

Master Intelligence Artificielle et Analyse des Données

Systèmes Distribués

COMPTE RENDU DU TP3

Réalisé par :

HALIMA DAOUDI

Année universitaire : 2023 – 2024

INTRODUCTION

Dans ce travail pratique, je vais me plonger dans le développement d'une application Web JEE utilisant les technologies Spring MVC, Thymeleaf et Spring Data JPA. Cette application sera conçue pour gérer efficacement les patients, en offrant une gamme de fonctionnalités allant de l'affichage et la pagination à la suppression des patients, tout en intégrant des améliorations pour une expérience utilisateur optimale.

Partie 1 : Créer une application Web JEE basée sur Spring MVC, Thymeleaf et Spring Data JPA qui permet de gérer les patients. L'application doit permettre les fonctionnalités suivantes :

- Afficher les patients
- Faire la pagination
- Chercher les patients
- Supprimer un patient
- Faire des améliorations supplémentaires

Partie 2 : Créer une page template et mettre en place la validation des formulaires.

Partie 3 : Assurer la sécurité de l'application avec Spring Security.

- InMemory Authentication
- JDBC Authentication
- UserDetails Service

Partie 1 :

Dans le package **entities**, je crée une entité JPA nommée **Patient** pour représenter les informations d'un patient. Cette entité a des attributs tels que l'id qui agit comme clé primaire et est généré automatiquement par la base de données et le nom, la date de naissance, l'état de santé (malade), et un score. L'annotation **@Entity** indique que cette classe est une entité JPA associée à une table dans la base de données. Les annotations de Lombok (**@Data**, **@NoArgsConstructor**, **@AllArgsConstructor**) permettent de générer automatiquement des méthodes pour accéder et manipuler les données de l'entité

```
package fs.miaad.entities;
import ...
6 usages
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Patient {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    @Temporal(TemporalType.DATE) // prend la date sans heure
    private Date dateNaissance;
    private boolean malade;
    private int score;
}
```

Dans le package repository, j'ai créé une interface nommée **PatientRepository** qui étend **JpaRepository**. Cette extension permet d'utiliser les méthodes fournies par JpaRepository pour interagir avec la base de données

```
package fs.miaad.repository;
import ...
2 usages
@Repository
public interface PatientRepository extends JpaRepository<Patient, Long> {
}
```

J'ai créé une classe principale nommée "**Tp3DaoudiHalimaApplication**" et je l'ai annotée avec "**@SpringBootApplication**" pour définir le point d'entrée de mon application Spring Boot. En implémentant l'interface "**CommandLineRunner**", j'ai personnalisé la méthode "**run**" pour injecter un "**PatientRepository**" grâce à "**@Autowired**", me permettant ainsi d'accéder à la base de données. Enfin, j'ai utilisé cette instance de "**PatientRepository**" pour ajouter des patients à la base de données en utilisant la méthode "**save**".

```
package fs.miaad;
import ...
@SpringBootApplication
public class Tp3DaoudiHalimaApplication implements CommandLineRunner {
    @Autowired
    private PatientRepository patientRepository;
    public static void main(String[] args) { SpringApplication.run(Tp3DaoudiHalimaApplication.class, args); }
    @Override
    public void run(String... args) throws Exception {
        patientRepository.save(new Patient(id: null, nom: "halima1", new Date(), malade: false, score: 35));
        patientRepository.save(new Patient(id: null, nom: "halima2", new Date(), malade: true, score: 10));
        patientRepository.save(new Patient(id: null, nom: "halima3", new Date(), malade: false, score: 70));
    }
}
```

Dans le fichier **application.properties**, je configure les paramètres essentiels pour que mon application fonctionne correctement avec la base de données. Cela inclut le nom de l'application, le port sur lequel elle écoute les requêtes, ainsi que l'url de la base de données H2 et l'activation de sa console

```
spring.application.name=TP3-DAOUDI-HALIMA
server.port=8085
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:db_patients
```

Voici les résultats obtenus lors de l'exécution de cette classe :

SELECT * FROM PATIENT;

DATE_NAISSANCE	MALADE	SCORE	ID	NOM
2024-04-20	FALSE	35	1	halima1
2024-04-20	TRUE	10	2	halima2
2024-04-20	FALSE	70	3	halima3

(3 rows, 3 ms)

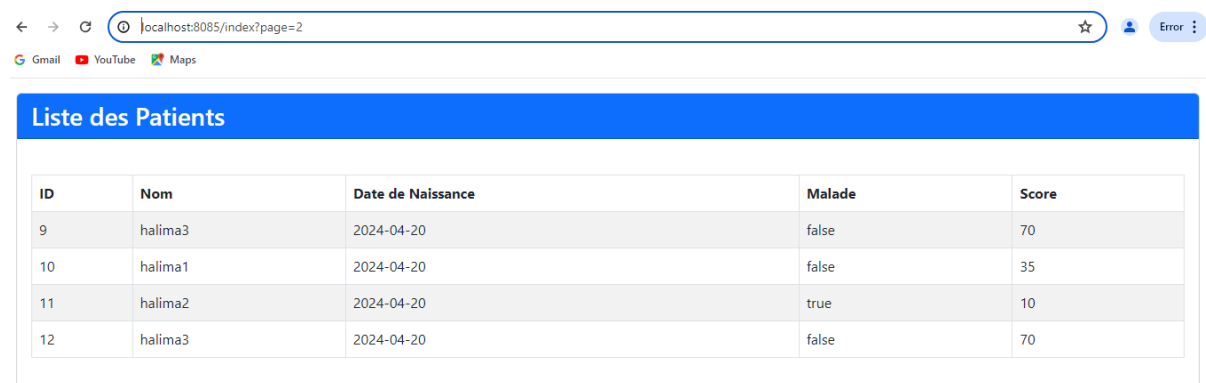
Dans le contrôleur **PatientController**, j'ai ajouté une méthode "index" qui gère les requêtes GET à l'URL "/index". Elle récupère une liste de tous les patients depuis la base de données et l'ajoute au modèle. Enfin, elle renvoie le nom de la vue "patients".

```
package fs.miaad.web;
import ...
@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";
    }
}
```

La vue "patients.html" utilise Thymeleaf pour afficher dynamiquement la liste des patients. Elle récupère les données transmises par le contrôleur PatientController et les intègre dans le contenu HTML. Cette approche permet une personnalisation fluide de l'affichage des patients directement dans la page web.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"><title>Patients</title>
    <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 bg-primary text-white"><h3 class="mb-0">Liste des Patients</h3></div>
        <div class="card-body">
            <table class="table mt-4 table-striped table-bordered">
                <thead class="table">
                    <tr><th>ID</th><th>Nom</th><th>Date de Naissance</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>
                    </tr>
                </tbody>
            </table>
        </div>
    </div></div>
</body>
</html>
```

En accédant à **localhost:8085/index** dans mon navigateur web, je peux voir la liste des patients affichée dynamiquement grâce à la vue "patients.html", qui intègre Thymeleaf.



The screenshot shows a web browser window with the address bar displaying 'localhost:8085/index?page=2'. Below the browser, there is a table titled 'Liste des Patients' with a blue header. The table has five columns: ID, Nom, Date de Naissance, Malade, and Score. It contains four rows of patient data.

ID	Nom	Date de Naissance	Malade	Score
9	halima3	2024-04-20	false	70
10	halima1	2024-04-20	false	35
11	halima2	2024-04-20	true	10
12	halima3	2024-04-20	false	70

Avant de migrer de H2 Database vers MySQL, je vais d'abord ajouter la dépendance **MySQL Connector/J** au fichier **pom.xml** de mon projet. Cette configuration spécifie que je veux utiliser le pilote JDBC MySQL dans mon application. La portée "runtime" indique que cette dépendance ne sera nécessaire que lors de l'exécution de mon application, et non lors de la compilation. Une fois que cette dépendance est ajoutée, je serai prêt à migrer mon application de H2 Database vers MySQL.

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

Je configure les paramètres dans le fichier **application.properties** pour migrer de H2 Database vers MySQL. Cela inclut la définition du nom de l'application et du port, ainsi que les informations de connexion à la base de données MySQL, telles que l'url, le nom d'utilisateur et le mot de passe. Une fois ces configurations ajoutées, l'application est prête à utiliser MySQL comme base de données.

