Université Sultan Moulay Slimane - Béni Mellal
École Nationale des Sciences Appliquées - Khouribga

JAVAEE

COMPTE RENDU

ATELIER 5

Réalisé par : *EL FATHI Zakaria*

Encadré par : *Pr. BAKKOURI Ibtissam*

SOMMAIRE

SOMMAIRE 1
INTRODUCTION 2
PARTIE 1 2
Fichier de configuration pom.xml 2
Le fichier application.properties 4
Structure MVC : 5
ETUDIANT.JAVA 5
repoEtudiant.java : 7
ControllerEtudiant.java 7
Formulaire etudiant 9
Affichage des Etudiants 10
Test de saisie et affichage 13
PARTIE 2 14
REINSCRIPTION.java 14
Reinscription controller : 16
Repository : 17
Formulaire de reinscription 18
Affichage des donnees saisies : 20
CONCLUSION 22

INTRODUCTION

Le but de cet atelier est la création d'une application de gestion d'inscription des étudiants en JEE, en utilisant le modèle MVC avec Spring Boot. Cette application permet aux utilisateurs de créer un étudiant en saisissant des données depuis un formulaire et de visualiser la fiche étudiant en résultant, ainsi que de créer une inscription en remplissant un autre formulaire et de visualiser la fiche inscription en résultant. Ce projet permet aux étudiants de développer leurs compétences en programmation Java et en conception d'applications web, tout en leur donnant une expérience concrète dans la création d'une application utile pour la gestion des données d'étudiants.

PARTIE 1

Fichier de configuration pom.xml

Dans ce cas, le fichier POM inclut des informations sur le parent du projet, les dépendances utilisées pour la gestion de la base de données (spring-boot-starter-data-jpa), la création de l'interface utilisateur (spring-boot-starter-thymeleaf et spring-boot-starter-web), et pour les tests (spring-boot-starter-test). Il inclut également une dépendance pour la connexion à la base de données MySQL et une autre pour la validation de formulaire avec Hibernate.
Le plugin spring-boot-maven-plugin est également inclus dans la configuration, ce qui permet de construire une application exécutable avec Spring Boot.
En résumé, ce fichier POM définit toutes les dépendances et les plugins nécessaires pour la création d'une application Spring Boot pour l'atelier 5 de l'Université Sultan Moulay Slimane ENSA-Khouribga.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.5</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.ensa</groupId>
```

```xml
<artifactId>TP5_SpringBoot</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>TP5_SpringBoot</name>
<description>Atelier 5 ensakh</description>
<properties>
    <java.version>17</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
    </dependency>
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.hibernate.validator</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>6.0.13.Final</version>
    </dependency>
</dependencies>
```

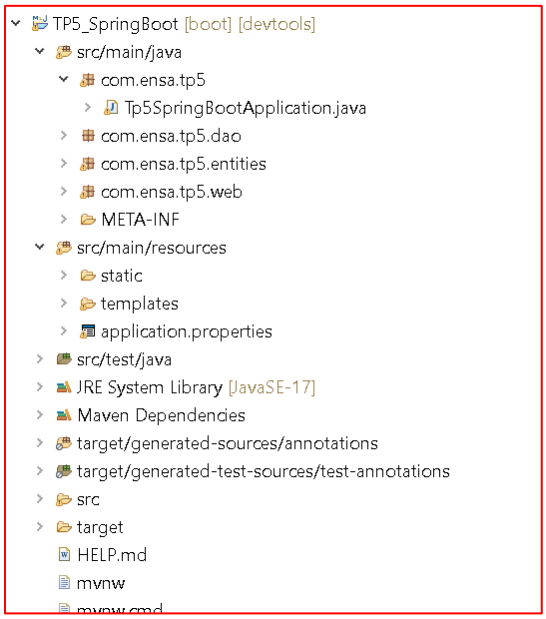
```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Le fichier application.properties

