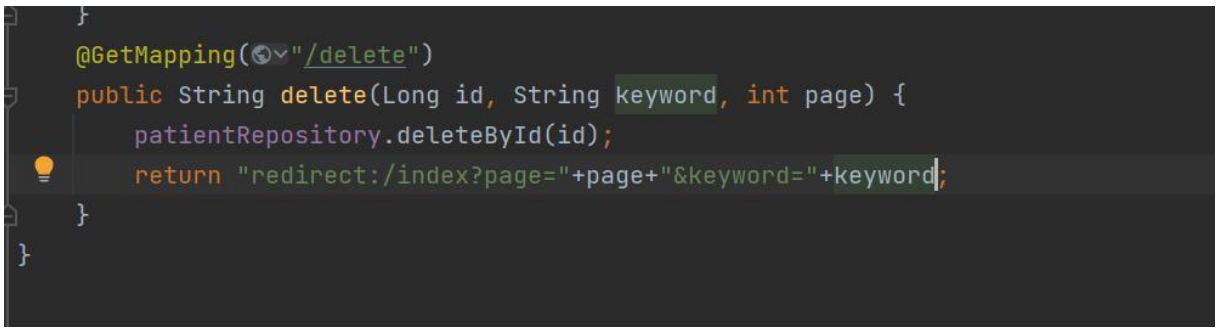
On affiche un message de confirmation qui permet de confirmer la suppression

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

    </tr>
    </thead>
```

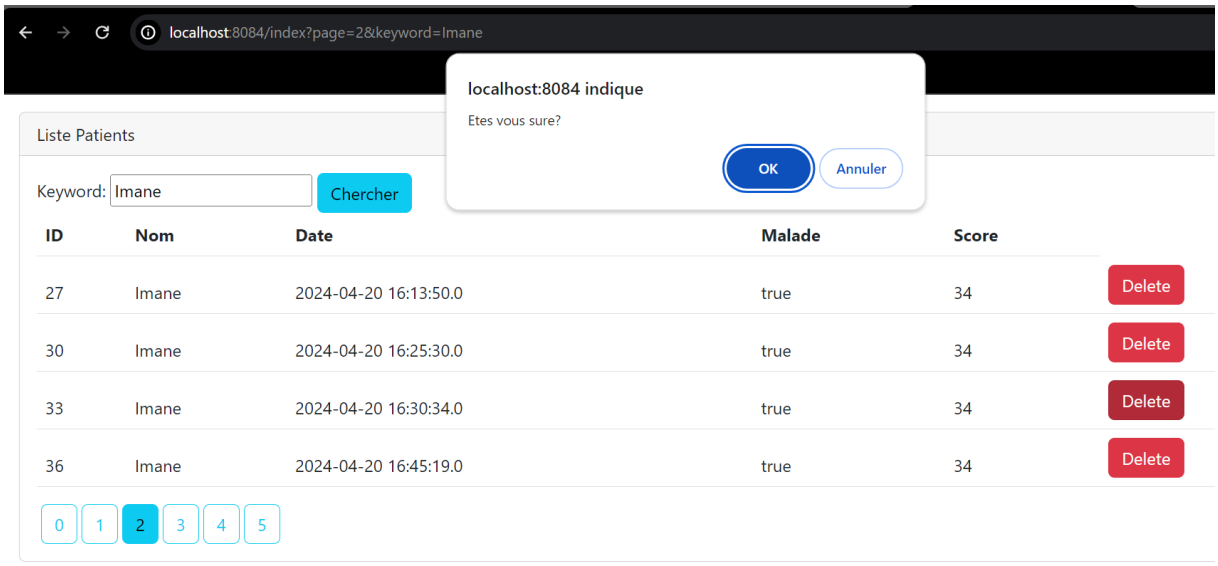


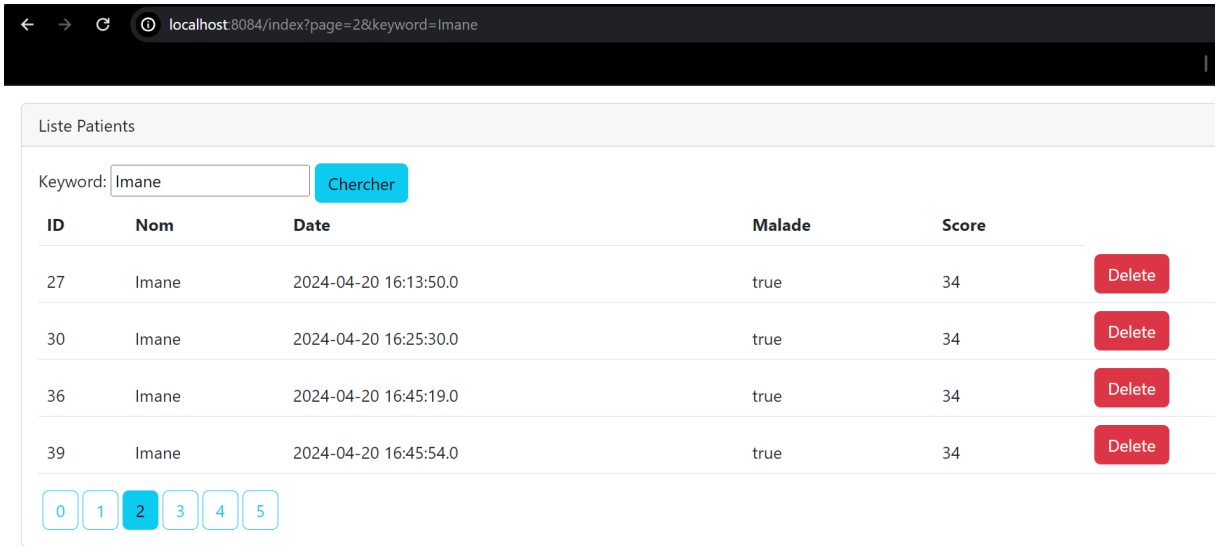
Lorsqu'on supprime un patient, on perd le mot clé et on a résoudre le problème de la même manière qu'on a fait précédemment





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



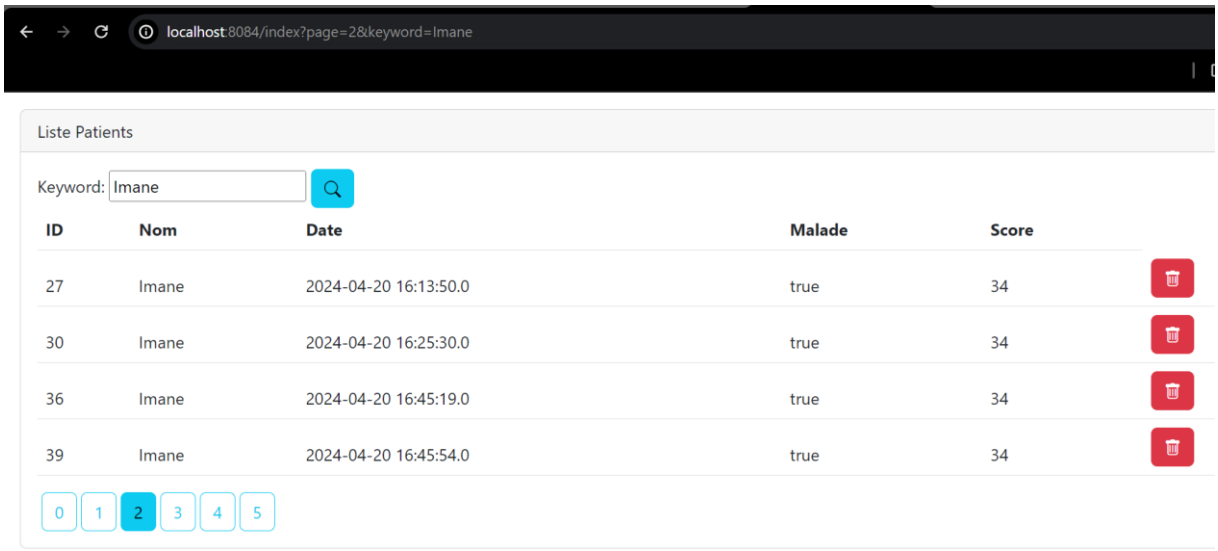


On améliore notre application par des icones, tout d'abord on ajoute la dépendance bootstrap-icons

```
<!-- 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>
/button>

m>
e 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>
    <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>
```



Partie 2 :

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

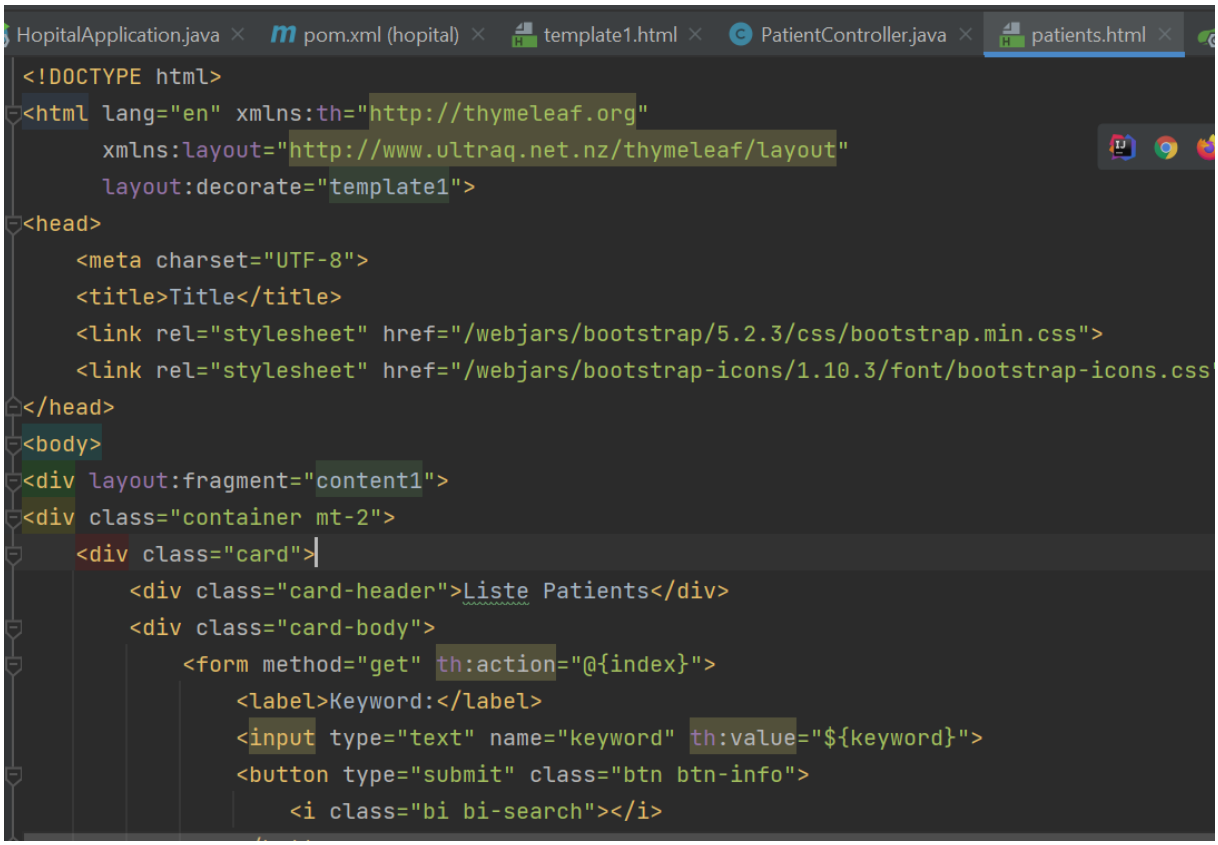
On ajoute les dépendances de thymleaf layout dialect pour avoir un header fixe dans toutes les pages sans faire copier-coller à chaque fois

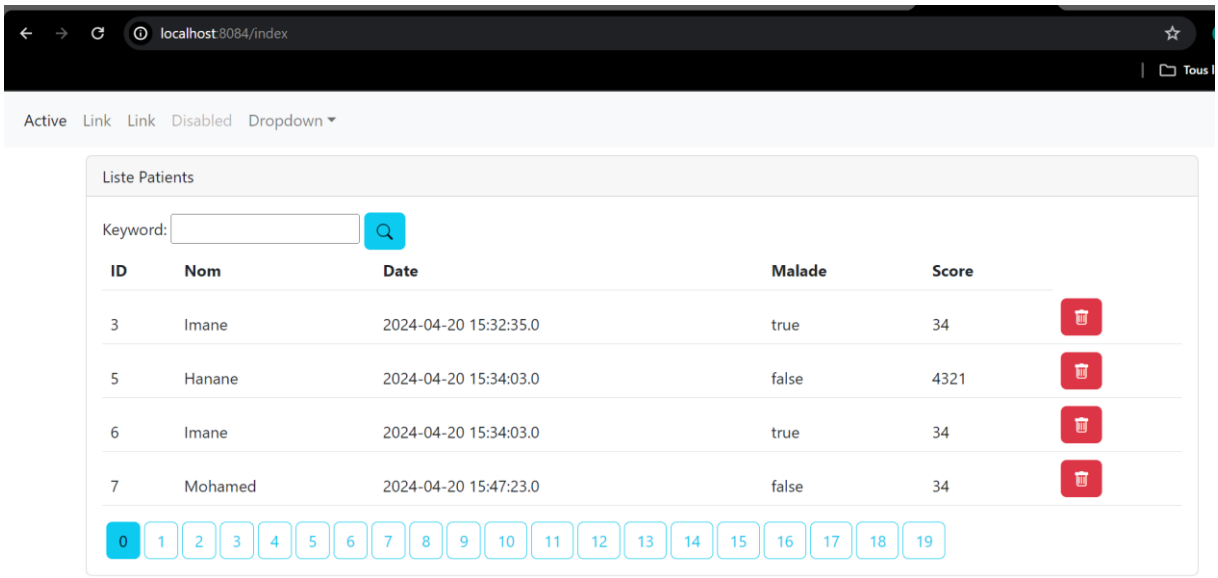
```
        </dependency>
        <!-- https://mvnrepository.com/artifact/nz.net.ultraq.thymeleaf/thymeleaf-layout-dialect -->
        <dependency>
            <groupId>nz.net.ultraq.thymeleaf</groupId>
            <artifactId>thymeleaf-layout-dialect</artifactId>
            <version>3.1.0</version>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