```
spring.application.name=TP3-DAOUDI-HALIMA
server.port=8085
#Configuration pour MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/db_patients?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
```

Après avoir exécuté la classe **Tp3DaoudiHalimaApplication**, nous avons vérifié l'état de notre base de données MySQL à l'aide de phpmyadmin. Voici un aperçu de la table patient et des données stockées dans la base de données "**db_patients**". Cela confirme que la migration de H2 vers MySQL s'est déroulée avec succès

<div><div><div></div><div></div><div></div></div></div>				id	nom	date_naissance	malade	score	
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	1	halima1	2024-04-20	0	35
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	2	halima2	2024-04-20	1	10
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	3	halima3	2024-04-20	0	70
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	4	halima1	2024-04-20	0	35
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	5	halima2	2024-04-20	1	10
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	6	halima3	2024-04-20	0	70
<input type="checkbox"/>		Éditer	 Copier	 Supprimer	7	halima1	2024-04-20	0	35

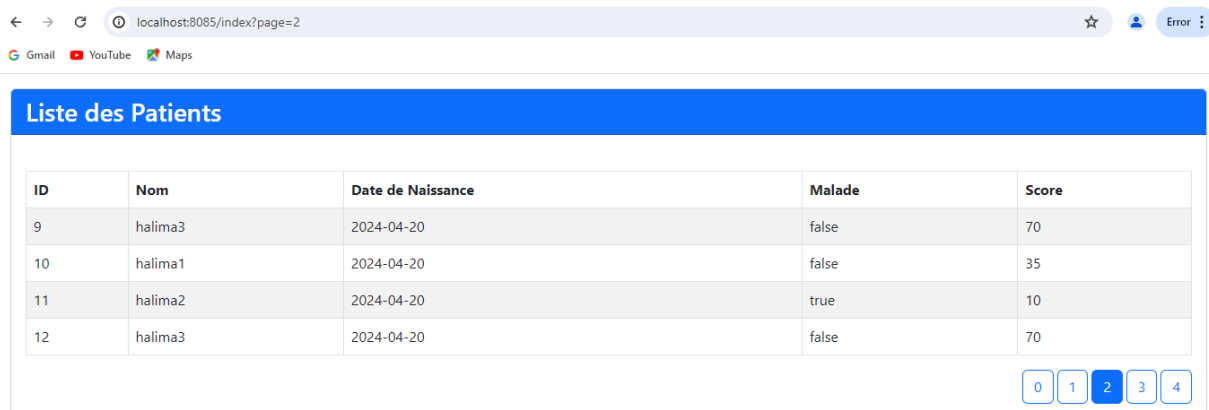
Dans le contrôleur **PatientController**, j'ai amélioré la méthode "**index**" pour intégrer la pagination. Elle récupère désormais la liste des patients paginée en fonction des paramètres "page" et "size" passés dans l'URL. Les attributs "pages" et "currentPage" sont ajoutés au modèle pour faciliter la navigation entre les pages et afficher la page actuelle. Enfin, le nom de la vue "patients" est retourné.

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

Dans la vue "**patients.html**", j'ai ajouté une liste déroulante permettant la navigation entre les pages de la liste des patients. Chaque numéro de page est affiché comme un lien cliquable, permettant de passer à la page correspondante

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

En accédant au lien, j'ai pu observer le résultat de la pagination intégrée, permettant une navigation fluide entre les pages de la liste des patients.



ID	Nom	Date de Naissance	Malade	Score
9	halima3	2024-04-20	false	70
10	halima1	2024-04-20	false	35
11	halima2	2024-04-20	true	10
12	halima3	2024-04-20	false	70

Dans l'interface **PatientRepository**, j'ai ajouté deux méthodes pour rechercher des patients : "findByNomContains" permet de trouver les patients dont le nom contient un mot-clé spécifique, tandis que "Chercher" utilise une requête JPQL personnalisée pour une recherche similaire. Les deux méthodes retournent une page de résultats paginée.

```
package fs.miaad.repository;
import ...
4 usages
@Repository
public interface PatientRepository extends JpaRepository<Patient, Long> {
    // 2 methodes pour rechercher
    1 usage
    Page<Patient> findByNomContains(String keyword , Pageable pageable);
    no usages
    @Query("select p from Patient p where p.nom like :x")
    Page<Patient> Chercher(@Param("x") String keyword , Pageable pageable);
}
```

Dans la méthode "index" du contrôleur **PatientController**, j'ai ajouté un paramètre "keyword" pour prendre en compte la recherche par mots-clés. Cela permet de filtrer la liste des patients en fonction du terme spécifié. Les résultats sont ensuite affichés dans la vue "patients.html".


```

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

```

Dans la vue **patients.html**, j'ai ajouté un formulaire qui permet de rechercher des patients par mot-clé. Ce formulaire envoie une requête GET à l'URL `"/index"` avec le paramètre `"keyword"` pour la recherche.

```

<div class="card-body">
    <form method="get" th:action="@{index}" class="row g-2">
        <div class="col-auto">
            <label for="keyword" class="visually-hidden">Keyword :</label>
            <input th:value="${keyword}" type="text" class="form-control"
                id="keyword" name="keyword" placeholder="Mot-clé...">
        </div>
        <div class="col-auto">
            <button type="submit" class="btn btn-primary">Rechercher</button>
        </div>
    </form>
    <table class="table mt-4 table-striped table-bordered">

```

Dans la vue **patients.html**, j'ai ajouté le paramètre `"keyword"` à la construction des liens de pagination, permettant de conserver le mot-clé de recherche lors de la navigation entre les pages.

```

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

```

Voici la page en accédant au lien **index**, présentant la liste des patients avec la possibilité de pagination et de recherche par mot-clé.

The screenshot shows a web browser at localhost:8085/index?keyword=halima1. The page has a blue header 'Liste des Patients'. Below it is a search bar containing 'halima1' and a 'Rechercher' button. A table displays the results:

ID	Nom	Date de Naissance	Malade	Score
1	halima1	2024-04-20	false	35
4	halima1	2024-04-20	false	35
7	halima1	2024-04-20	false	35
10	halima1	2024-04-20	false	35

At the bottom right, there are pagination controls showing '0' and '1'.

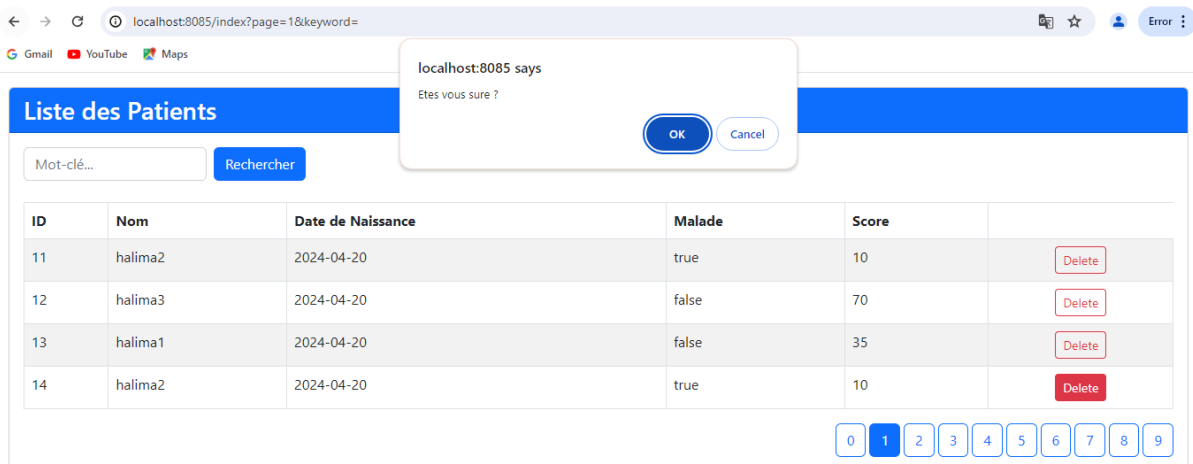
Dans le contrôleur **PatientController**, j'ai ajouté une méthode **delete** pour gérer la suppression d'un patient. Cette méthode prend en paramètre l'identifiant du patient à supprimer de la base de données, puis redirige l'utilisateur vers la liste des patients après la suppression.