Ce fichier permet de configurer la connexion à une base de données MySQL en précisant l'URL de la base de données, le nom d'utilisateur et le mot de passe pour l'accès à la base de données.
La ligne server.port=8884 spécifie le port sur lequel le serveur Web sera exécuté.
La ligne spring.datasource.url indique l'URL de la base de données MySQL utilisée pour stocker les données de l'application. Le paramètre serverTimezone est ajouté pour éviter les problèmes liés aux fuseaux horaires.
Les paramètres spring.datasource.username et spring.datasource.password sont utilisés pour spécifier le nom d'utilisateur et le mot de passe de la base de données.
La ligne spring.datasource.driverClassName spécifie le nom de la classe du pilote JDBC utilisé pour la connexion à la base de données.
Les paramètres spring.jpa.show-sql et spring.jpa.hibernate.ddl-auto sont utilisés pour configurer la création automatique des tables de la base de données et pour afficher les requêtes SQL générées par Hibernate.
Enfin, la ligne spring.jpa.properties.hibernate.dialect indique le dialecte de la base de données utilisé par Hibernate, tandis que la ligne #spring.thymeleaf.cache=false est un commentaire indiquant que le cache Thymeleaf est désactivé pour faciliter le développement.

```
server.port=8884
spring.datasource.url=jdbc:mysql://localhost:3306/atelier5?serverTimezone=UTC
spring.datasource.username = root
spring.datasource.password =

spring.datasource.driverClassName = com.mysql.jdbc.Driver

spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
#spring.thymeleaf.cache=false
```

Structure MVC :



ETUDIANT.JAVA

définit l'entité d'un étudiant pour votre application Spring Boot. La classe est annotée avec "@Entity" pour indiquer qu'elle sera utilisée comme une entité persistante dans une base de données. Elle contient plusieurs attributs tels que le prénom, le nom de famille, l'adresse e-mail, le CNE (numéro national étudiant), la date de naissance et le numéro de téléphone de l'étudiant, qui sont tous définis à l'aide de getters et setters. La classe contient également un constructeur par défaut et un autre constructeur prenant en paramètres les attributs de l'étudiant.

```
package com.ensa.tp5.entities;
@Entity
public class Etudiant implements Serializable{
    @Id @GeneratedValue
    private Long id;

    private String firstName;
    private String lastName;
    private String email;
    @NotNull
    @Size(min=10,max=10)
    private String cne;
    private String dNaiss;
    @Size(min=10,max=10)
    private String tele;
    public Etudiant() {
        super();
```

```
    }
    public Etudiant(String firstName, String lastName, String email, String cne, String
dNaiss, String tele) {
        super();
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.cne = cne;
        this.dNaiss = dNaiss;
        this.tele = tele;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getCne() {
        return cne;
    }
    public void setCne(String cne) {
        this.cne = cne;
    }
    public String getdNaiss() {
        return dNaiss;
    }
    public void setdNaiss(String dNaiss) {
        this.dNaiss = dNaiss;
```

```
    }
    public String getTele() {
        return tele;
    }
    public void setTele(String tele) {
        this.tele = tele;
    } }
```

repoEtudiant.java :

une interface nommée "repoEtudiant" qui étend la classe "JpaRepository" de Spring Data JPA et qui permet de gérer les opérations CRUD (Create, Read, Update, Delete) sur l'entité "Etudiant".
La méthode "chercher" permet de chercher les étudiants dont le prénom contient une chaîne de caractères donnée en paramètre (:x). Cette méthode utilise une requête JPQL (@Query) qui sélectionne tous les étudiants dont le prénom contient la chaîne de caractères passée en paramètre. Le résultat est paginé (Page<Etudiant>) en utilisant un objet Pageable qui permet de définir la page et la taille de la pagination.
Notez que cette interface ne nécessite pas d'implémentation car Spring Data JPA se charge de générer dynamiquement le code pour les méthodes CRUD et de les associer aux requêtes JPQL/SQL appropriées.

```
package com.ensa.tp5.dao;

import org.springframework.data.domain.Page;
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;

import com.ensa.tp5.entities.Etudiant;

public interface repoEtudiant extends JpaRepository<Etudiant, Long> {
@Query("select e from Etudiant e where e.firstName like :x")
    public Page<Etudiant> chercher(@Param("x")String mc, Pageable pageable);
}
```

ControllerEtudiant.java

un contrôleur Spring MVC pour la gestion des entités Etudiant.