HopitalApplication.java | pom.xml (hopital) | template1.html | PatientController.java | patients.html | application.pro

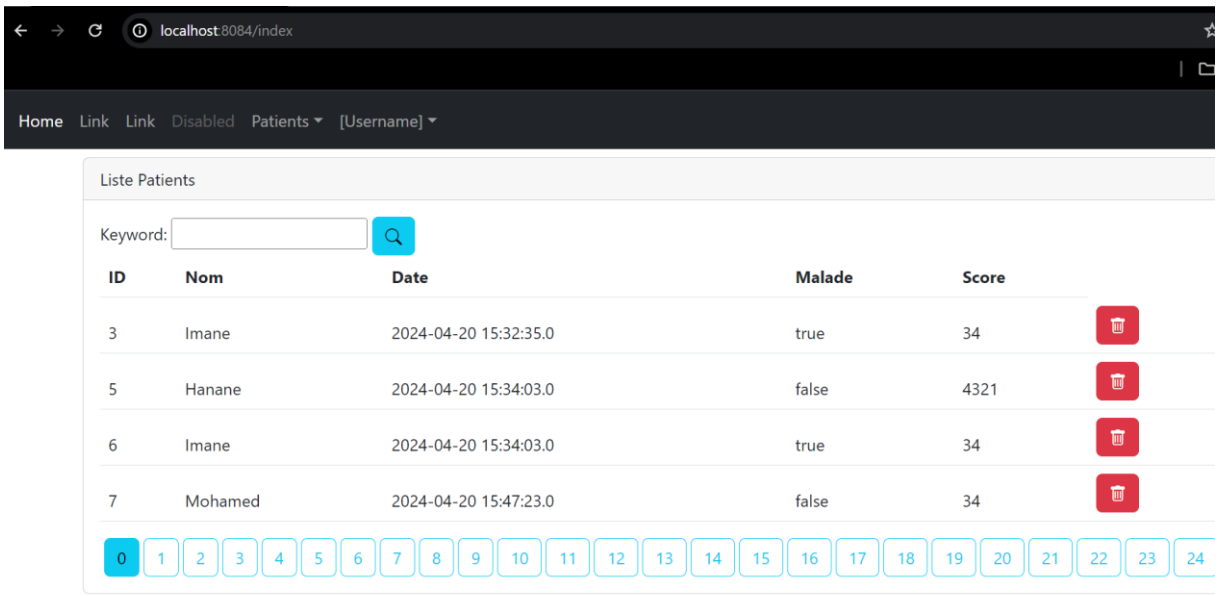
```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://thymeleaf.org"
              xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" type="text/css" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
    <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">
```

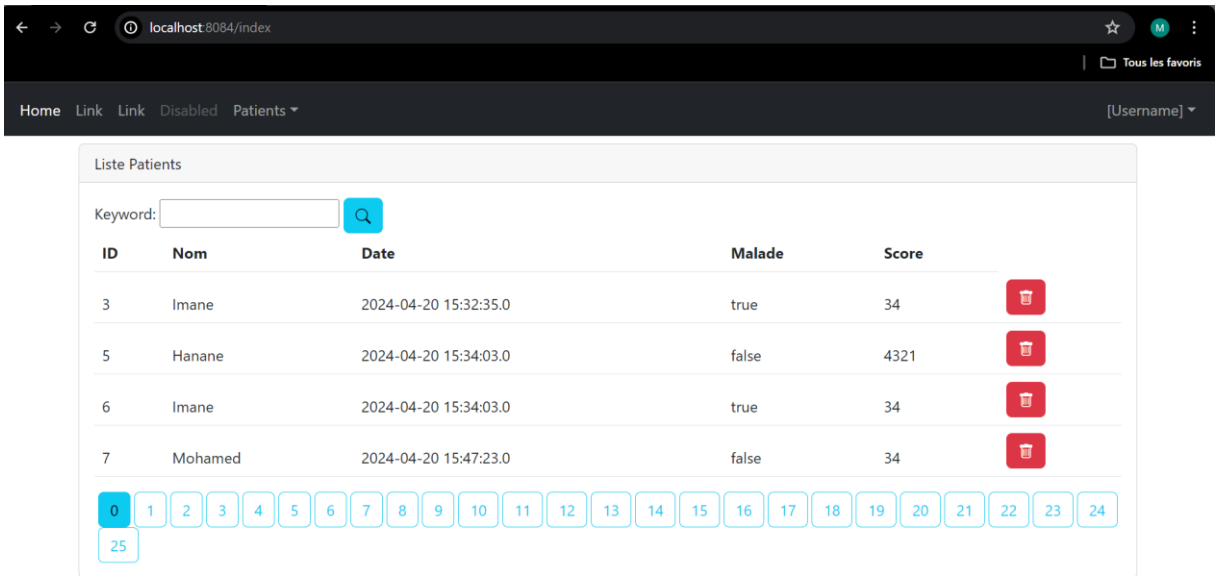




On va changer l'arrière-plan du header et les éléments pour qu'on puisse faire ajouter un nouveau patient, chercher un patient, et logout

```
    </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>
```



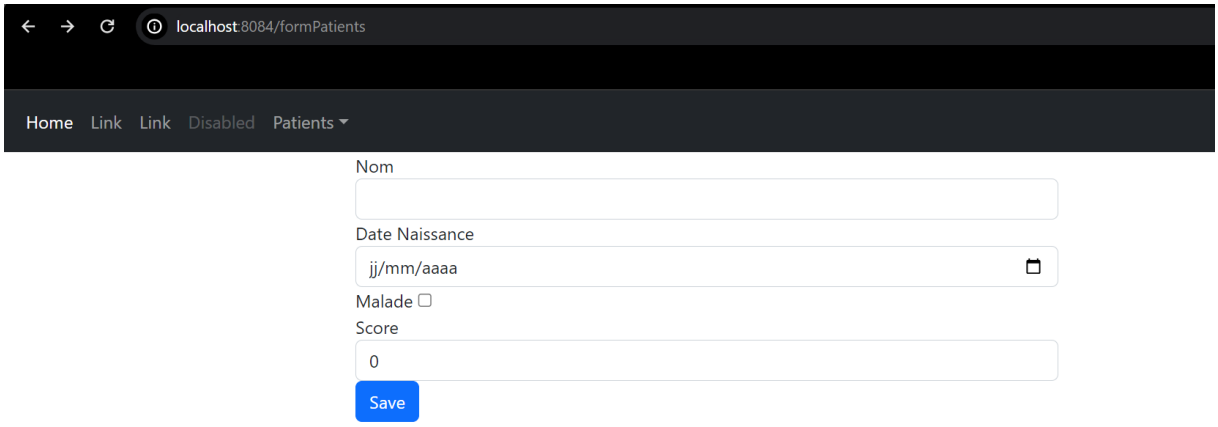


On ajoute une méthode formPatients qui permet d'ajouter un patient dans la classe contrôleur puis on va créer un fichier HTML que l'on appelle formPatients.html, pour générer la vue du formulaire d'ajout

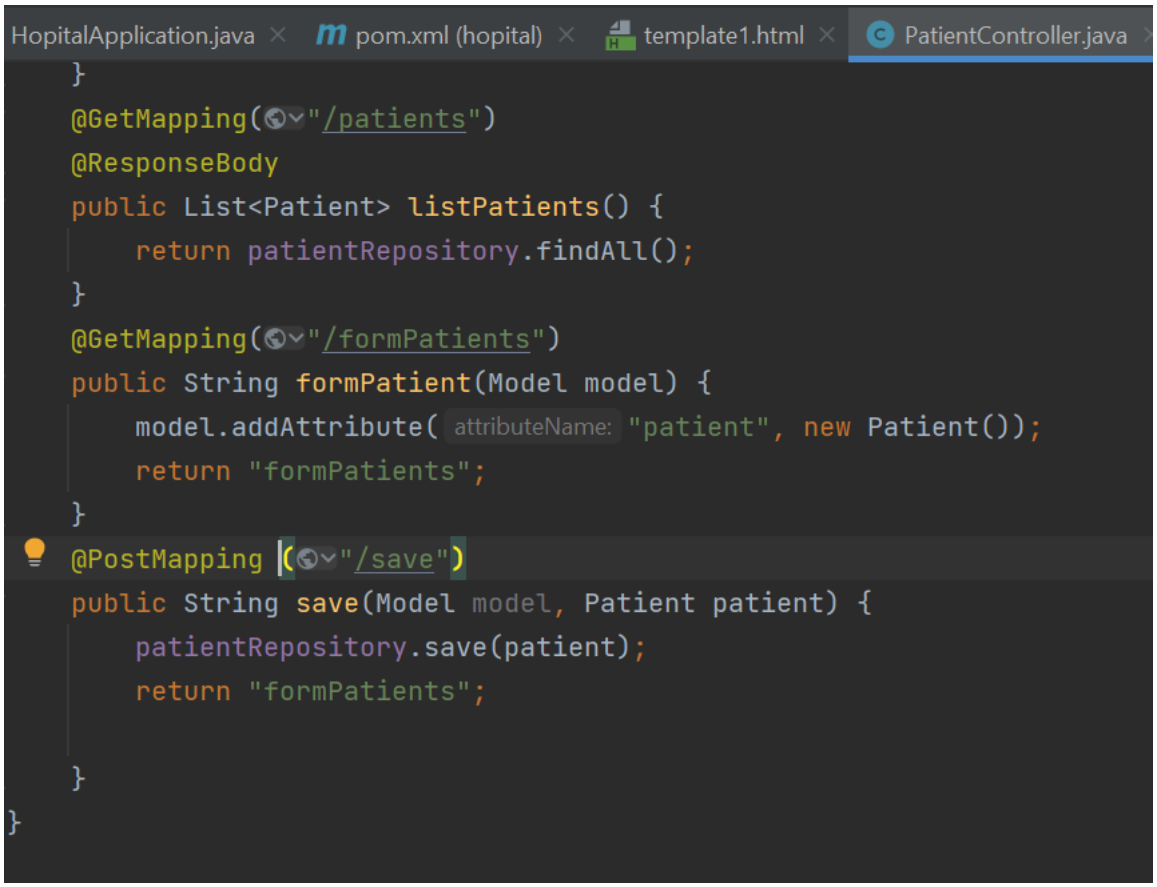
```
        model.addAttribute( attributeName: "listPatients", pagePatients.getContent());
        model.addAttribute( attributeName: "pages", new int[pagePatients.getTotalPages()]);
        model.addAttribute( attributeName: "currentPage",p);
        model.addAttribute( attributeName: "keyword", kw);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