```
@GetMapping("/{delete}")
public String delete(Long id){
    patientRepository.deleteById(id);
    return "redirect:/index";
}
```

Dans la vue patients.html, j'ai ajouté un bouton **delete** pour chaque patient. Ce bouton déclenche une fenêtre de confirmation JavaScript pour s'assurer que l'utilisateur souhaite vraiment supprimer le patient. En cliquant sur le bouton, une requête est envoyée à l'URL **"/delete"** avec l'identifiant du patient à supprimer.

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

Voici la page index après avoir cliqué sur le bouton **delete** correspondant à un patient. Le patient est supprimé de la base de données et la liste des patients est actualisée pour refléter cette modification.



J'ai amélioré le bouton delete dans la vue **patients.html** pour conserver le mot-clé de recherche et la page actuelle lors de la suppression d'un patient.

```
<td th:text="${p.dateNaissance}"></td><td th:text="${p.malade}"></td><td th:text="${p.score}"></td>
<td class="text-center"><a onclick="javascript:return confirm('Etes vous sure ?')";
th:href="@{delete(id=${p.id}, keyword=${keyword},page=${currentPage})}"
class="btn btn-outline-danger btn-sm">
delete </a>
</td>
```

Pour correspondre à cette modification, j'ai ajouté les paramètres "keyword" et "page" à la méthode **delete** du contrôleur **PatientController**. Ainsi, lors de la suppression d'un patient, ces paramètres sont pris en compte pour rediriger l'utilisateur vers la même page avec les mêmes paramètres de recherche.

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

J'ai ajouté les dépendances **Bootstrap Icons** et **jQuery** dans le fichier pom.xml pour faciliter l'utilisation des icônes Bootstrap et des fonctionnalités jQuery dans mon application.

```
<!-- https://mvnrepository.com/artifact/org.webjars/jquery -->
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>jquery</artifactId>
  <version>3.7.1</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.webjars.npm/bootstrap-icons -->
<dependency>
  <groupId>org.webjars.npm</groupId>
  <artifactId>bootstrap-icons</artifactId>
  <version>1.11.3</version>
</dependency>
```

Ensuite, j'ai ajouté les **liens** vers les fichiers Bootstrap et jQuery, hébergés localement dans les répertoires Webjars, dans la section head de la vue patients.html.

```
<meta charset="UTF-8"><title>Patients</title>
<link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
<script src="/webjars/jquery/3.7.1/jquery.min.js"></script>
<link rel="stylesheet" href="/webjars/bootstrap-icons/1.11.3/font/bootstrap-icons.css">
```

Ensuite, j'ai ajouté une **icône de loupe** pour représenter la fonctionnalité de recherche par mot-clé dans la vue patients.html.

```
<div class="col-auto">
  <button type="submit" class="btn btn-primary"><i class="bi bi-search"></i></button>
</div>
```






De plus, j'ai remplacé le texte **Delete** par l'**icône de la corbeille** dans le bouton delete de la vue patients.html. Cela améliore l'esthétique de l'interface utilisateur et rend la suppression des patients plus intuitive.

```
<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 class="text-center"><a onclick="javascript:return confirm('Etes vous sure ?')"
  th:href="@{delete(id=${p.id}, keyword=${keyword},page=${currentPage})}"
  class="btn btn-danger btn-sm">
  <i class="bi bi-trash"></i></a>
</td>
</tr>
```

Et Après, j'ai ajusté la numérotation des pages pour qu'elle commence par 1 au lieu de 0 dans la vue patients.html.

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

Voici la page après avoir apporté toutes les améliorations mentionnées.

Liste des Patients					
Mot-clé...					
ID	Nom	Date de Naissance	Malade	Score	
11	halima2	2024-04-20	true	10	
12	halima3	2024-04-20	false	70	
13	halima1	2024-04-20	false	35	
14	halima2	2024-04-20	true	10	
<div>12345</div>					

Partie 2 :

Ensuite, j'ai ajouté le dialecte **Thymeleaf Layout Dialect** dans le fichier pom.xml pour faciliter la gestion des mises en page (layouts) dans l'application Thymeleaf. Cela permet d'organiser et de réutiliser efficacement les éléments de mise en page, améliorant ainsi la maintenabilité et la lisibilité du code HTML.

```
<!-- https://mvnrepository.com/artifact/nz.net.ultraq.thymeleaf/thymeleaf-layout-dialect -->
<dependency>
  <groupId>nz.net.ultraq.thymeleaf</groupId>
  <artifactId>thymeleaf-layout-dialect</artifactId>
  <version>3.3.0</version>
</dependency>
```

J'ai ensuite créé un modèle de barre de navigation qui inclut le nom 'hôpital' et trois liens. Le dernier lien est un menu déroulant Patients qui contient deux options : "Nouveau" et "Rechercher". Du côté droit de la barre de navigation, j'ai ajouté le nom d'utilisateur avec un menu déroulant contenant l'option "Logout".

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org" xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
  <meta charset="UTF-8">
  <title>Template1</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-md navbar-light bg-primary">
    <div class="container-fluid">
      <a class="navbar-brand" href="#" style="color: white; font-size: 1.2rem;"><h3>HOSPITAL</h3></a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav"
        aria-expanded="false" aria-label="Toggle navigation"><span class="navbar-toggler-icon"></span></button>
      <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav me-auto">
          <li class="nav-item"><a class="nav-link active" aria-current="page" href="#" style="color: white; font-size: 1.1rem;">Accueil</a></li>
          <li class="nav-item"><a class="nav-link" href="#" style="color: white; font-size: 1.1rem;">Link</a></li>
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown"
              aria-expanded="false" style="color: white; font-size: 1.1rem;">Patients</a>
            <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
              <li><a class="dropdown-item" th:href="@{/formPatients}" style="color: black;">Nouveau</a></li>
              <li><a class="dropdown-item" th:href="@{/index}" style="color: black;">Chercher</a></li>
            </ul>
          </li>
        </ul>
      </div>
    </div>
  </nav>
```

```

<ul class="navbar-nav ms-auto">
  <li class="nav-item dropdown">
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown2" role="button" data-bs-toggle="dropdown"
      aria-expanded="false" style="color: white; font-size: 1.1rem;">[Username]</a>
    <ul class="dropdown-menu" aria-labelledby="navbarDropdown2">
      <li><a class="dropdown-item" th:href="@{/logout}" style="color: black;">logout</a></li>
    </ul>
  </li>
</ul>
</div>
</nav>
<section layout:fragment="content1">
</section>
</body>
</html>

```

Ensuite, j'ai inséré le modèle de **barre de navigation** dans la page patients.html en utilisant le fragment "content1".

```

<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
  xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
  layout:decorate="template1">
<head>
  <meta charset="UTF-8"><title>Patients</title>
  <link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
  <script src="/webjars/jquery/3.7.1/jquery.min.js"></script>
  <link rel="stylesheet" href="/webjars/bootstrap-icons/1.11.3/font/bootstrap-icons.css">
</head>
<body>
  <div layout:fragment="content1">
    <div class="mt-4 p-3"><div class="card">

```

Voici la page patients avec la barre de navigation incluse.