- La méthode index permet d'afficher la liste des étudiants paginée en utilisant la méthode chercher définie dans le repository repoEtudiant. Les paramètres p, s et mc permettent de spécifier respectivement la page courante, la taille de la page et la chaîne de caractères à rechercher.

- La méthode formEtudiant renvoie une vue contenant un formulaire permettant d'ajouter un nouvel étudiant.
- La méthode save est appelée lors de la soumission du formulaire et permet de sauvegarder l'étudiant ajouté en utilisant la méthode save définie dans le repository repoEtudiant. Si des erreurs de validation sont détectées, elle renvoie la vue form-part1.
- Le contrôleur utilise des vues définies dans des fichiers JSP (JavaServer Pages) pour générer du HTML à afficher dans le navigateur. Les noms des vues sont retournés en tant que chaînes de caractères à partir des méthodes du contrôleur.

```
package com.ensa.tp5.web;

@Controller
public class controllerEtudiant {
    @Autowired
    private repoEtudiant repEtudiant;

    @RequestMapping(value = "/index")
    public String index(Model model, @RequestParam(name = "page", defaultValue = "0") int
p,
            @RequestParam(name = "size", defaultValue = "6") int s,
            @RequestParam(name = "mc", defaultValue = "") String mc) {
        Page<Etudiant> pageEtudiants = repEtudiant.chercher("%" + mc + "%",
PageRequest.of(p, s));
        model.addAttribute("listEtudiants", pageEtudiants.getContent());
//      tableau de pages
        int[] pages = new int[pageEtudiants.getTotalPages()];
        model.addAttribute("pages", pages);
        model.addAttribute("size", s);
        model.addAttribute("currentPage", p);
        model.addAttribute("mc", mc);

        return "affichageEtudiants-part1";
    }

    @RequestMapping(value = "/form", method = RequestMethod.GET)
    public String formEtudiant(Model model) {
        model.addAttribute("etudiant", new Etudiant());

        return "form-part1";
    }

    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public String save(Model model, @Valid Etudiant etudiant, BindingResult bindingResult,
@RequestParam(name = "page", defaultValue = "0") int p,
            @RequestParam(name = "size", defaultValue = "5") int s,
```

```
            @RequestParam(name = "mc", defaultValue = "") String mc) {
        Page<Etudiant> pageEtudiants = repEtudiant.chercher("%" + mc + "%",
PageRequest.of(p, s));
        model.addAttribute("listEtudiants", pageEtudiants.getContent());
//      tableau de pages
        int[] pages = new int[pageEtudiants.getTotalPages()];
        model.addAttribute("pages", pages);
        model.addAttribute("size", s);
        model.addAttribute("currentPage", p);
        model.addAttribute("mc", mc);

        if (bindingResult.hasErrors())
            return "form-part1";
        repEtudiant.save(etudiant);
        return "ajoutEtudiantConfirmation";
    }}
```

Formulaire etudiant

"form-part1" contient le formulaire pour ajouter un nouvel étudiant. Les champs obligatoires sont le nom, le prénom, l'adresse e-mail, le CNE (Code National de l'Etudiant), la date de naissance et le numéro de téléphone.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
    <meta charset="utf-8" />
    <title>Ajouter Etudiant</title>
    <link rel="stylesheet" type="text/css" th:href="@{css/style.css}" />
    <link rel="stylesheet" type="text/css" th:href="@{bootstrap/bootstrap.min.css}" />

</head>

<body>
    <div class="login-box ">
        <h2>Ajouter Etudiant:</h2>
        <form method="post" th:action="@{save}">
            <div class="input-group">
                <div class="user-box">
                    <input type="text" class="form-control" th:value="${etudiant.firstName}" id="firstName"
                        name="firstName" placeholder="votre prenom" required/>
                </div>
                <div class="user-box">
                    <input type="text" class="form-control" th:value="${etudiant.lastName}" id="lastName"
```