```
<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="${patient.nom}">
        <span th:errors="${patient.nom}"></span>
    </div>
    <div>
        <label>Date Naissance</label>
        <input class="form-control" type="date" name="nom" th:value="${patient.dateNaissance}">
        <span th:errors="${patient.dateNaissance}"></span>
    </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}">
```

html › body › div › div.col-md-6.offset-3 › form › div › input



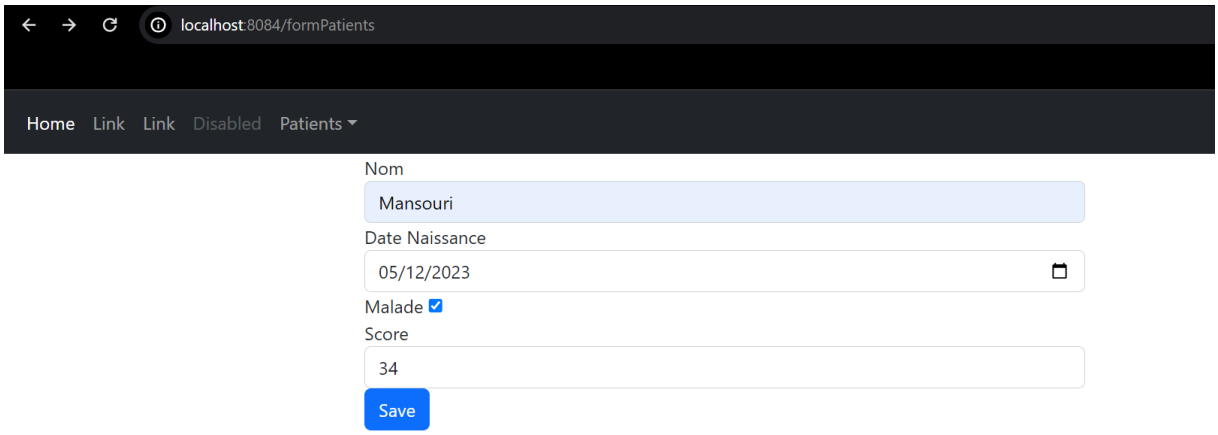
On crée une méthode save() qui permet d'ajouter un patient dans la classe contrôleur

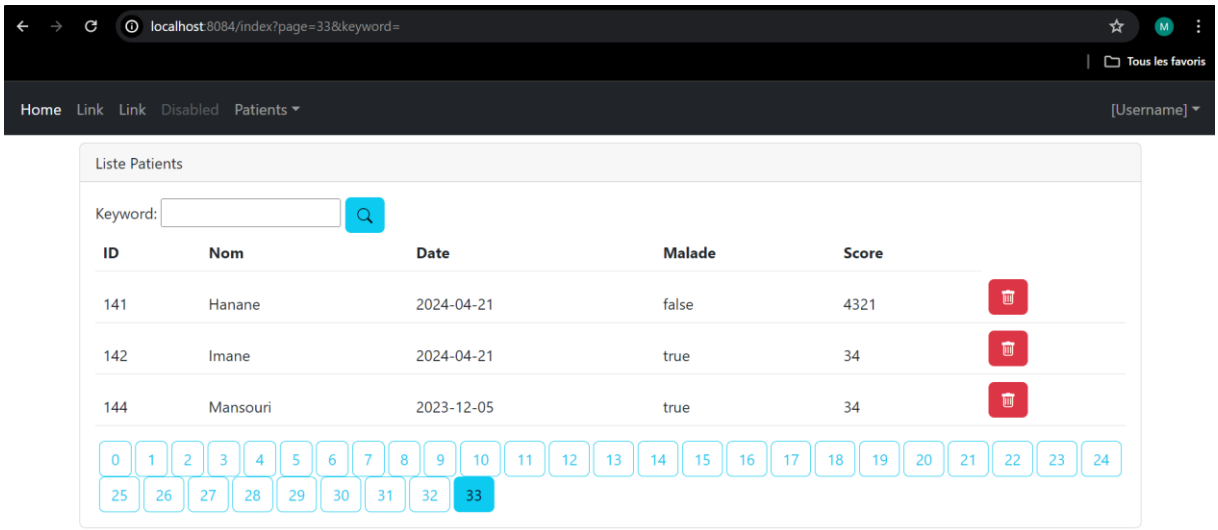


On utilise la notation @DateTimeFormat pour spécifier format des dates

```java
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





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

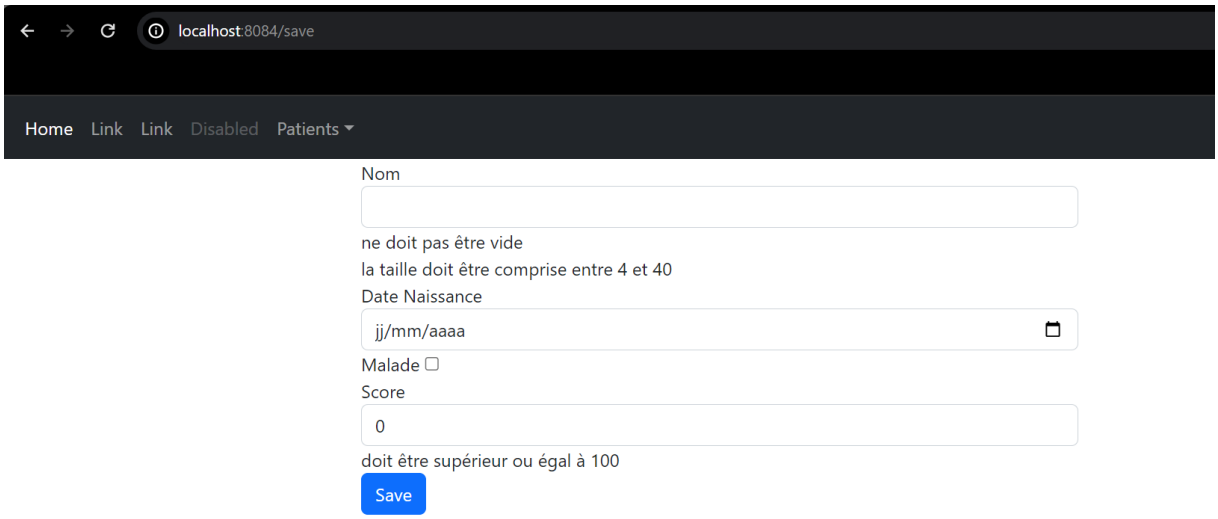
```
</dependency>
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-validation -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
    <version>2.6.5</version>
</dependency>
```

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

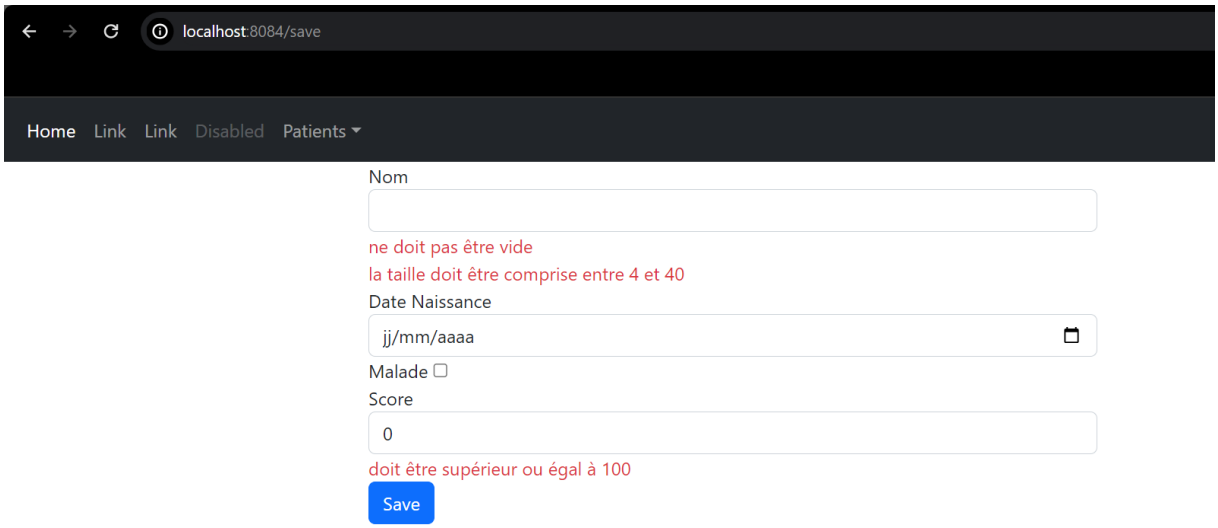
On ajoute la notation @Valid pour qu'on dire à spring mvc une fois que tu stocke les données dans un patient et faire la validation, et si jamais il y a des erreurs qui va nous sauter la connexion des erreurs dans une connexion de type BindingResult

```java
    @GetMapping("/formPatients")
    public String formPatient(Model model) {
        model.addAttribute( attributeName: "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";

    }
}
```

```java
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;
}
```

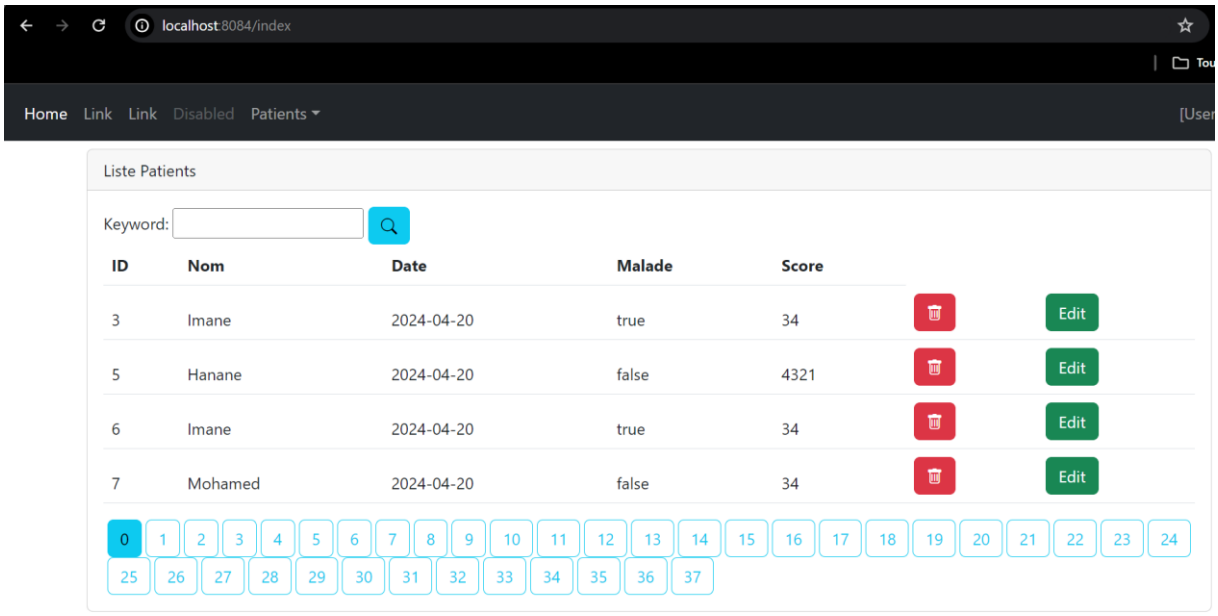


On met les messages en rouge



On ajoute un bouton Edit

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



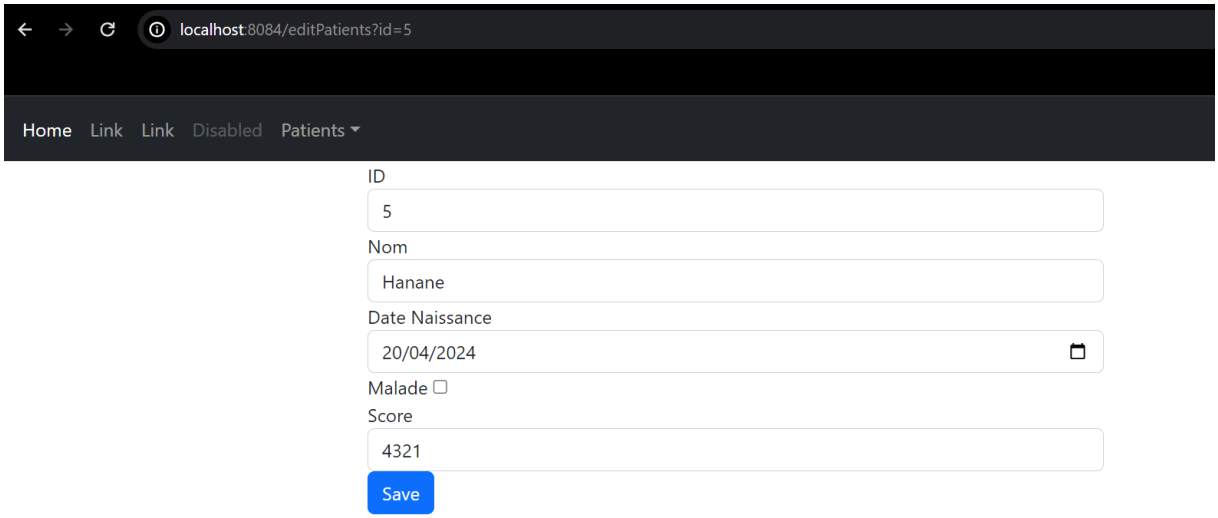
On ajoute la méthode editPatient au contrôleur pour modifier les informations d'un patient