The screenshot shows a web application interface for a hospital. The header is blue with the text 'HOSPITAL' and navigation links: 'Accueil', 'Link', and 'Patients'. A dropdown menu is open under 'Patients', showing 'Nouveau' and 'Chercher'. The main content area is titled 'Liste des Patients' and features a search bar with the placeholder 'Mot-clé...'. Below the search bar is a table with the following data:

ID	Nom	Date de Naissance	Malade	Score	
126	halima3	2024-04-20	false	70	
127	halima1	2024-04-20	false	35	
129	halima3	2024-04-20	false	70	
142	halima1	2024-04-20	false	35	

At the bottom right of the table, there are pagination controls showing numbers 1 through 6, with 5 being the active page.

j'ai ajouté une méthode **formPatients** dans le contrôleur **PatientController** pour gérer les requêtes GET vers l'URL `/formPatients`. Cette méthode prépare un objet Patient vide dans le modèle, permettant ainsi de saisir les informations d'un nouveau patient dans un formulaire. Ensuite, elle retourne le nom de la vue `formPatients`.

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

Dans la vue **formPatients.html**, j'ai créé un formulaire permettant de saisir les informations d'un nouveau patient. J'ai utilisé Thymeleaf pour lier les champs du formulaire aux propriétés de l'objet "patient" et afficher les messages d'erreur de validation à côté des champs. Le formulaire soumet les données à un endpoint spécifié lorsqu'il est soumis.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="template1">
<head>
    <meta charset="UTF-8"><title>Form Patient</title>
    <link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
    <script src="/webjars/jquery/3.7.1/jquery.min.js"></script>
    <link rel="stylesheet" href="/webjars/bootstrap-icons/1.11.3/font/bootstrap-icons.css">
    <style> body {background-color: #f8f9fa;} </style>
</head>
<body>
<div layout:fragment="content1">
    <div class="container mt-4">
        <div class="row justify-content-center">
            <div class="col-lg-6">
                <div class="card shadow border-primary">
                    <div class="card-header bg-light text-primary text-center">
                        <h3 class="mb-0">Nouveau Patient</h3>
                    </div>
                    <div class="card-body">
                        <form method="post" th:action="@{save}">
                            <div class="mb-3">
                                <label for="nom" class="form-label">Nom</label>
                                <input type="text" class="form-control" id="nom" name="nom" th:value="{patient.nom}">
                                <span class="text-danger" th:errors="{patient.nom}"></span>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

```

</div>
<div class="mb-3">
  <label for="dateNaissance" class="form-label">Date de naissance</label>
  <input type="date" class="form-control" id="dateNaissance" name="dateNaissance" th:value="${patient.dateNaissance}" />
  <span class="text-danger" th:errors="${patient.dateNaissance}"></span>
</div>
<div class="mb-3">
  <label for="malade" class="form-label">Malade</label>
  <select class="form-select" id="malade" name="malade">
    <option th:selected="${patient.malade}" value="true">Oui</option>
    <option th:selected="${!patient.malade}" value="false">Non</option>
  </select>
  <span class="text-danger" th:errors="${patient.malade}"></span>
</div>
<div class="mb-3">
  <label for="score" class="form-label">Score</label>
  <input type="number" class="form-control" id="score" name="score" th:value="${patient.score}" />
  <span class="text-danger" th:errors="${patient.score}"></span>
</div>
<div class="d-grid">
  <button type="submit" class="btn btn-primary">Enregistrer</button>
</div>
</form></div></div></div></div>
</div>
</body>
</html>

```

En accédant à `/formPatients`, la page affiche un formulaire pour ajouter un nouveau patient, tout en incluant la barre de navigation pour une expérience utilisateur cohérente.



J'ai ajouté la dépendance **Spring Boot Starter Validation** au fichier pom.xml pour intégrer la validation des données dans l'application

```

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

```


Dans la classe **Patient**, j'ai ajouté des annotations de validation **@NotEmpty** et **@Size** pour le nom, et **@DecimalMin** pour le score, assurant ainsi que ces champs respectent les contraintes de longueur et de valeur minimale.

```
package fs.miaad.entities;
import ...
14 usages
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Patient {
    ⚡ @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotEmpty
    @Size(min = 4 , max = 20)
    private String nom;
    @Temporal(TemporalType.DATE) // prend la date sans heure
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date dateNaissance;
    private boolean malade;
    @DecimalMin("30")
    private int score;
}
```

J'ai ajouté une méthode **save** au contrôleur **PatientController** pour sauvegarder les données d'un patient. Elle valide les données avec **@Valid**, gère les erreurs avec **BindingResult**, puis sauvegarde le patient et redirige vers la page d'accueil.

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

Si une erreur survient lors de la validation, un message d'erreur s'affiche sous le champ correspondant dans le formulaire. Sinon, les données du patient sont sauvegardées et l'utilisateur est redirigé vers la page d'accueil (/index).









Nom

size must be between 4 and 20
must not be empty

Dans la vue **patients.html**, j'ai ajouté un bouton **Edit** à côté de chaque patient, en passant les paramètres d'identifiant du patient, de la page actuelle et du mot-clé de recherche dans l'URL.

```
<td class="text-center">
  <a onclick="javascript:return('ETES VOUS SURE?')"
    th:href="@{delete(id=${p.id}, keyword=${keyword},page=${currentPage})}"
    class="btn btn-danger btn-sm">
    <i class="bi bi-trash"></i>
  </a>
  <a th:href="@{editPatient(id=${p.id}, keyword=${keyword},page=${currentPage})}"
    class="btn btn-primary btn-sm">
    <i class="bi bi-pencil"></i>
  </a>
</td>
```

Voici la liste des patients avec les boutons **Edit** et **Delete** ajoutés à côté de chaque patient.

ID	Nom	Date de Naissance	Malade	Score	
4	halima1	2024-04-20	false	35	 
7	halima1	2024-04-20	false	35	 
8	halima2	2024-04-20	true	10	 
9	halima3	2024-04-20	false	70	 

J'ai créé une page **editPatient.html** qui permet de modifier les informations d'un patient. Cette page affiche un formulaire pré-rempli avec les données du patient sélectionné, et l'utilisateur peut soumettre les modifications en cliquant sur le bouton "Modifier".

```
<div layout:fragment="content1">
  <div class="container mt-4">
    <div class="row justify-content-center">
      <div class="col-lg-6">
        <div class="card shadow border-primary">
          <div class="card-header bg-light text-primary text-center">
            <h3 class="mb-0">Modifier Patient</h3>
          </div>
          <div class="card-body">
            <form method="post" th:action="@{save(page=${page},keyword=${keyword})}">
              <div class="mb-3">
                <label for="id" th:text="${patient.id}" class="form-label">ID</label>
                <input class="form-control" id="id" type="hidden" name="id" th:value="${patient.id}">
              </div>
              <div class="mb-3">
                <label for="nom" class="form-label">Nom</label>
                <input type="text" class="form-control" id="nom" name="nom" th:value="${patient.nom}">
                <span class="text-danger" th:errors="${patient.nom}"></span>
              </div>
              <div class="mb-3">
                <label for="dateNaissance" class="form-label">Date de naissance</label>
                <input type="date" class="form-control" id="dateNaissance" name="dateNaissance" th:value="${patient.dateNaissance}">
                <span class="text-danger" th:errors="${patient.dateNaissance}"></span>
              </div>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>
```

```

<div class="mb-3">
  <label for="malade" class="form-label">Malade</label>
  <select class="form-select" id="malade" name="malade">
    <option th:selected="${patient.malade}" value="true">Oui</option>
    <option th:selected="${!patient.malade}" value="false">Non</option>
  </select>
  <span class="text-danger" th:errors="${patient.malade}"></span>
</div>
<div class="mb-3">
  <label for="score" class="form-label">Score</label>
  <input type="number" class="form-control" id="score" name="score" th:value="${patient.score}">
  <span class="text-danger" th:errors="${patient.score}"></span>
</div>
<div class="d-grid">
  <button type="submit" class="btn btn-primary">Modifier</button>
</div>
</form></div></div></div></div></div>
</div>
</body>
</html>

```

Au contrôleur, j'ai ajouté une méthode POST `/save` pour sauvegarder les modifications du patient, redirigeant ensuite vers la page d'accueil avec les paramètres de pagination et de mot-clé. De plus, une méthode GET `/editPatient` récupère les informations du patient pour afficher la page de modification avec les paramètres de recherche conservés.