```
                    name="lastName" placeholder="votre prenom" required/>
            </div>
            <div class="user-box">
                <input type="email" class="form-control" th:value="${etudiant.email}" id="email" name="email"
                    placeholder="votre email" required />
            </div>
            <div class="user-box">
                <input type="text" class="form-control" th:value="${etudiant.cne}" id="cne" name="cne"
                    placeholder="votre CNE" required />
                <span th:errors="${etudiant.cne}"></span>
            </div>
            <label class="user-box">Date de naissance:</label>
            <div class="user-box">
                <input type="date" th:value="${etudiant.dNaiss}" id="dNaiss" name="dNaiss" required/>
            </div>
            <label class="user-box">telephone:</label>
            <div class="user-box">
                <input type="tele" class="form-control" th:value="${etudiant.tele}" id="tele" name="tele"
                    placeholder="votre numero de telephone" required />
                <span th:errors=${etudiant.tele}></span>


            </div>
        </div>
        <button type="submit">save</button>
    </form>
</div>
</body>
</html>
```

Affichage des Etudiants

On affiche les informations d'un étudiant ainsi qu'une table contenant les informations de tous les étudiants. Il y a également un formulaire pour chercher des étudiants en utilisant un mot-clé. Le code utilise le moteur de template Thymeleaf pour insérer dynamiquement les données de l'étudiant et de la liste d'étudiants dans la page.

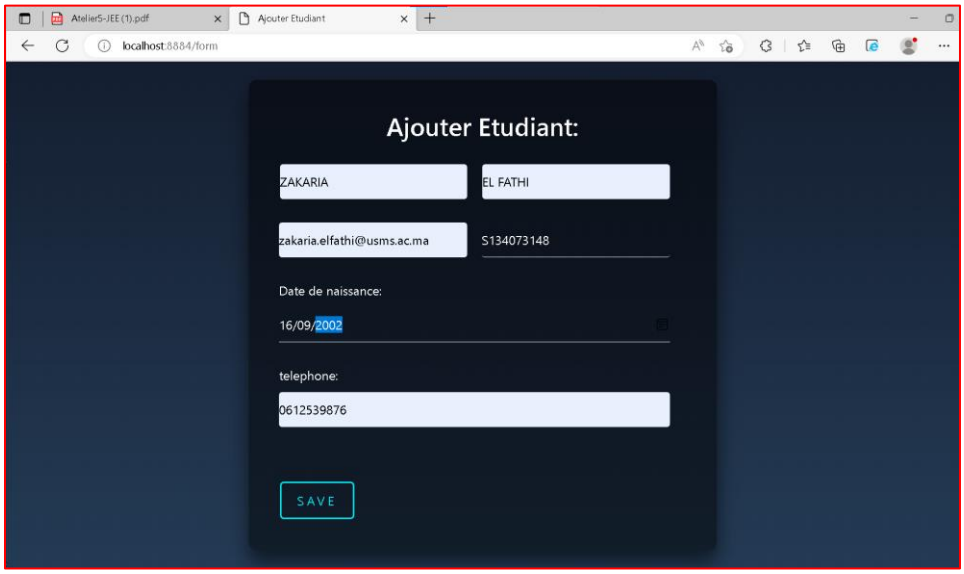
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
    <meta charset="utf-8" />
    <title>Etudiants Dashboard</title>
    <link rel="stylesheet" type="text/css" th:href="@{css/styleAffichage.css}" />
</head>

<body>
```

```
<div class="login-box">
        <h3>ETUDIANT QUE VOUS AVEZ SAISIE:</h3>
    <hr/>
    <div class="input-group">
        <div class="user-box">
            <h1><strong>PRENOM:</strong>
                <h3 th:text="${etudiant.firstName}"></h3>
            </h1>
        </div>
        <div class="user-box">
            <h1><strong>NOM:</strong>
                <h3 th:text="${etudiant.lastName}"></h3>
            </h1>
        </div>
        <div class="user-box">
            <h1><strong>EMAIL:</strong>
                <h3 th:text="${etudiant.email}"></h3>

            </h1>
        </div>
        <div class="user-box">
            <h1><strong>CNE:</strong>
                <h3 th:text="${etudiant.cne}"></h3>
            </h1>

        </div>
        <div class="user-box">
            <h1> <strong>DATE DE NAISSANCE:</strong>
                <h3 th:text="${etudiant.dNaiss}"></h3>

            </h1>
        </div>
        <div class="user-box">
            <h1><strong>TELEPHONE:</strong>
                <h3 th:text="${etudiant.tele}"></h3>

            </h1>
        </div>
                        <a  th:href="@{/form}">Ajouter un autre Etudiant </a>

    </div>
</div>
<div class="login-box">
    <h3>TABLE DES ETUDIANTS:</h3>       <hr/>

    <table class="input-group " border="1">
```