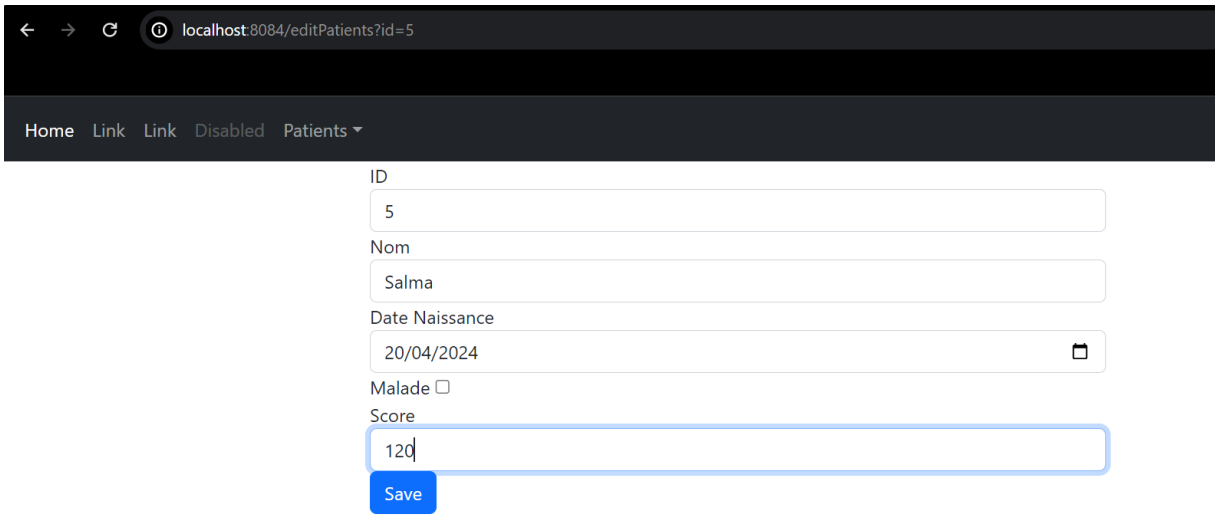
```
@GetMapping("/formPatients")
public String formPatient(Model model) {
    model.addAttribute( attributeName: "patient", new Patient());
    return "formPatients";
}
@PostMapping (path = "/save")
public String save(Model model, @Valid Patient patient, BindingResult bindingResult) {
    if(bindingResult.hasErrors())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

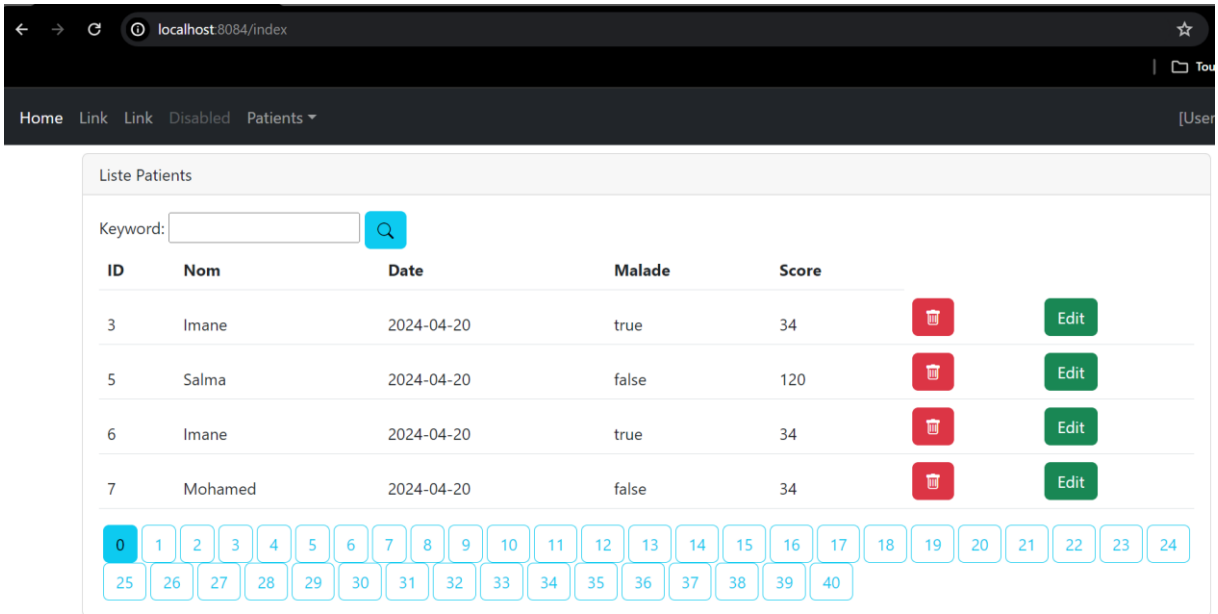
On va créer un fichier HTML editPatients.html

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







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

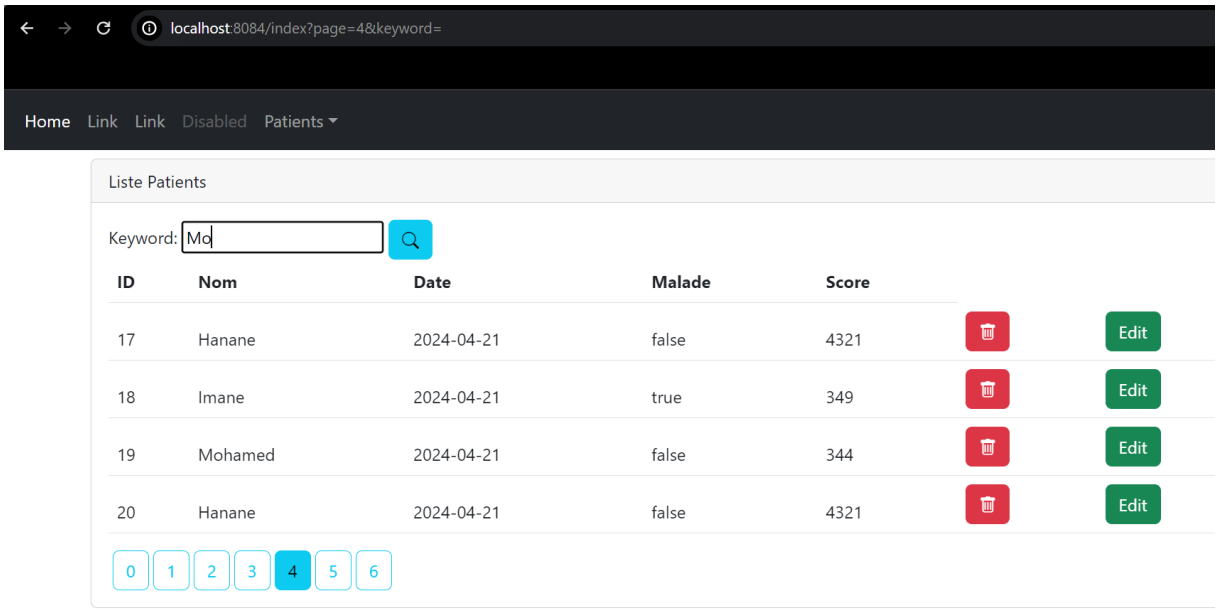
```
    @GetMapping("/editPatients")
    public String editPatient(Model model, Long id, String keyword, int page) {
        Patient patient=patientRepository.findById(id).orElse( other: null);
        if(patient==null) throw new RuntimeException("Patient introuvable");
        model.addAttribute( attributeName: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

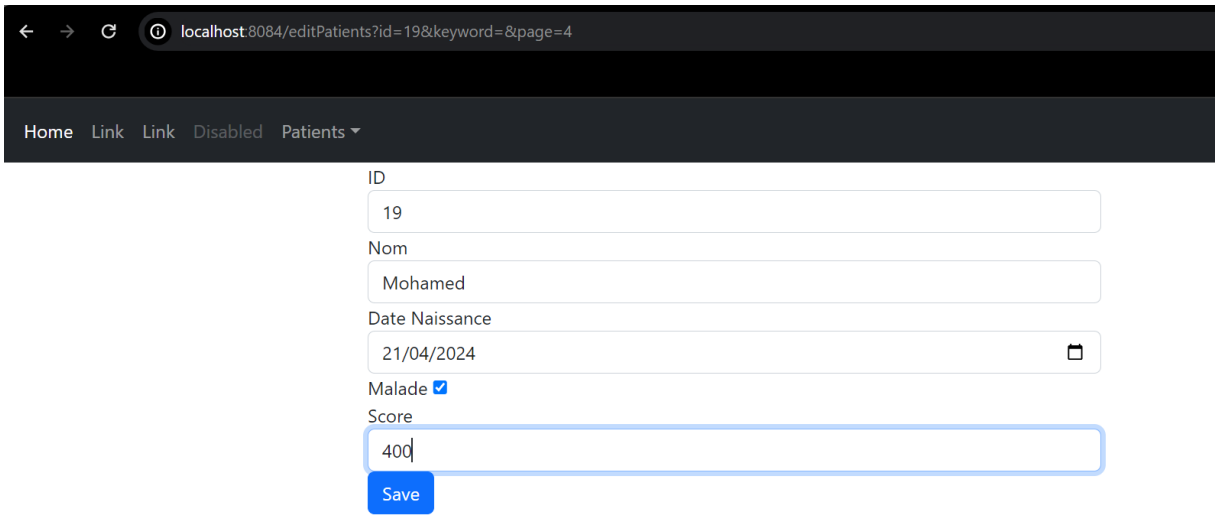
On ajoute les paramètres page et keyword dans save au niveau de th:action et on les ajouter dans la méthode save dans le contrôleur

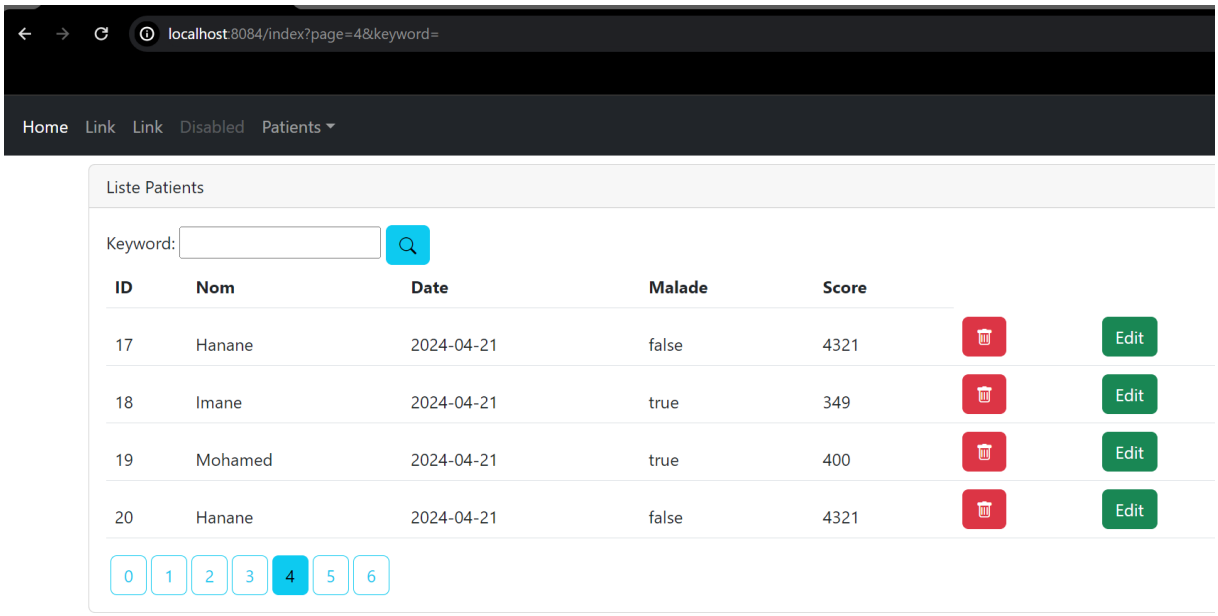
```
<div layout:fragment="content1">
    <div class="col-md-6 offset-3">
    <form method="post" th:action="@{save(page=${page},keyword=${keyword})}">
        <div>
            <label>ID</label>
            <input class="form-control" type="text" name="id" th:value="${patient.id}">
```

```
    @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

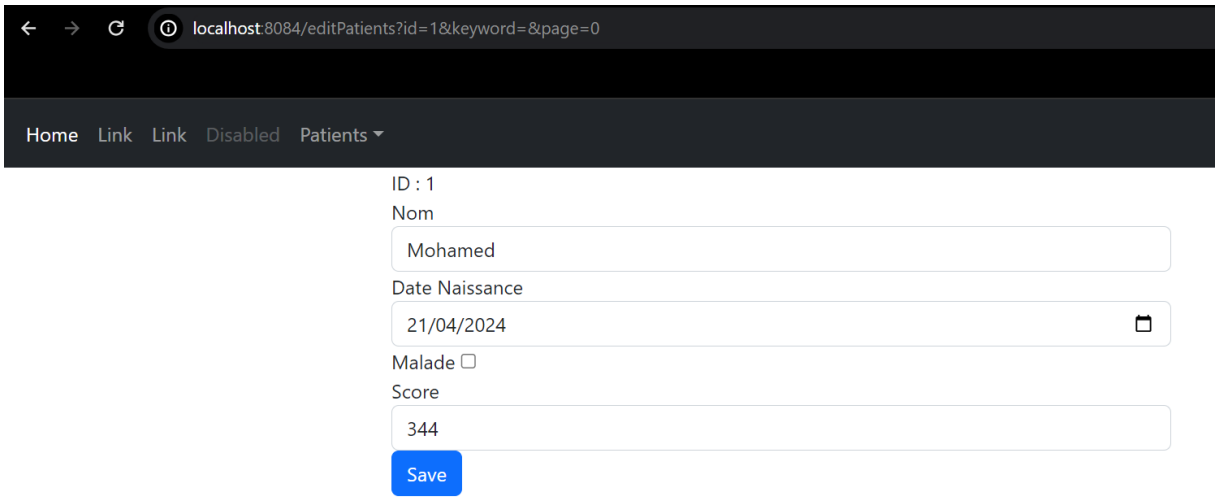






On cache la zone de texte de l'id et on affiche sa valeur mais on ne peut pas le modifier