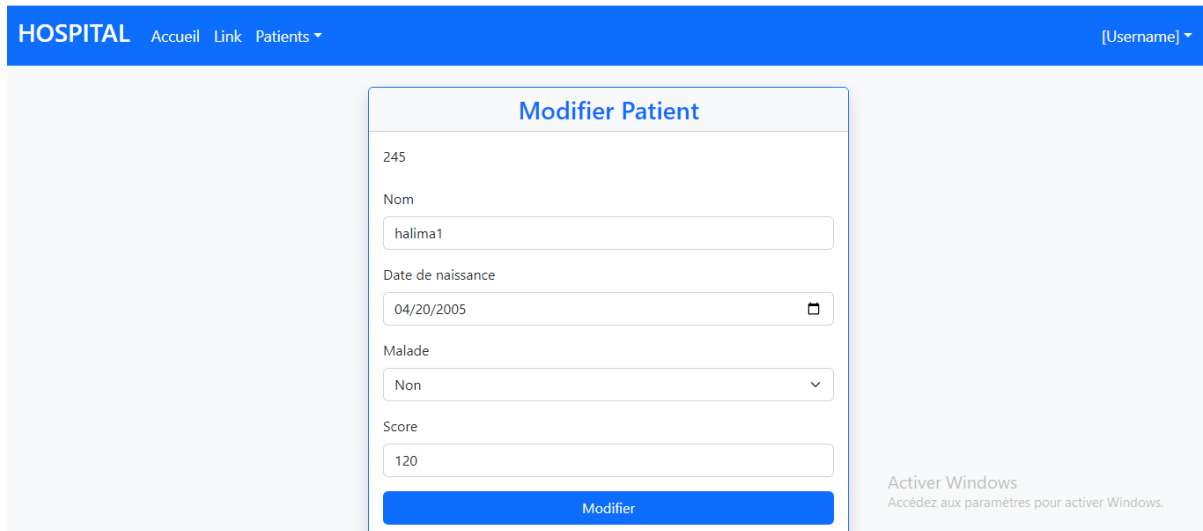
```

@PostMapping("/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;
}

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

```

Lorsque je clique sur le **boutton edit** pour un patient spécifique, la page de modification editPatient.html s'affiche avec un formulaire pré-rempli contenant toutes les données du patient sélectionné, me permettant ainsi de modifier facilement ces informations.



The screenshot shows the 'Modifier Patient' form in the HOSPITAL application. The form is pre-filled with the following data:

- ID: 245
- Nom: halima1
- Date de naissance: 04/20/2005
- Malade: Non
- Score: 120

A blue 'Modifier' button is located at the bottom of the form. The application header shows 'HOSPITAL' with links to 'Accueil', 'Link', and 'Patients', and a user profile dropdown labeled '[Username]'.

Après avoir modifié les champs et cliqué sur **Modifier**, la page redirige vers la liste des patients avec la même page et le même mot-clé déjà sélectionnés, maintenant mis à jour avec les modifications apportées.



The screenshot shows the 'Liste des Patients' page in the HOSPITAL application. The search bar contains 'halima1'. The table below displays the patient data:

ID	Nom	Date de Naissance	Malade	Score	
245	halima1	2003-03-31	true	195	 

At the bottom right of the table, there are pagination controls showing '1', '2', and '3'.

Partie 3 :

1. InMemory Authentication

J'ai ajouté la dépendance **spring-boot-starter-security** au fichier pom.xml pour intégrer Spring Security à l'application

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

Dans la classe **Tp3DaoudiHalimaApplication**, j'ai configuré un bean **passwordEncoder()** qui retourne un **BCryptPasswordEncoder** pour sécuriser les mots de passe stockés dans l'application.

```
@Bean
PasswordEncoder passwordEncoder(){
    return new BCryptPasswordEncoder();
}
```

Dans cette classe, j'utilise **Spring Security** pour gérer l'authentification et l'autorisation dans mon application. J'instancie un gestionnaire de détails d'utilisateurs en mémoire avec des utilisateurs statiques et leurs rôles. Ensuite, je configure les règles d'autorisation pour les différentes URL de l'application, en permettant l'accès à certaines ressources publiques et en exigeant l'authentification pour les autres. Enfin, je gère les redirections vers la page de connexion et la page d'accès non autorisé en cas d'erreur.

```
package fs.miaad.security;
import ...
@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: "12345")).roles("USER").build(),
            User.withUsername("user2").password(passwordEncoder.encode(rawPassword: "12345")).roles("USER").build(),
            User.withUsername("user3").password(passwordEncoder.encode(rawPassword: "12345")).roles("USER").build(),
            User.withUsername("admin").password(passwordEncoder.encode(rawPassword: "12345")).roles("USER", "ADMIN").build()
        );
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.formLogin().loginPage("/Login").permitAll();
        httpSecurity.authorizeHttpRequests().requestMatchers("/webjars/**").permitAll();
        httpSecurity.rememberMe();
        //httpSecurity.authorizeHttpRequests().requestMatchers("/user/**").hasRole("USER");
        //httpSecurity.authorizeHttpRequests().requestMatchers("/admin/**").hasRole("ADMIN");
        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        httpSecurity.exceptionHandling().accessDeniedPage(accessDeniedUrl: "/notAuthorized");
        return httpSecurity.build();
    }
}
```

J'ajoute la dépendance **Thymeleaf Extras Spring Security** pour intégrer des fonctionnalités de sécurité dans mes pages Thymeleaf, telles que l'accès conditionnel basé sur les rôles utilisateur.

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
    <version>3.1.2.RELEASE</version>
</dependency>
```

Dans le contrôleur **SecurityController**, deux méthodes GET sont définies pour gérer les chemins d'accès **/notAuthorized** et **/login**. Lorsqu'un utilisateur accède à **/notAuthorized**, la vue **notAuthorized** est retournée, et lorsqu'il accède à **/login**, la vue **login** est renvoyée.

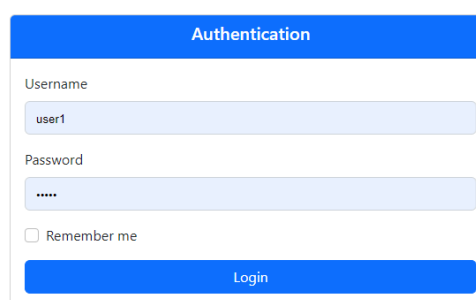
```
@Controller
public class SecurityController {
    @GetMapping("/notAuthorized")
    public String notAuthorized(){
        return "notAuthorized";
    }