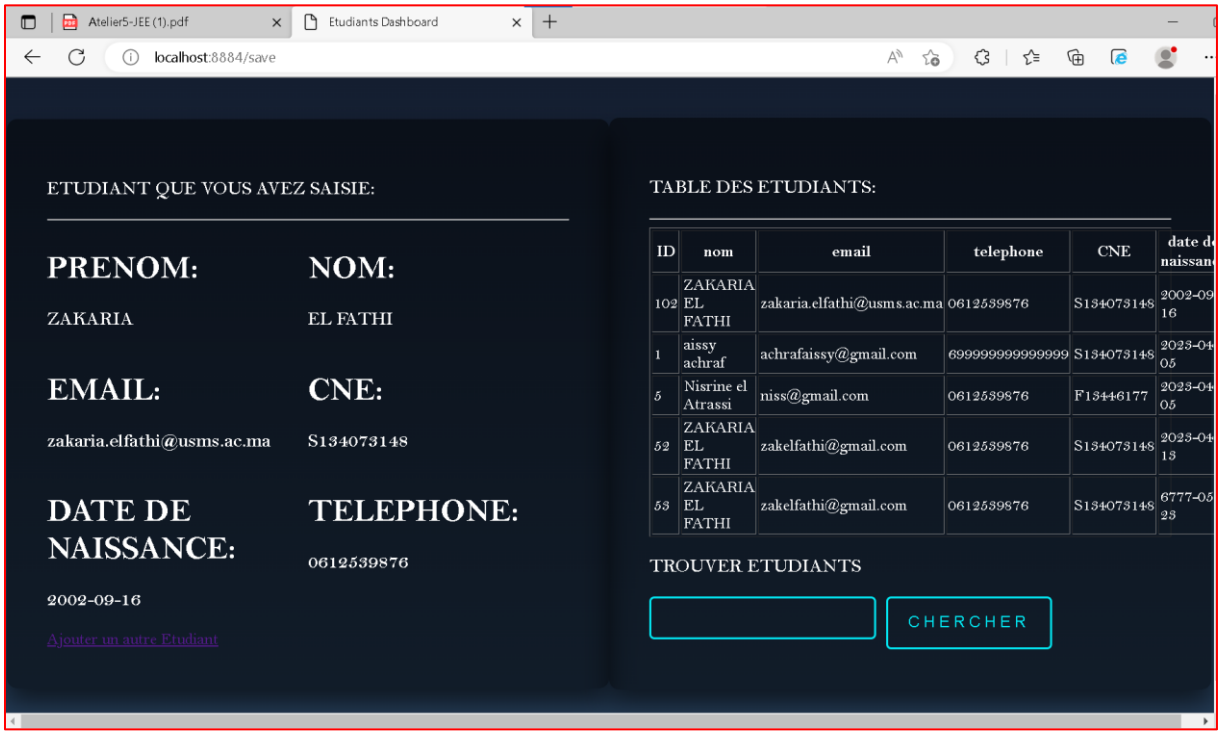
```
            <tr>
                <th>ID</th>
                <th>nom</th>
                <th>email</th>
                <th>telephone</th>
                <th>CNE</th>
                <th>date de naissance</th>
            </tr>
            <tr >
                <td th:text="${etudiant.id}"></td>
                <td th:text="${etudiant.firstName + ' ' + etudiant.lastName}"></td>
                <td th:text="${etudiant.email}"></td>
                <td th:text="${etudiant.tele}"></td>
                <td th:text="${etudiant.cne}"></td>
                <td th:text="${etudiant.dNaiss}"></td>
            </tr>

            <tr th:each="e:${listEtudiants}">
                <td th:text="${e.id}"></td>
                <td th:text="${e.firstName + ' ' + e.lastName}"></td>
                <td th:text="${e.email}"></td>
                <td th:text="${e.tele}"></td>
                <td th:text="${e.cne}"></td>
                <td th:text="${e.dNaiss}"></td>
            </tr>

        </table>
        <div class="input-group">
        <h3>TROUVER ETUDIANTS</h3>
        <form th:action="@{save}" method="post">
            <input type="text" name="mc" th:value="${mc}" />
            <button class="btn btn-primary">chercher</button>
        </form>
    </div>
            <!--

        <div class="container">
            <ul class="nav nav-pills">
                <li th:class="${currentPage}==${status.index}?'active':''"
th:each="page,status:${pages}">
                    <a th:class="page-link" th:href="@{index(page=${status.index}, size=${size},
mc=${mc})}"
                        th:text="${status.index}"></a>
                </li>
            </ul>
```

```
        </div>
        -->
    </div>
</body>
</html>
```