```
<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>
```

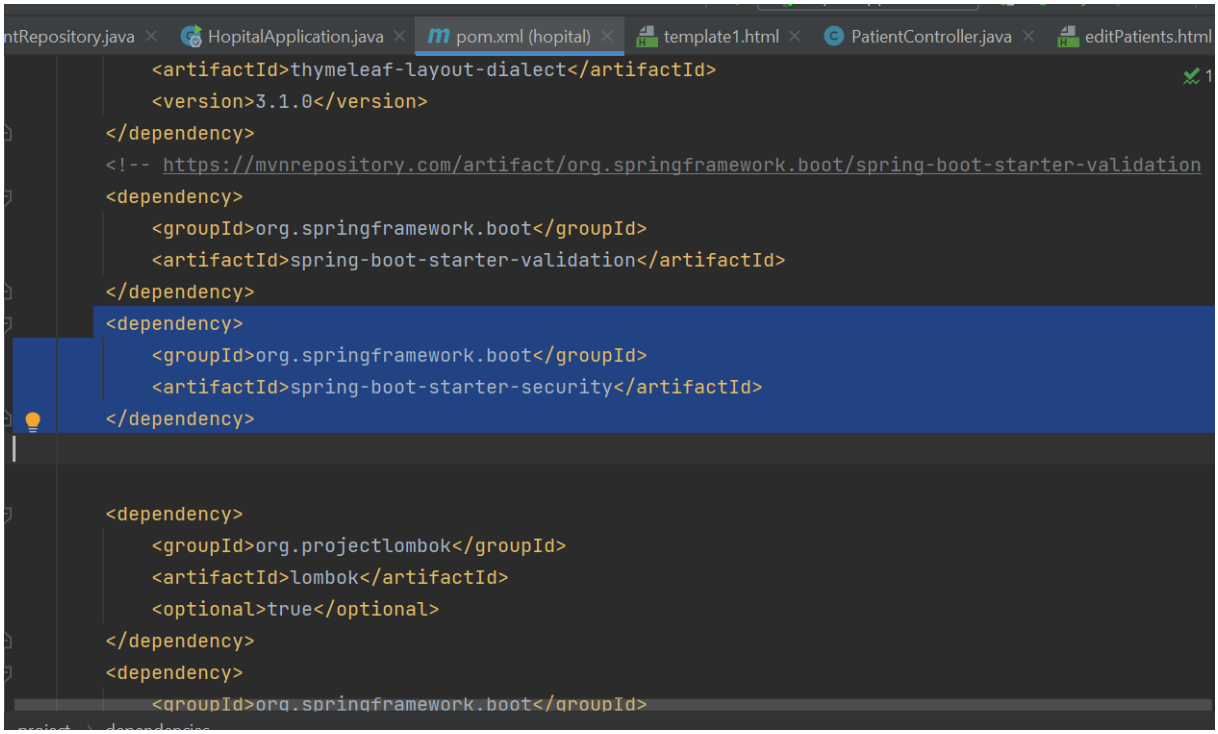


Partie 3 :

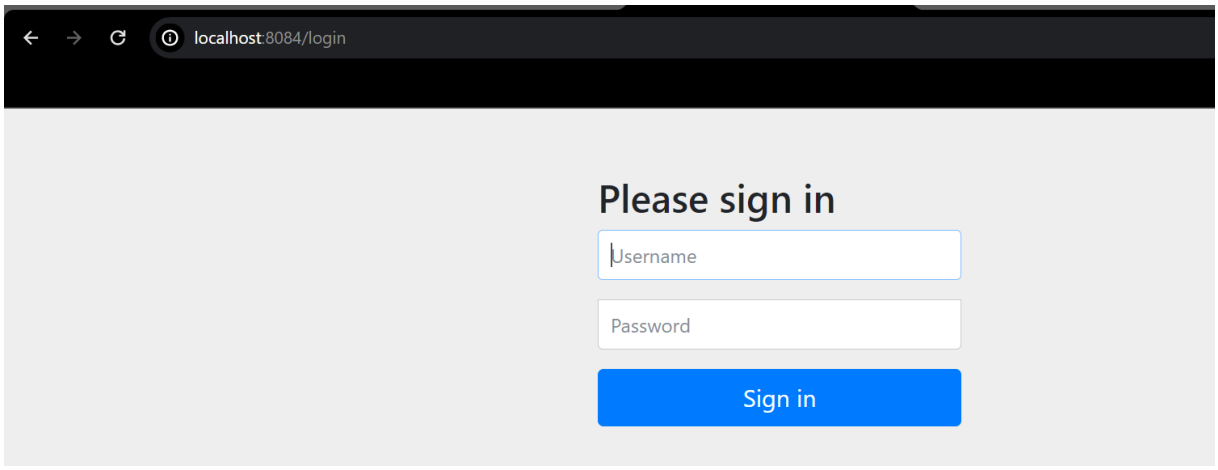
Sécurité avec Spring security

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

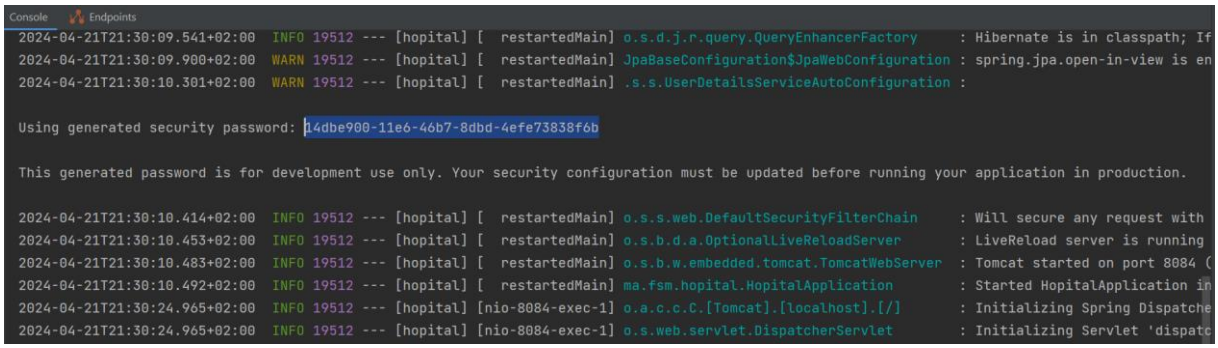
Tout d'abord, on va ajouter la dépendance security

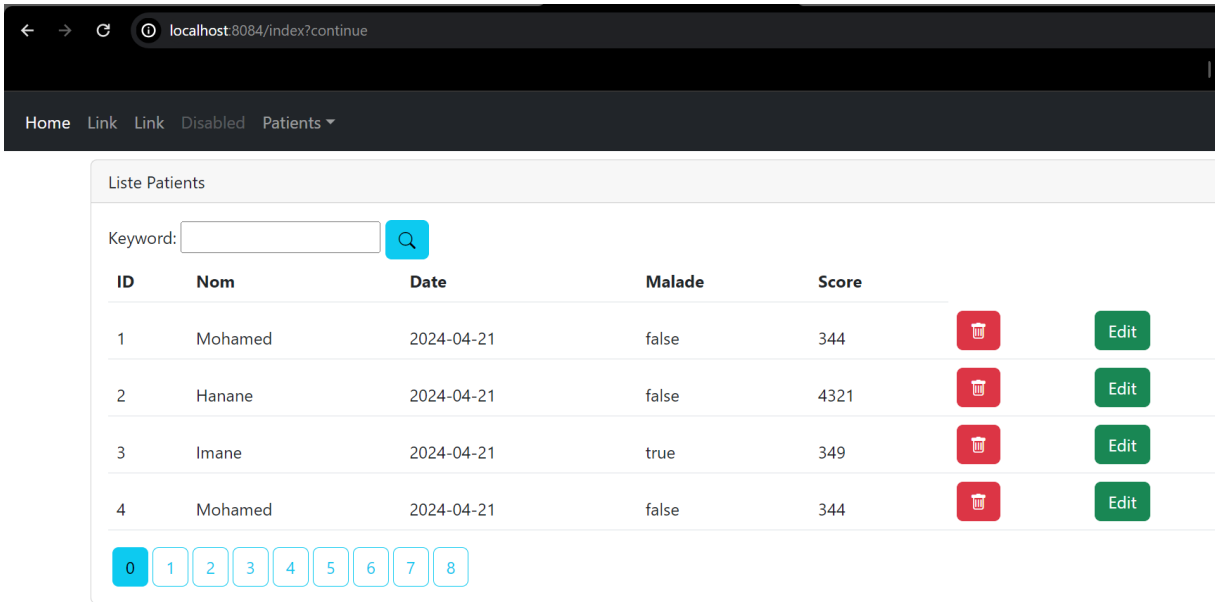


On démarre l'application et quand on accède, on trouve qu'on a une configuration par défaut



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





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

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

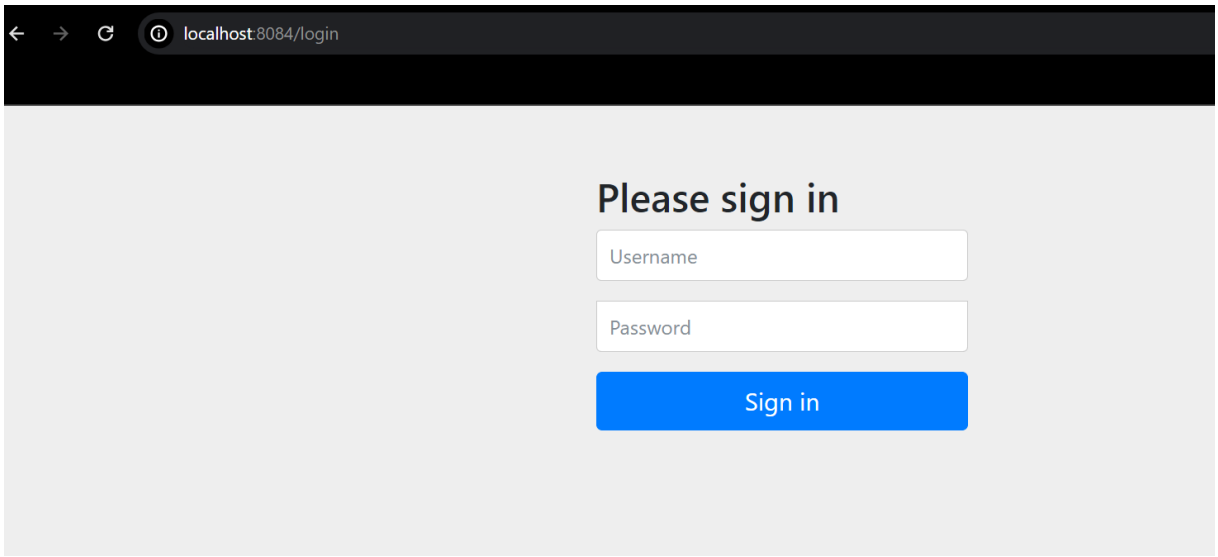
On utilise également l'annotation @Bean pour que notre méthode s'exécute au démarrage.