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

J'ai créé la page de connexion qui contient un formulaire avec deux champs : Nom d'utilisateur et Mot de passe, ainsi qu'une case à cocher Se souvenir de moi et un bouton Connexion pour soumettre le formulaire.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Authentication</title>
    <link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
    <link rel="stylesheet" href="/webjars/bootstrap-icons/1.11.3/font/bootstrap-icons.css">
</head>
<body>
    <div class="container mt-5"><div class="row justify-content-center"><div class="col-md-6">
        <div class="card">
            <div class="card-header bg-primary text-white text-center"><h5 class="card-title">Authentication</h5></div>
            <div class="card-body"><form method="post" th:action="@{/login}">
                <div class="mb-3"><label for="username" class="form-label">Username</label>
                <input type="text" class="form-control" id="username" placeholder="Username" name="username">
                </div>
                <div class="mb-3"><label for="password" class="form-label">Password</label>
                <input type="password" class="form-control" id="password" placeholder="Password" name="password"></div>
                <div class="mb-3 form-check"><input type="checkbox" class="form-check-input" id="remember-me" name="remember-me">
                <label class="form-check-label" for="remember-me">Remember me</label></div>
                <div class="d-grid"><button type="submit" class="btn btn-primary">Login</button></div>
            </form></div></div></div>
        </div>
    </body>
</html>
```

Voici la page de formulaire de connexion lorsque j'accède à mon application.



The screenshot shows a web browser displaying the 'Authentication' login page. The page has a blue header with the title 'Authentication'. Below the header, there is a form with the following elements: a 'Username' label followed by a text input field containing 'user1'; a 'Password' label followed by a password input field with masked characters '.....'; a 'Remember me' checkbox which is currently unchecked; and a blue 'Login' button at the bottom of the form.

Après, j'ajoute le nom de l'utilisateur connecté sur la barre de navigation et un bouton de déconnexion qui permet de se déconnecter directement en quittant la session.

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown2" role="button" data-bs-toggle="dropdown"
    aria-expanded="false" style="color: white; font-size: 1.1rem;" th:text="${#authentication.name}"></a>
  <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
    <li>
      <form method="post" th:action="@{/logout}">
        <button class="dropdown-item" type="submit">Logout</button>
      </form>
    </li>
  </ul>
</li>
```

Après la connexion, le nom de l'utilisateur connecté s'affiche dans la barre de navigation.

HOSPITAL Accueil Link Patients ▾

user1 ▾

Après cela, j'ai ajouté des annotations de sécurité (@**PreAuthorize**) pour attribuer les rôles utilisateur et administrateur aux différentes méthodes du contrôleur Patient. Ces annotations garantissent que seuls les utilisateurs ayant les autorisations appropriées peuvent accéder à ces fonctionnalités, telles que la suppression, l'ajout et la modification de patients.

```
@GetMapping(Ⓜ"/user/index")
public String index(Model model ,@RequestParam(name="page" , defaultValue = "0") int page ,
    @RequestParam(name="size" , defaultValue = "4") int size,
    @RequestParam(name="keyword" , defaultValue = "") String kw){
    Page<Patient> patientPage =patientRepository.findByNomContains(kw,PageRequest.of(page,size));
    model.addAttribute( attributeName: "listPatients", patientPage.getContent());
    model.addAttribute( attributeName: "pages", new int[patientPage.getTotalPages()]);
    model.addAttribute( attributeName: "currentPage" , page);
    model.addAttribute( attributeName: "keyword", kw);
    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(Ⓜ"/")
public String home(){
    return "redirect:/user/index";
}
@GetMapping(Ⓜ"/admin/formPatients")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String formPatients(Model model){
    model.addAttribute(attributeName: "patient", new Patient());
    return "formPatients";
}
@PostMapping(Ⓜ"/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;
}

```

```

@GetMapping(Ⓜ"/admin/editPatient")
@PreAuthorize("hasRole('ROLE_ADMIN')")
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 "editPatient";
}

```

Après avoir attribué les rôles utilisateur et administrateur aux méthodes du contrôleur Patient, j'ai ensuite mis à jour les liens dans les pages HTML pour diriger vers les URL appropriées, en fonction des rôles.

```

<a th:href="@{/admin/editPatient(id=${p.id},keyword=${keyword},page=${currentPage})}"
    class="btn btn-primary btn-sm">
    <i class="bi bi-pencil"></i>
</a>

```


Pour masquer les boutons **Edit** et **Delete** pour les utilisateurs avec le rôle "USER", j'ai utilisé une expression Thymeleaf conditionnelle qui vérifie si l'utilisateur a le rôle "ADMIN". Cela garantit que seuls les utilisateurs ayant ce privilège peuvent voir et utiliser ces boutons.

```
<td class="text-center" th:if="${#authorization.expression('hasRole('ADMIN'))}">
  <a onclick="javascript:return confirm('ETES VOUS SURE?')"
    th:href="@{/admin/delete(id=${p.id},keyword=${keyword},page=${currentPage})}"
    class="btn btn-danger btn-sm">
    <i class="bi bi-trash"></i>
  </a>
  <a th:href="@{/admin/editPatient(id=${p.id},keyword=${keyword},page=${currentPage})}"
    class="btn btn-primary btn-sm">
    <i class="bi bi-pencil"></i>
  </a>
</td>
```

Lorsque l'utilisateur connecté n'a pas le rôle d'administrateur mais seulement celui d'utilisateur, la liste est affichée sans les boutons Edit et Delete. Cela permet de limiter les actions disponibles aux utilisateurs avec des privilèges spécifiques.

Liste des Patients				
<input type="text" value="Mot-clé..."/> <input type="button" value="Q"/>				
ID	Nom	Date de Naissance	Malade	Score
4	halima1	2024-04-20	false	35
7	halima1	2024-04-20	false	35
8	halima2	2024-04-20	true	10
9	halima3	2024-04-20	false	70

J'ai créé la page notAuthorized.html qui contient un message d'erreur indiquant que l'utilisateur n'est pas autorisé à accéder à cette page.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
  xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
  layout:decorate="template1">
<head>
  <meta charset="UTF-8">
  <title>notAuthorized</title>
  <link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
  <link rel="stylesheet" href="/webjars/bootstrap-icons/1.11.3/font/bootstrap-icons.css">
</head>
<body>
<div layout:fragment="content1">
  <div class="alert alert-danger m-3 mt-5" role="alert">
    <strong>Error:</strong> No authorized to access this page!
  </div>
</div>
</body>
</html>
```

Si un utilisateur connecté n'est pas administrateur et tente d'accéder à une fonctionnalité qu'il n'est pas autorisé à utiliser, comme l'ajout d'un nouveau patient, la page "Notauthorized.html" s'affiche pour lui indiquer qu'il n'est pas autorisé à accéder à cette fonctionnalité.



J'ajoute cette configuration pour rediriger les utilisateurs vers la page d'accueil après une connexion réussie et permettre l'accès à la page de connexion à tous les utilisateurs, authentifiés ou non.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.formLogin().loginPage("/login").defaultSuccessUrl("/").permitAll();
    httpSecurity.authorizeHttpRequests().requestMatchers("/webjars/**").permitAll();
    httpSecurity.rememberMe();
    httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
    httpSecurity.exceptionHandling().accessDeniedPage(accessDeniedUrl: "/notAuthorized");
    return httpSecurity.build();
}
```

2. JDBC Authentication

J'ajoute également cette configuration en **securityconfig** pour définir un gestionnaire d'utilisateurs basé sur JDBC, permettant de gérer les détails des utilisateurs en utilisant une source de données.

```
@Bean
public JdbcUserDetailsManager jdbcUserDetailsManager(DataSource dataSource)
{
    return new JdbcUserDetailsManager(dataSource);
}
```

J'ai créé un fichier **schema.sql** pour définir la structure de la base de données. Il crée les tables users et authorities pour stocker les informations des utilisateurs et leurs rôles, ainsi qu'un index unique sur les colonnes username et authority de la table authorities.

```
create table if not exists users(username varchar(50) not null primary key,
    password varchar(500) not null,enabled boolean not null);
create table if not exists authorities (username varchar(50) not null,authority varchar(50) not null,
    constraint fk_authorities_users foreign key(username) references users(username));
create unique index if not exists ix_auth_username on authorities (username,authority);
```