Test de saisie et affichage



On reçoive ces données dans :



PARTIE 2

REINSCRIPTION.java

C'est une classe Java nommée "Reinscription". Cette classe définit une entité pour stocker des informations sur les réinscriptions d'étudiants. Elle est annotée avec des annotations Jakarta Persistence pour déclarer qu'elle est une entité persistante et pour spécifier les relations avec d'autres entités. La classe contient des attributs pour stocker les informations suivantes : l'ID de la réinscription, l'étudiant concerné, la date d'inscription, le niveau, la filière, le groupe, le cycle et le nombre d'années d'inscription. Elle a également des constructeurs et des méthodes getters et setters pour accéder et modifier les attributs.

```
package com.ensa.tp5.entities;
@Entity
public class Reinscription implements Serializable {
    @Id
    @GeneratedValue
    private Long id_insc;
    @ManyToOne
    private Etudiant etudiant;
    private String dInscription;
    private String niveau;
    private String filiere;
    private String groupe;
    private String cycle;
    private String nbrAnneeInscription;

    public Reinscription() {
        super();
    }

    public Reinscription(Etudiant etudiant, String dInscription, String niveau, String filiere, String
groupe,
            String cycle, String nbrAnneeInscription) {
        super();
        this.etudiant = etudiant;
        this.dInscription = dInscription;
        this.niveau = niveau;
        this.filiere = filiere;
        this.groupe = groupe;
        this.cycle = cycle;
        this.nbrAnneeInscription = nbrAnneeInscription;
    }
```

```
public Long getId_insc() {
    return id_insc;
}

public void setId_insc(Long id_insc) {
    this.id_insc = id_insc;
}

public Etudiant getEtudiant() {
    return etudiant;
}

public void setEtudiant(Etudiant etudiant) {
    this.etudiant = etudiant;
}

public String getdInscription() {
    return dInscription;
}

public void setdInscription(String dInscription) {
    this.dInscription = dInscription;
}

public String getNiveau() {
    return niveau;
}

public void setNiveau(String niveau) {
    this.niveau = niveau;
}

public String getFiliere() {
    return filiere;
}

public void setFiliere(String filiere) {
    this.filiere = filiere;
}

public String getGroupe() {
    return groupe;
}

public void setGroupe(String groupe) {
    this.groupe = groupe;
```

```
    }

    public String getCycle() {
        return cycle;
    }

    public void setCycle(String cycle) {
        this.cycle = cycle;
    }

    public String getNbrAnneeInscription() {
        return nbrAnneeInscription;
    }

    public void setNbrAnneeInscription(String nbrAnneeInscription) {
        this.nbrAnneeInscription = nbrAnneeInscription;
    }
}
```

Reinscription controller :

C'est un contrôleur Spring MVC pour la gestion des inscriptions d'étudiants. Il contient deux méthodes :