```java
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

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

Et le formulaire de l’authentification s’affiche

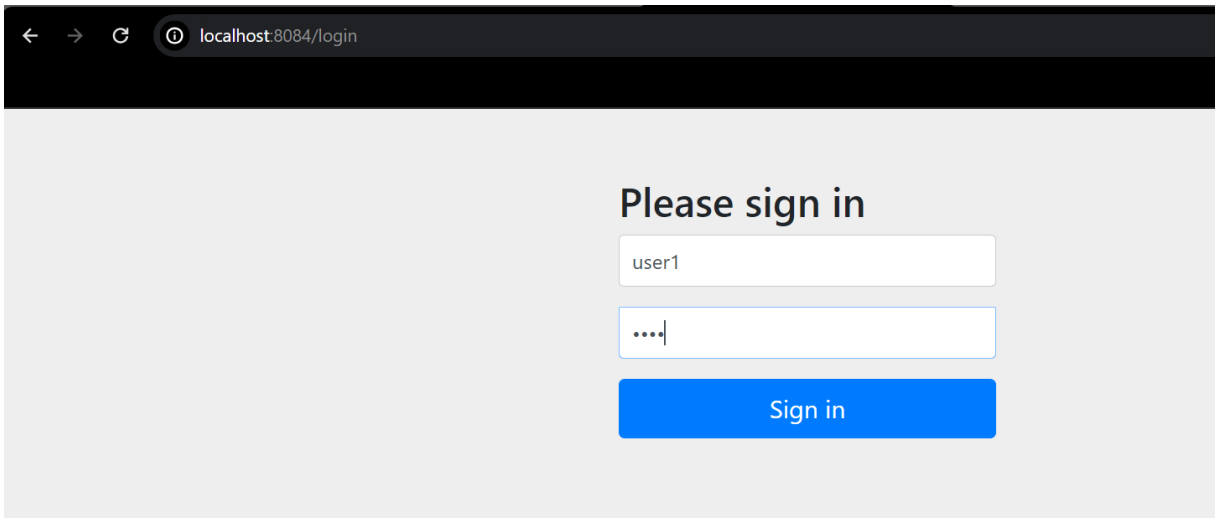


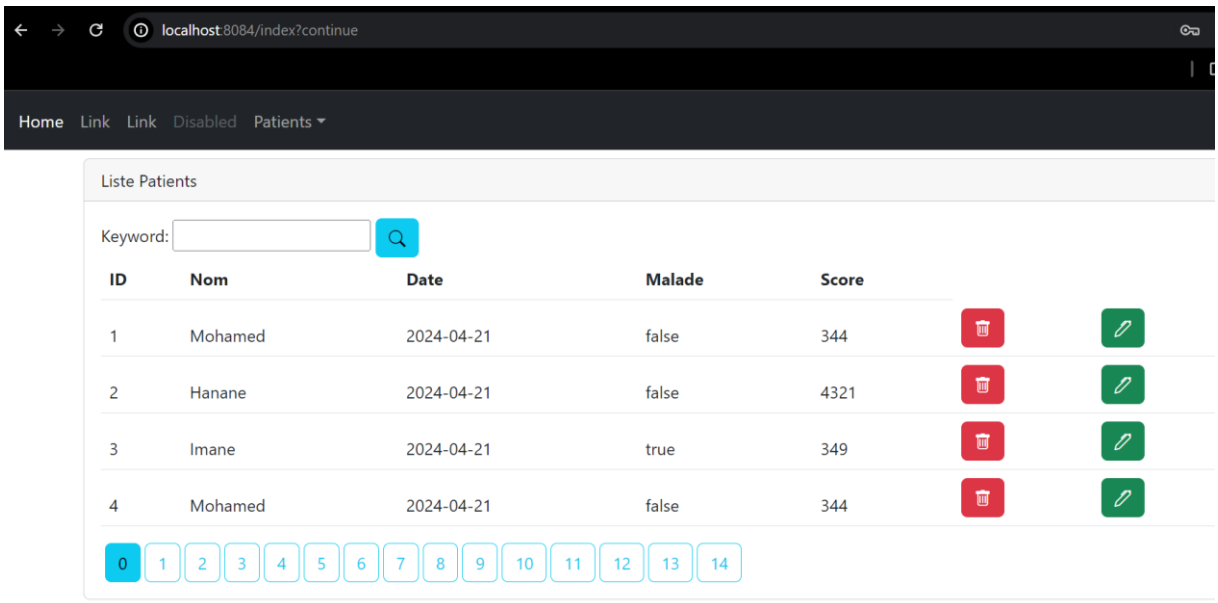
On doit créer des autorisations pour les utilisateurs, pour cela on utilise un objet de type InMemoryAuthentication, qui permet de préciser au mémoire les utilisateurs qui ont le droit d'accéder à l'application.

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

```java
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 :





Pour travailler avec spring security on utilise un password encoder BCrypt

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

```
Config.java × HopitalApplication.java ×
                                                                    ⚠1 ⚠1 ✔3 ^
    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();
    }

}
```

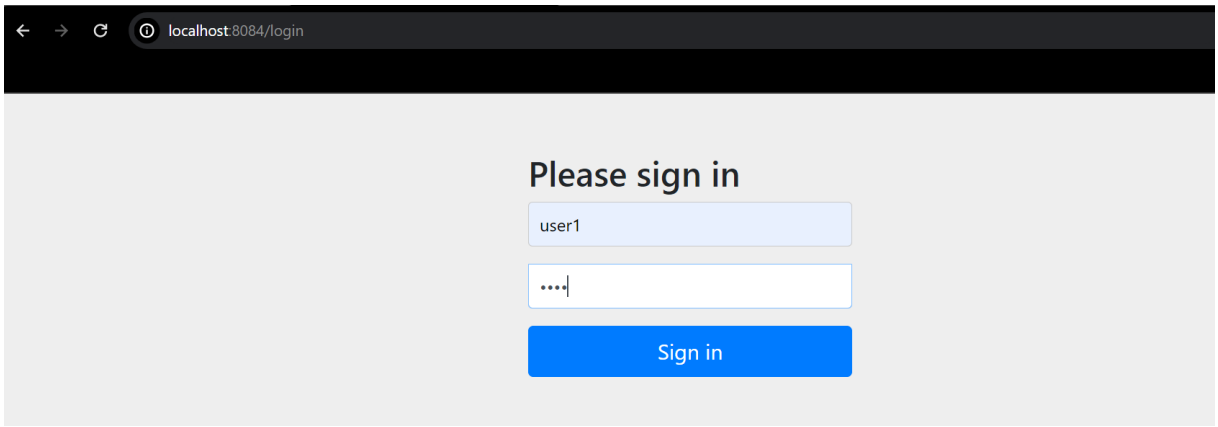
On affiche le nom du user connecté au lieu de [User], et on ajoute une opération du logout.

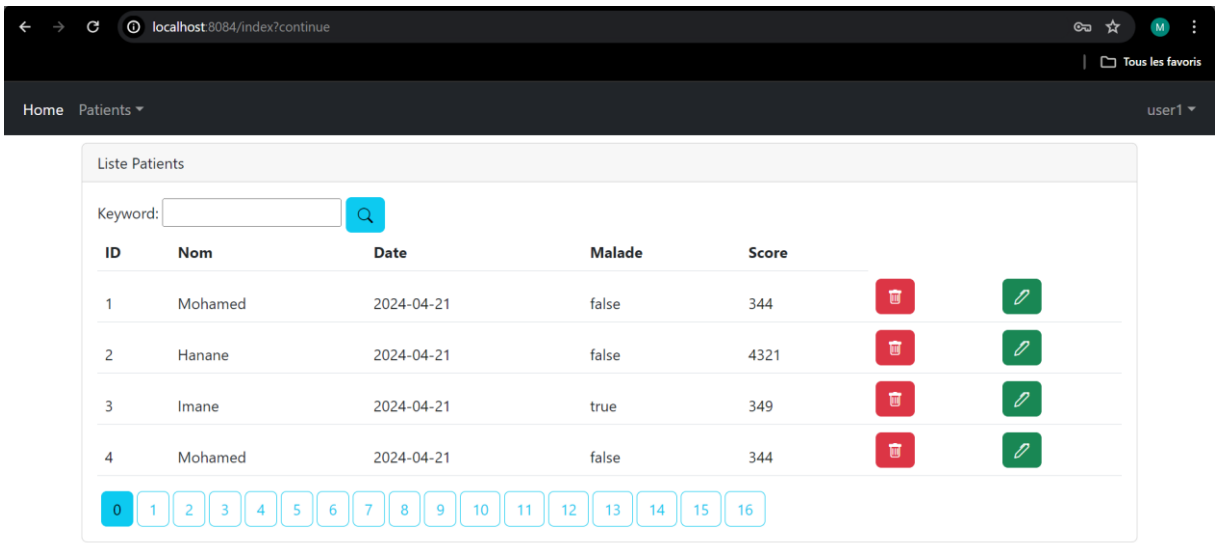
Pour afficher l'utilisateur authentifié, on est besoin d'ajouter une autre dépendance thymeleaf extras

```
                                                                    ✔1
            <artifactId>lombok</artifactId>
            <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

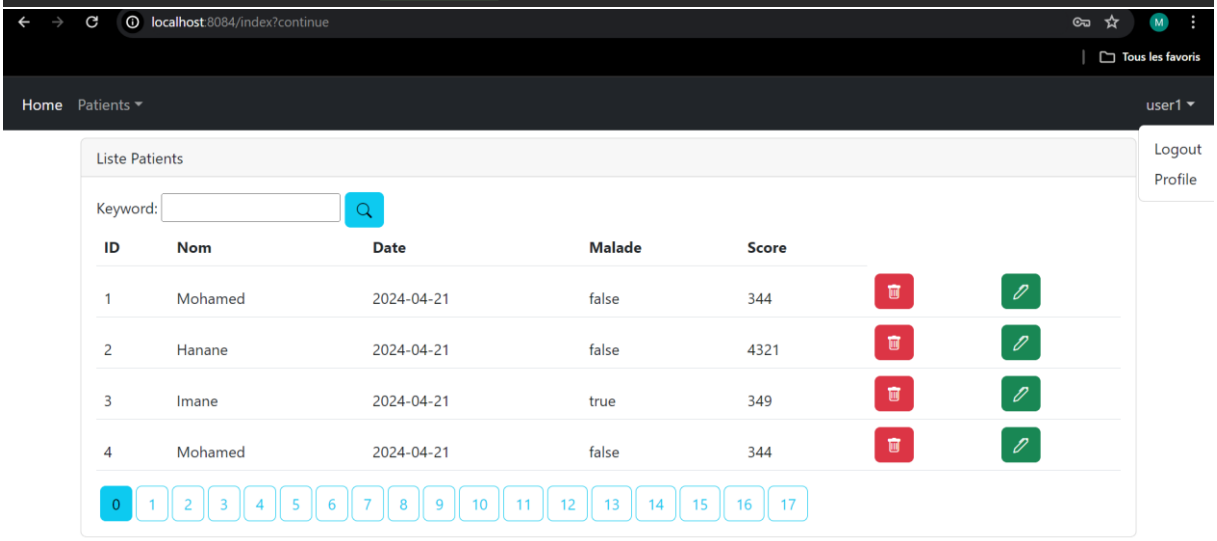
```
    </li>
    <li class="nav-item dropdown">
      <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown">Patients</a>
      <ul class="dropdown-menu">
        <li><a class="dropdown-item" th:href="@{formPatients}">Nouveau</a></li>
        <li><a class="dropdown-item"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

  </ul>
</div>
```

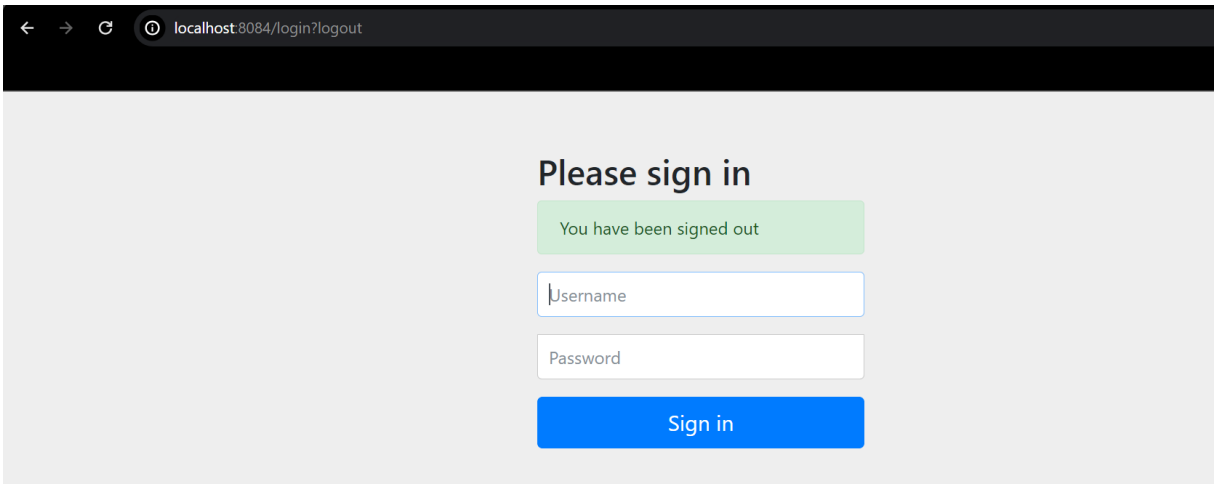




Pour se déconnecter, on passe par un autre bouton logout

```
  </ul>
  <ul class ="navbar-nav">
    <li class="nav-item dropdown">
      <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle=
         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>
```