J'ajoute ces configurations à **application.properties** pour différer l'initialisation de la source de données JPA et exécuter automatiquement les scripts SQL à chaque démarrage de l'application.

```
spring.application.name=TP3-DAOUDI-HALIMA
server.port=8085
#Configuration pour MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/db_patients?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode=always
```

J'ajoute un nouveau fichier **data.sql** pour insérer des données initiales dans la base de données. Ce fichier insère un utilisateur nommé "halimadaoudi" avec le mot de passe "0000" et le statut activé.

```
INSERT INTO `users` (`username`, `password`, `enabled`) VALUES ('HalimaDAOUDI', '0000', '1');
```

Voici les tables que j'ai créées dans la base de données : **users** pour stocker les informations des utilisateurs, **authorities** pour gérer leurs rôles et permissions, et **patient** pour les données spécifiques aux patients

Table	Action
<input type="checkbox"/> authorities	Parcourir Structure Rechercher Insérer Vider Supprimer
<input type="checkbox"/> patient	Parcourir Structure Rechercher Insérer Vider Supprimer
<input type="checkbox"/> users	Parcourir Structure Rechercher Insérer Vider Supprimer
3 tables	Somme

J'ai inséré un utilisateur nommé "halimadaoudi" avec le mot de passe "0000" et le statut activé dans la table users.

	username	password	enabled
<input type="checkbox"/> Éditer Copier Supprimer	HalimaDAOUDI	0000	1

J'ai supprimé le fichier **data.sql** et ajouté un **commandlinerunner** à l'application pour initialiser la base de données avec trois utilisateurs (user1, user2, et admin). Si ces utilisateurs n'existent pas encore, ils seront créés avec un mot de passe encodé et des rôles appropriés.

```
@Bean
CommandLineRunner commandLineRunner(JdbcUserDetailsManager jdbcUserDetailsManager, PasswordEncoder passwordEncoder)
{
    return args -> {
        UserDetails u1 = jdbcUserDetailsManager.loadUserByUsername("user1");
        if (u1 == null) {
            jdbcUserDetailsManager.createUser(User.withUsername("user1")
                .password(passwordEncoder.encode("0000"))
                .roles("USER").build());
        }
        UserDetails u2 = jdbcUserDetailsManager.loadUserByUsername("user2");
        if (u2 == null) {
            jdbcUserDetailsManager.createUser(User.withUsername("user2")
                .password(passwordEncoder.encode("0000"))
                .roles("USER").build());
        }
        UserDetails u3 = jdbcUserDetailsManager.loadUserByUsername("admin");
        if (u3 == null) {
            jdbcUserDetailsManager.createUser(User.withUsername("admin")
                .password(passwordEncoder.encode("0000"))
                .roles("USER", "ADMIN").build());
        }
    };
}
```

Voici la démonstration de la création automatique des utilisateurs admin, user1, et user2 avec leurs mots de passe encodés et leur statut activé.

		username	password	enabled
<input type="checkbox"/>	Éditer	admin	\$2a\$10\$cFu.13a6q8QvkgloJUdn2Ow1RkmMqs12cAQOH5wT0TR...	1
<input type="checkbox"/>	Éditer	user1	\$2a\$10\$spjfMblu7tQfr1PQHg6m.usdjJU.VQWtLZPcKJgNYDJ...	1
<input type="checkbox"/>	Éditer	user2	\$2a\$10\$3EdpvMJPqU.cLHGfEIL5xuftrg1HBzQHCKZuLpvAaFY...	1

Voici les rôles assignés aux utilisateurs : admin possède les rôles ROLE_ADMIN et ROLE_USER, tandis que user1 et user2 ont le rôle ROLE_USER.

		username	authority
<input type="checkbox"/>	Éditer	admin	ROLE_ADMIN
<input type="checkbox"/>	Éditer	admin	ROLE_USER
<input type="checkbox"/>	Éditer	user1	ROLE_USER
<input type="checkbox"/>	Éditer	user2	ROLE_USER

3. Userdetails Service :

J'ai créé un repository pour les **entités** dans le package **security** et défini deux classes : AppRole et AppUser.

J'ai défini la classe **AppUser** pour représenter un utilisateur de l'application avec ses attributs et ses rôles, en utilisant les annotations JPA et Lombok pour faciliter la gestion des entités.

```
package fs.miaad.security.entities;

import ...
no usages
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class AppUser
{
    @Id
    private String userId;
    @Column(unique = true)
    private String username;
    private String password;
    private String email;
    @ManyToMany(fetch = FetchType.EAGER)
    private List<AppRole> roles;
}
```

J'ai défini la classe **AppRole** pour représenter les rôles des utilisateurs, avec un seul attribut role, en utilisant les annotations JPA et Lombok pour simplifier la gestion des entités.

```
package fs.miaad.security.entities;
import ...
1 usage
@Entity @Data
@NoArgsConstructor @AllArgsConstructor @Builder
public class AppRole
{
    @Id
    private String role;
}
```

J'ai créé le package repo dans le package security pour stocker les repositories.

J'ai créé l'interface **AppUserRepository** dans le package repo. Elle étend JpaRepository pour fournir des méthodes CRUD pour l'entité AppUser et inclut une méthode personnalisée findByusername pour rechercher un utilisateur par son nom d'utilisateur.

```
package fs.miaad.security.repo;

import fs.miaad.security.entities.AppUser;
import org.springframework.data.jpa.repository.JpaRepository;
no usages
public interface AppUserRepository extends JpaRepository<AppUser, String>
{
    no usages
    AppUser findByUsername(String username);
}
```

J'ai créé l'interface **AppRoleRepository** dans le package repo. Elle étend JpaRepository pour fournir des méthodes CRUD pour l'entité AppRole.

```
package fs.miaad.security.repo;
import fs.miaad.security.entities.AppRole;
import org.springframework.data.jpa.repository.JpaRepository;

no usages
public interface AppRoleRepository extends JpaRepository<AppRole,String> {
}
```

J'ai créé le package service et défini l'interface accountservice. Cette interface déclare des méthodes pour ajouter de nouveaux utilisateurs et rôles, assigner et retirer des rôles d'un utilisateur, et charger un utilisateur par son nom d'utilisateur.

```
package fs.miaad.security.service;
import ...
1 usage 1 implementation
public interface AccountService
{
    no usages 1 implementation
    AppUser addNewUser(String username,String password,String email, String confirmPassword);
    no usages 1 implementation
    AppRole addNewRole(String role);
    no usages 1 implementation
    void addRoleToUser(String username, String role);
    no usages 1 implementation
    void removeRoleFromUser(String username, String role);
    no usages 1 implementation
    AppUser loadUserByUsername(String username);
}
```

J'ai créé la classe **AccountServiceImpl** qui implémente l'interface AccountService. Cette classe est annotée avec @Service et @Transactional. Elle utilise les repositories AppUserRepository et AppRoleRepository ainsi qu'un encodeur de mots de passe pour ajouter un nouvel utilisateur. Si l'utilisateur existe déjà ou si les mots de passe ne correspondent pas, elle lance une exception

```
package fs.miaad.security.service;
import ...
@Service @Transactional @AllArgsConstructor
public class AccountServiceImpl implements AccountService
{
    private AppUserRepository appUserRepository;
    private AppRoleRepository appRoleRepository;
    private PasswordEncoder passwordEncoder;
    no usages
    @Override
    public AppUser addNewUser(String username, String password, String email, String confirmPassword)
    {
        AppUser appUser=appUserRepository.findByUsername(username);
        if(appUser!=null) throw new RuntimeException("ce utilisateur existe déjà");
        if(!password.equals(confirmPassword)) throw new RuntimeException("mot de passe incorrecte");
        appUser=AppUser.builder()
            .userId(UUID.randomUUID().toString())
            .username (username)
            .password(passwordEncoder.encode(password))
            .email(email)
            .build();
        return appUserRepository.save(appUser);
    }
}
```