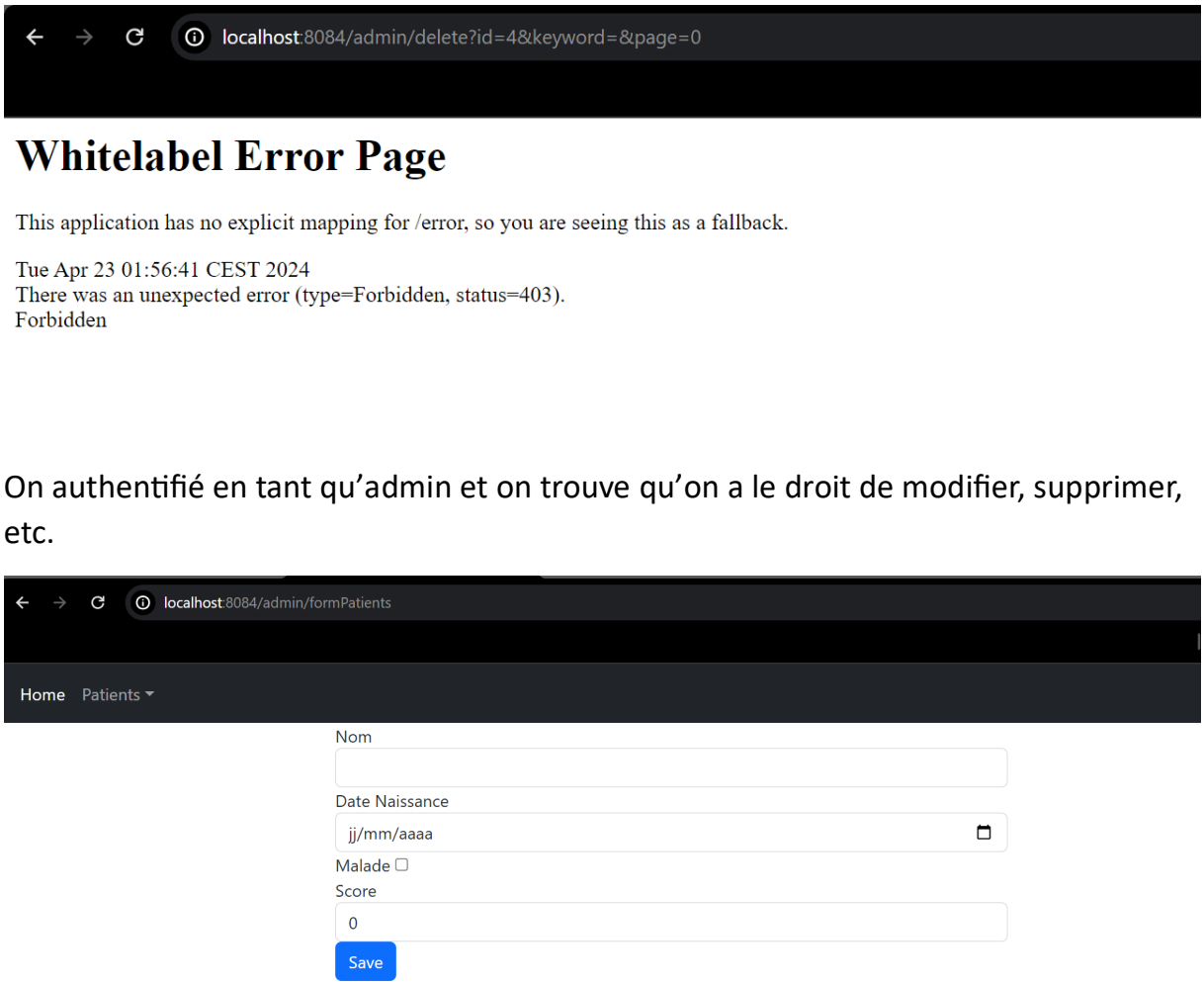
On ajoute un méthode Home() pour rediriger vers la page index

```
    @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( rawPassword: "1234")).roles("USER").build(),
                User.withUsername("user2").password(passwordEncoder.encode( rawPassword: "1234")).roles("USER").build(),
                User.withUsername("admin").password(passwordEncoder.encode( rawPassword: "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

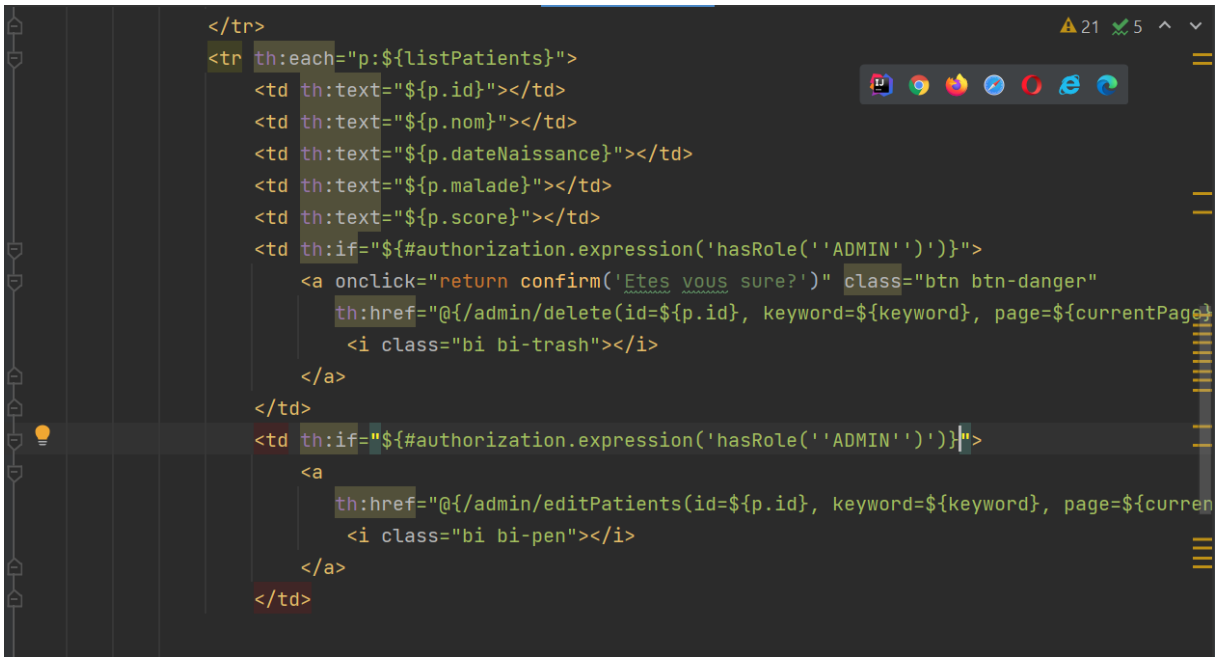


On authentifié en tant qu'admin et on trouve qu'on a le droit de modifier, supprimer, etc.

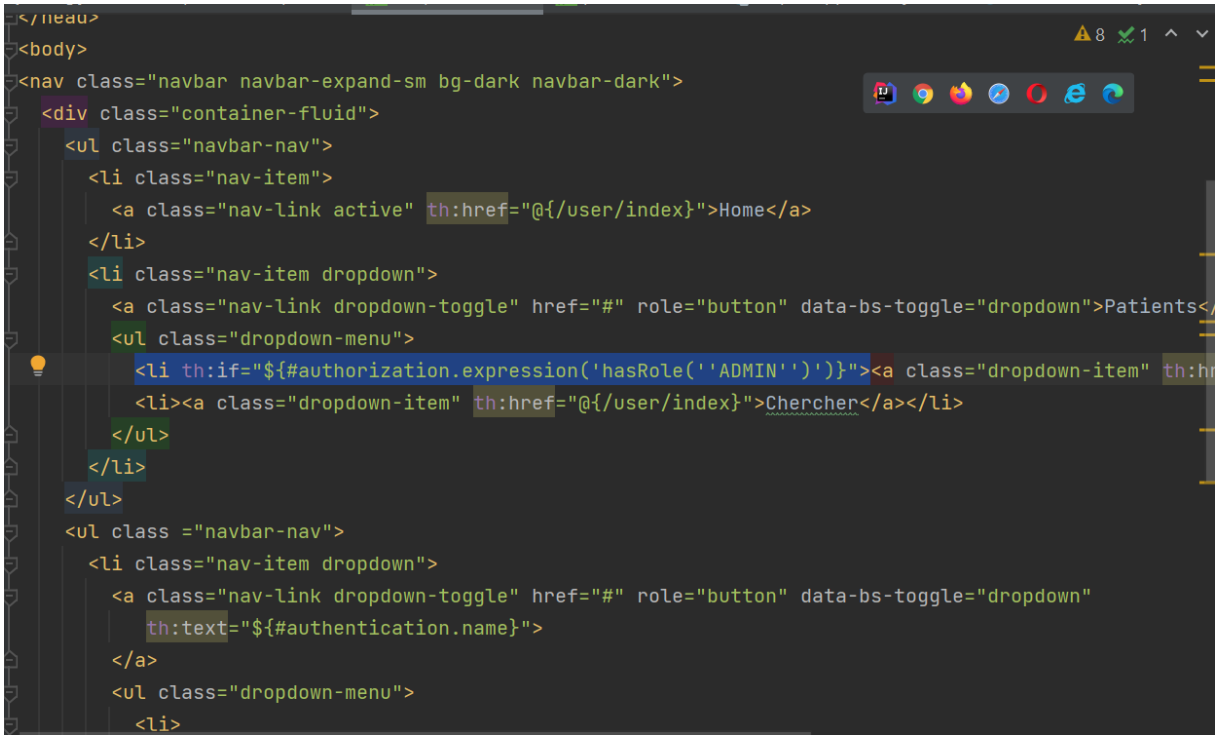
Si on se connecte en tant que admin, on peut faire tous les actions.

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 :

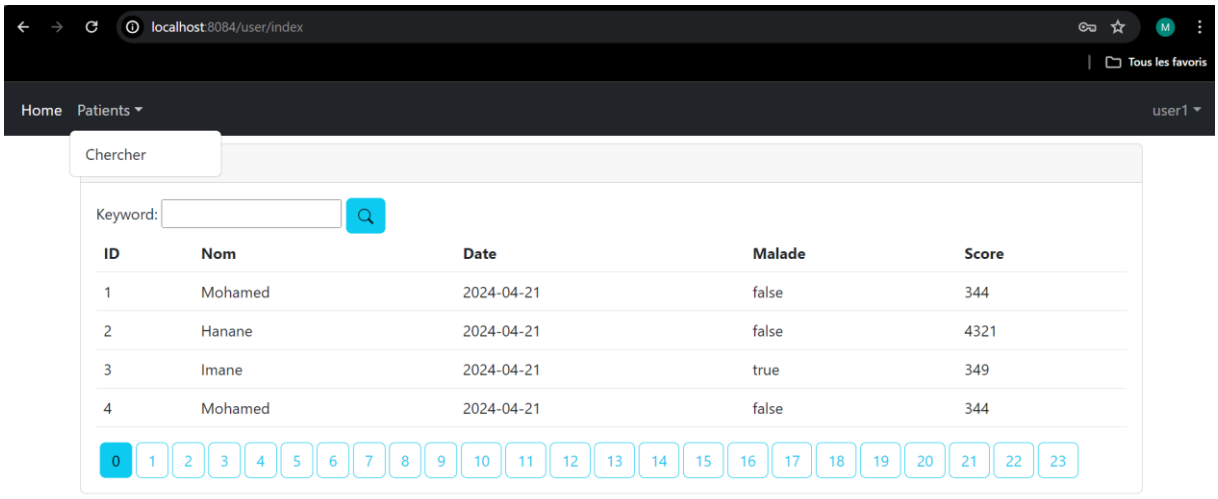


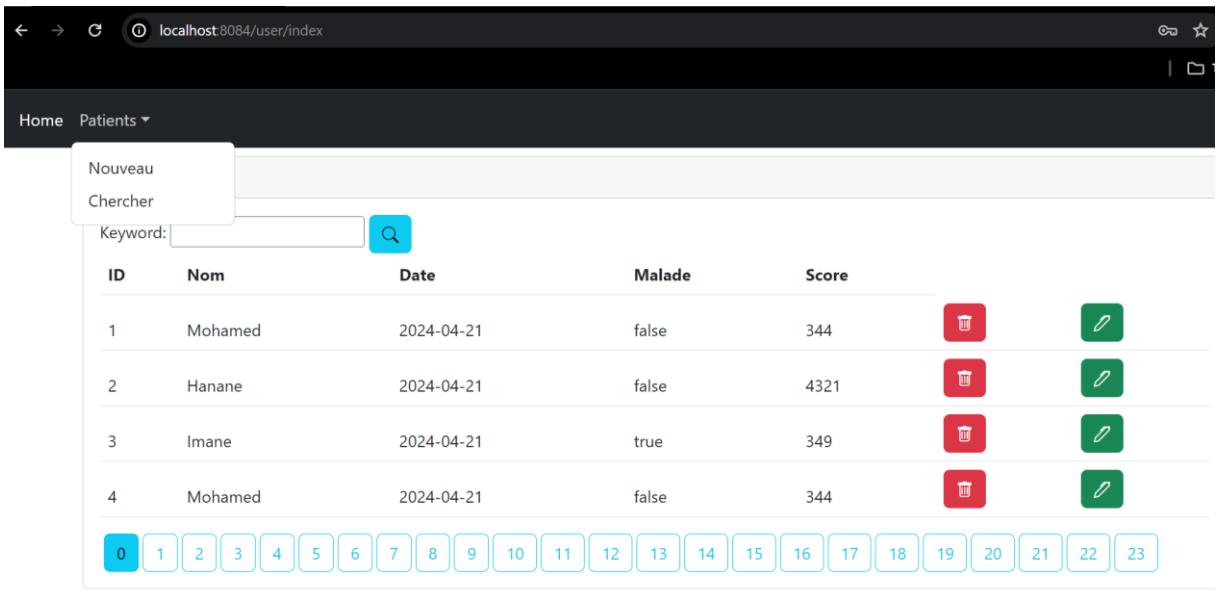
```
</tr>
<tr th:each="p:${listPatients}">
    <td th:text="${p.id}"></td>
    <td th:text="${p.nom}"></td>
    <td th:text="${p.dateNaissance}"></td>
    <td th:text="${p.malade}"></td>
    <td th:text="${p.score}"></td>
    <td th:if="${#authorization.expression('hasRole(''ADMIN'')')}">
        <a onclick="return confirm('Etes vous sure?')" class="btn btn-danger"
           th:href="@{/admin/delete(id=${p.id}, keyword=${keyword}, page=${currentPage}
            <i class="bi bi-trash"></i>
        </a>
    </td>
    <td th:if="${#authorization.expression('hasRole(''ADMIN'')')}">
        <a
           th:href="@{/admin/editPatients(id=${p.id}, keyword=${keyword}, page=${curren
            <i class="bi bi-pen"></i>
        </a>
    </td>
```



```
</head>
<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</
        <ul class="dropdown-menu">
          <li th:if="${#authorization.expression('hasRole(''ADMIN'')')}"><a class="dropdown-item" th:hr
          <li><a class="dropdown-item" th:href="@{/user/index}">Chercher</a></li>
        </ul>
      </li>
    </ul>
    <ul class ="navbar-nav">
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown"
           th:text="${#authentication.name}">
        </a>
        <ul class="dropdown-menu">
          <li>
```

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





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 :

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

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

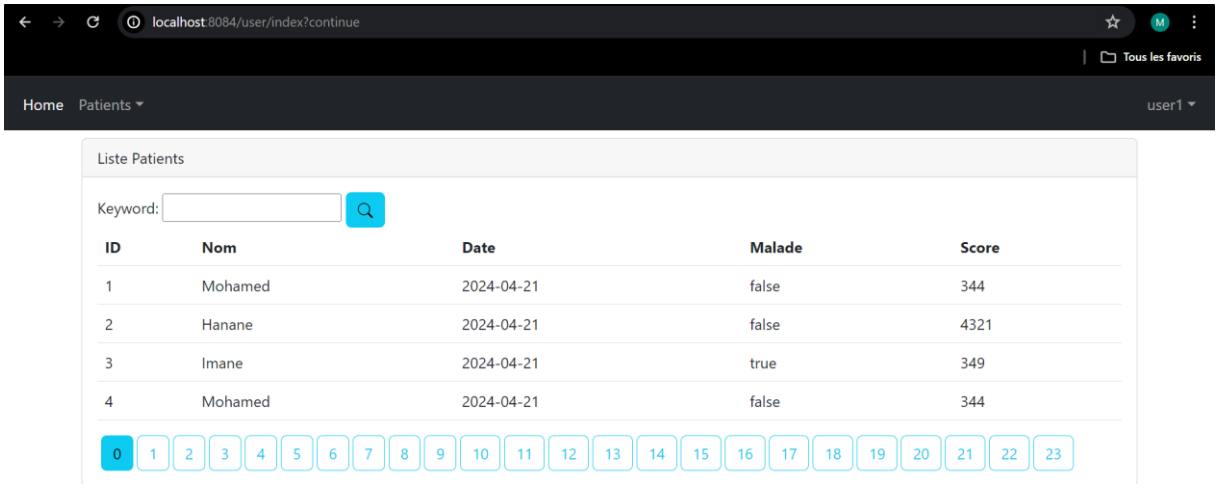
Et on ajoute un autre contrôleur SecurityController

```java
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";

    }

}
```

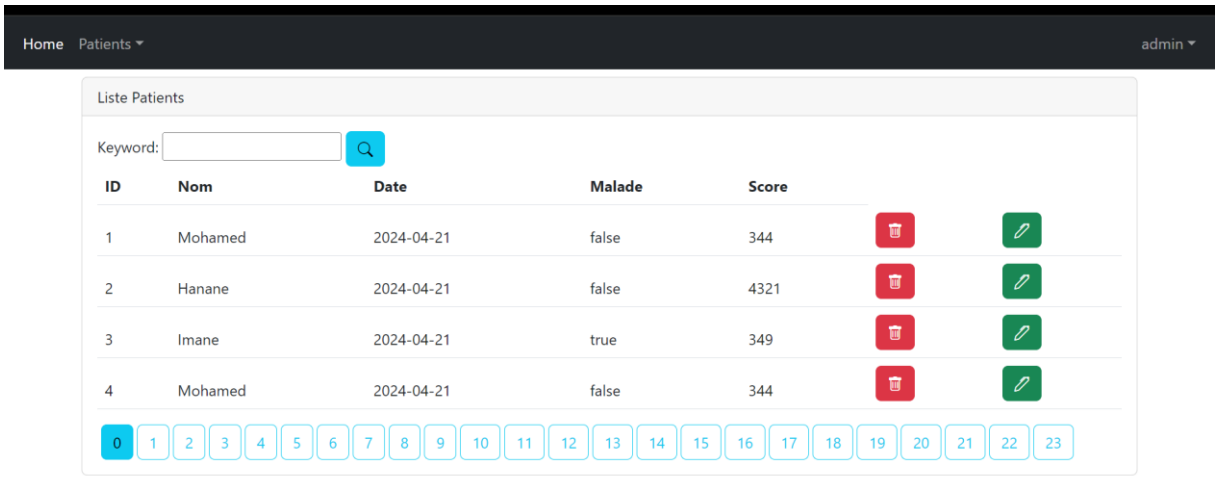
Et aussi une vue notAuthorized.html :

```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 :



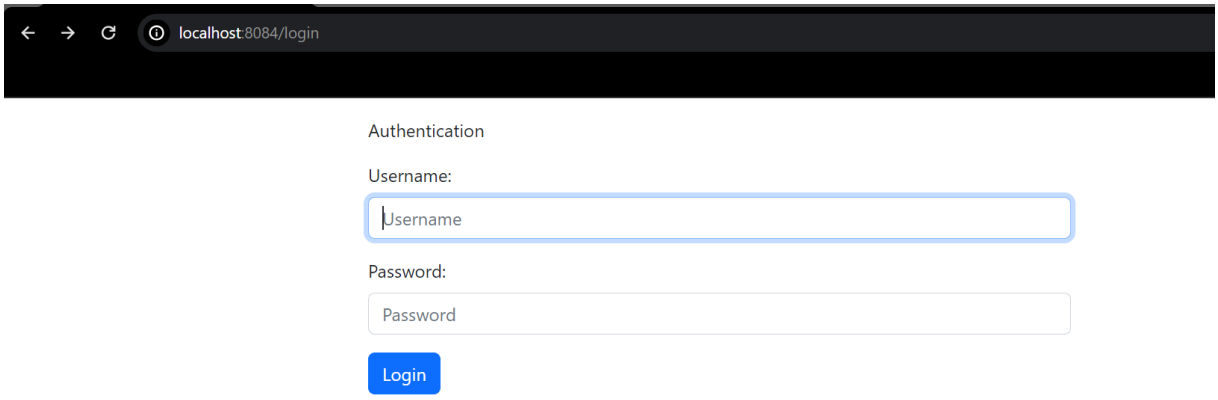
Page admin :



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" nam
                   </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"
                   </div>
                   <button type="submit" class="btn btn-primary">Login</button>
```

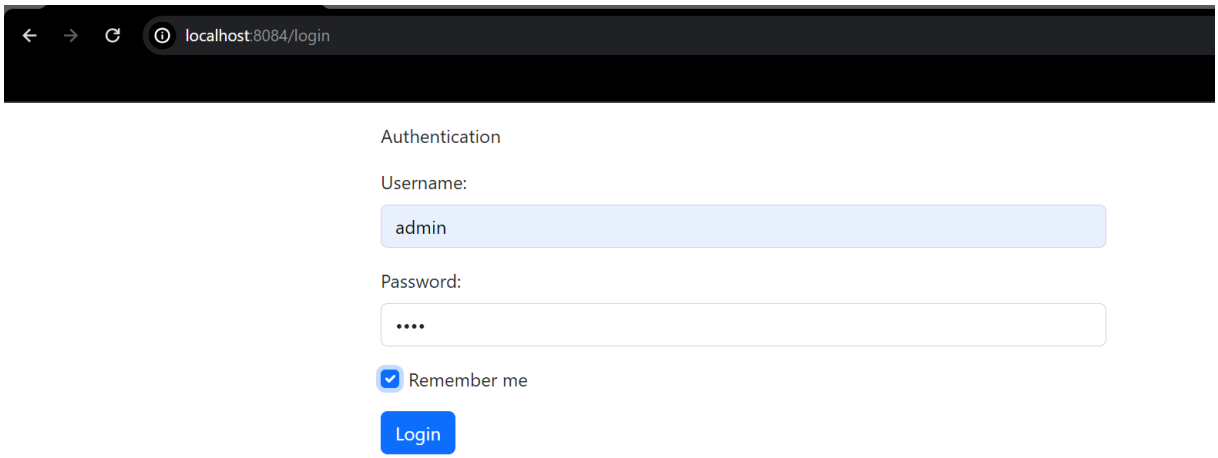
```
    return new InMemoryUserDetailsManager(
            User.withUsername("user1").password(passwordEncoder.encode( rawPassword: "1234")).roles("USE
            User.withUsername("user2").password(passwordEncoder.encode( rawPassword: "1234")).roles("USE
            User.withUsername("admin").password(passwordEncoder.encode( rawPassword: "1234")).roles("USE
    );
}
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.formLogin().loginPage("/login").permitAll();
    httpSecurity.authorizeHttpRequests().requestMatchers( ...patterns: "/webjars/**", "/h2-console/**").per
    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();
}
```

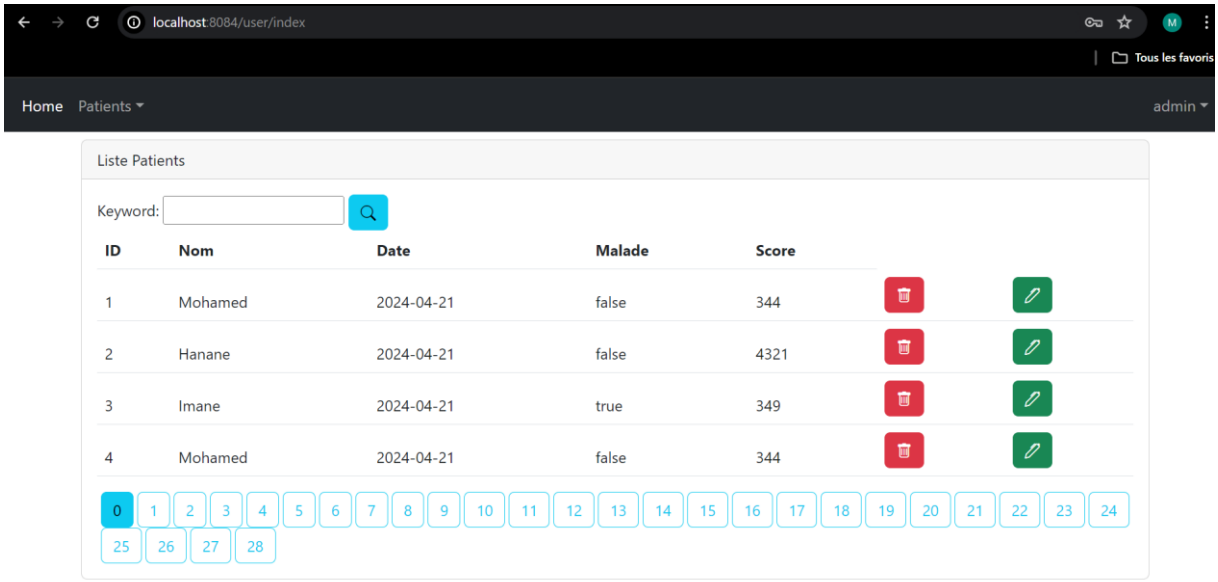


On veut ajouter une case à cocher Remember me

```
        <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" nam
                </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"
                </div>
                <div class="form-check mb-3">
                    <label class="form-check-label">
                        <input class="form-check-input" type="checkbox" name="remember-me"> Rememb
                    </label>
                </div>
                <button type="submit" class="btn btn-primary">Login</button>
            </form>
        </div>

    </div>
</div>
```

```
@Bean
public InMemoryUserDetailsManager inMemoryUserDetailsManager() {
    return new InMemoryUserDetailsManager(
            User.withUsername("user1").password(passwordEncoder.encode( rawPassword: "1234")).roles("USE
            User.withUsername("user2").password(passwordEncoder.encode( rawPassword: "1234")).roles("USE
            User.withUsername("admin").password(passwordEncoder.encode( rawPassword: "1234")).roles("USE
    );
}
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.formLogin().loginPage("/login").permitAll();
    httpSecurity.rememberMe();
    httpSecurity.authorizeHttpRequests().requestMatchers( ...patterns: "/webjars/**", "/h2-console/**").pe
    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();
}
```





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.

```java
import org.springframework.security.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( rawPassword: "1234")).roles
                User.withUsername("user2").password(passwordEncoder.encode( rawPassword: "1234")).roles
                User.withUsername("admin").password(passwordEncoder.encode( rawPassword: "1234")).roles
        );
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
```

```java
        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( attributeName: "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);
```