Ensuite, j'ai implémenté les méthodes `addNewRole` et `addRoleToUser`. La méthode `addNewRole` ajoute un nouveau rôle à la base de données si celui-ci n'existe pas déjà. La méthode `addRoleToUser` assigne un rôle existant à un utilisateur en récupérant l'utilisateur et le rôle depuis les repositories, puis en ajoutant le rôle à la liste des rôles de l'utilisateur.

```
@Override
public AppRole addNewRole(String role)
{
    AppRole appRole = appRoleRepository.findById(role).orElse( other: null);
    if(appRole!=null) throw new RuntimeException("Ce rôle existe déjà");
    appRole = AppRole.builder()
        .role(role)
        .build();
    return appRoleRepository.save(appRole);
}

no usages
@Override
public void addRoleToUser(String username, String role)
{
    AppUser appUser = appUserRepository.findByUsername(username);
    AppRole appRole = appRoleRepository.findById(role).get();
    appUser.getRoles().add(appRole);
    //appUserRepository.save(appUser);
}
```

Et enfin, j'ai implémenté les méthodes `removeRoleFromUser` et `loadUserByUsername`. La méthode `removeRoleFromUser` retire un rôle assigné à un utilisateur en le récupérant depuis les repositories, puis en le supprimant de la liste des rôles de l'utilisateur. La méthode `loadUserByUsername` récupère un utilisateur par son nom d'utilisateur depuis le repository.

```
no usages
@Override
public void removeRoleFromUser(String username, String role)
{
    AppUser appUser = appUserRepository.findByUsername(username);
    AppRole appRole = appRoleRepository.findById(role).get();
    appUser.getRoles().remove(appRole);
}

no usages
@Override
public AppUser loadUserByUsername(String username)
{
    return appUserRepository.findByUsername(username); }
}
```

J'ai ajouté un **CommandLineRunner** pour initialiser les données utilisateur et rôle lors du démarrage de l'application, en utilisant **AccountService**. Ce bean crée les rôles USER et ADMIN, ajoute trois utilisateurs (user11, user22, et admin0), et assigne les rôles appropriés à ces utilisateurs.

```
@Bean
CommandLineRunner commandLineRunnerUserDetails(AccountService accountService)
{
    return args->{
        accountService.addNewRole("USER");
        accountService.addNewRole("ADMIN");

        accountService.addNewUser(username: "user11", password: "1111", email: "user11@gmail.com", confirmPassword: "1111");
        accountService.addNewUser(username: "user22", password: "1111", email: "user22@gmail.com", confirmPassword: "1111");
        accountService.addNewUser(username: "admin0", password: "1111", email: "admin0@gmail.com", confirmPassword: "1111");

        accountService.addRoleToUser(username: "user11", role: "USER");
        accountService.addRoleToUser(username: "user22", role: "USER");
        accountService.addRoleToUser(username: "admin0", role: "USER");
        accountService.addRoleToUser(username: "admin0", role: "ADMIN");
    };
}
```

J'ai ajouté `spring.jpa.hibernate.ddl-auto=update` dans **application.properties** pour que Hibernate mette à jour automatiquement le schéma de la base de données selon les entités JPA.

```
spring.jpa.hibernate.ddl-auto=update
```

J'ai créé la classe **UserdetailServiceImpl** qui implémente **UserDetailsService**. Cette classe utilise **AccountService** pour charger les détails de l'utilisateur par nom d'utilisateur, y compris les rôles, et lance une exception si l'utilisateur n'est pas trouvé.

```
package fs.miaad.security.service;
import ...

@Service @AllArgsConstructor
public class UserDetailServiceImpl implements UserDetailsService {
    private AccountService accountService;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        AppUser appUser = accountService.loadUserByUsername(username);
        if(appUser==null) throw new UsernameNotFoundException(String.format("utilisateur %s n'est pas trouvé", username));
        String[] roles = appUser.getRoles().stream().map(u->u.getRole()).toArray(String[]::new);
        UserDetails userDetails= User
            .withUsername(appUser.getUsername())
            .password(appUser.getPassword())
            .roles(roles).build();
        return userDetails;
    }
}
```

J'ai mis à jour la classe **SecurityConfig** pour inclure les détails des utilisateurs avec **UserdetailServiceImpl** et configuré l'accès à l'application avec une page de connexion personnalisée, la gestion des exceptions, et les autorisations pour les ressources statiques.

```
@Configuration @EnableWebSecurity @EnableMethodSecurity(prePostEnabled = true) @AllArgsConstructor
public class SecurityConfig {
    private PasswordEncoder passwordEncoder;
    private UserDetailServiceImpl userDetailServiceImpl;
}
```



```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.formLogin().loginPage("/login").defaultSuccessUrl("/").permitAll();
    httpSecurity.authorizeHttpRequests().requestMatchers("@*/webjars/**").permitAll();
    httpSecurity.rememberMe();
    httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
    httpSecurity.exceptionHandling().accessDeniedPage(accessDeniedUrl: "/notAuthorized");
    httpSecurity.userDetailsService(userDetailServiceImpl);

    return httpSecurity.build();
}

```

Voici la démonstration de l'association **app_user_roles** des utilisateurs avec leurs rôles dans la base de données. Les utilisateurs ont des rôles tels que **USER** et **ADMIN** correctement assignés selon la configuration.

app_user_user_id	roles_role
4329bdd7-dab0-4036-8fc3-d5a3ea04976a	USER
c7c79fe1-7319-49de-8b1d-61728b45ef69	USER
310c210c-8010-4733-9a4d-40b87748f3e4	USER
310c210c-8010-4733-9a4d-40b87748f3e4	ADMIN

Voici les utilisateurs créés dans la base de données, avec leurs identifiants uniques, noms d'utilisateur, adresses e-mail et mots de passe encodés.

	user_id	username	email	password
<input type="checkbox"/> Cliquer sur la flèche pour afficher/masquer la colonne	679af8c-f3d5-4500-8c-a193956ba9b5	user22	user22@gmail.com	\$2a\$10\$VQOuWxr8V7p6YkwbSgU3Uuqrg1BUE.SxBsNX0EYydsr...
<input type="checkbox"/> Éditer Copier Supprimer	24cbdc8f-ad51-4bc4-8ea6-3bf5eba1a4d3	user11	user11@gmail.com	\$2a\$10\$mUb5suih11LvS877DT38.OOYORqE9Zh5x3S.Am7C7Us...
<input type="checkbox"/> Éditer Copier Supprimer	a88c870b-42e7-4526-94ae-350e1b9321ab	admin0	admin0@gmail.com	\$2a\$10\$BkyTdWH47RyCw6zU6egtfOHEoNP5SyCbA8BIHLA/8W...

Je me suis connecté en tant qu'administrateur (admin0) et ai accédé à l'interface de l'application hospitalière.

HOSPITAL	Accueil	Link	Patients ▾	admin0 ▾
----------	---------	------	------------	----------

CONCLUSION

En Conclusion, ce TP a été l'occasion pour moi de concevoir et de mettre en œuvre une application Web JEE dédiée à la gestion des patients. Grâce à l'utilisation de Spring MVC, Thymeleaf et Spring Data JPA, j'ai développé des fonctionnalités essentielles telles que l'affichage des patients, leur pagination, la recherche et leur suppression.

L'intégration d'une validation de formulaire a permis d'améliorer l'interaction utilisateur, tandis que l'ajout de Spring Security a renforcé la sécurité de l'application. J'ai exploré trois méthodes d'authentification : InMemory Authentication, JDBC Authentication et UserDetails Service, assurant ainsi une gestion sécurisée et flexible des utilisateurs.

Ce projet m'a offert une opportunité précieuse de consolider mes compétences en développement Web JEE tout en produisant une solution fonctionnelle et sécurisée pour la gestion des patients