- formInscription : cette méthode est appelée lorsqu'on accède à l'URL "/form_inscription" en mode GET. Elle renvoie une vue "form-part2" qui permet à l'utilisateur de saisir les informations d'une inscription. Elle ajoute également un objet Reinscription vide au modèle pour permettre la liaison des données de formulaire.
- save : cette méthode est appelée lorsqu'on soumet le formulaire d'inscription en mode POST à l'URL "/save_inscription". Elle prend en entrée l'objet Reinscription rempli par l'utilisateur et effectue une validation via l'annotation @Valid. Si la validation échoue, elle renvoie la vue "form-part2". Sinon, elle enregistre l'inscription dans la base de données via l'interface repoInscription, puis renvoie une vue "ajoutInscriptionConfirmation". Elle effectue également une recherche paginée sur les inscriptions existantes dans la base de données en fonction des paramètres de pagination (page et taille) et de mot clé (mc), puis ajoute les résultats et les informations de pagination au modèle.

```
package com.ensa.tp5.web;

import com.ensa.tp5.dao.repoInscription;
import com.ensa.tp5.entities.Etudiant;
import com.ensa.tp5.entities.Reinscription;

@Controller
public class controllerInscription {
```

```
    @Autowired
    private repoInscription repInscription;



    @RequestMapping(value = "/form_inscription", method = RequestMethod.GET)
    public String formInscription(Model model) {
        model.addAttribute("inscription", new Reinscription());
           Reinscription inscription = new Reinscription();
           inscription.setEtudiant(new Etudiant()); // create a new Etudiant object and set it in the inscription

        return "form-part2";
    }

    @RequestMapping(value = "/save_inscription", method = RequestMethod.POST)
    public String save(Model model, @Valid Reinscription inscription, BindingResult bindingResult, @RequestParam(name
= "page", defaultValue = "0") int p,
            @RequestParam(name = "size", defaultValue = "5") int s,
            @RequestParam(name = "mc", defaultValue = "") String mc) {
        Page<Reinscription> pageInscriptions = repInscription.chercher("%" + mc + "%", PageRequest.of(p, s));
        model.addAttribute("listInscriptions",pageInscriptions.getContent());
//      tableau de pages
        int[] pages = new int[pageInscriptions.getTotalPages()];
        model.addAttribute("pages", pages);
        model.addAttribute("size", s);
        model.addAttribute("currentPage", p);
        model.addAttribute("mc", mc);

        if (bindingResult.hasErrors())
            return "form-part2";
        repInscription.save(inscription);
        return "ajoutInscriptionConfirmation";
    }
}
```

Repository :

C’est une interface qui utilise Spring Data JPA pour la gestion de la persistence des objets Reinscription.
L'interface repoInscription hérite de l'interface JpaRepository fournie par Spring Data JPA et spécialise le type Reinscription en tant que type d'entité et Long en tant que type de l'identifiant. Cela signifie que repoInscription dispose des méthodes de base de JpaRepository pour la manipulation des objets Reinscription, telles que save(), delete(), findAll(), etc.
De plus, il définit une méthode personnalisée nommée chercher() qui permet de rechercher des inscriptions en fonction d'une chaîne de caractères passée en paramètre (mc) en utilisant une requête JPQL (select e from Reinscription e where e.id_insc like :x). Cette méthode

retourne une page d'objets Reinscription pour prendre en charge la pagination en utilisant l'objet Pageable fourni par Spring Data JPA.
En bref, cette interface fournit une abstraction pour la manipulation des objets Reinscription en utilisant Spring Data JPA et permet de rechercher des inscriptions en fonction d'un critère de recherche donné.

```
package com.ensa.tp5.dao;
import com.ensa.tp5.entities.Reinscription;


public interface repoInscription extends JpaRepository<Reinscription, Long>{
    @Query("select e from Reinscription e where e.id_insc like :x")
    public Page<Reinscription> chercher(@Param("x")String mc, Pageable pageable);

}
```

Formulaire de reinscription

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="utf-8" />
    <title>Ajouter Inscription</title>
    <link rel="stylesheet" type="text/css" th:href="@{css/style.css}" />
    <link rel="stylesheet" type="text/css" th:href="@{bootstrap/bootstrap.min.css}" />
</head>
<body>
    <div class="login-box">
        <h2>Inscription:</h2>
        <form  method="post"  th:object="${inscription}" th:action="@{save_inscription}">
        <div class="input-group">


                <div class="user-box">
                <input type="text" class="form-control"  th:field="*{etudiant.firstName}" id="firstName"
      name="firstName" placeholder="votre prenom" required/>
                </div>


                <div class="user-box">
                    <input type="text" class="form-control"  th:field="*{etudiant.lastName}" id="lastName"
                        name="lastName" placeholder="votre prenom" required/>
                </div>
                <div class="user-box">
                    <input type="email" class="form-control"  th:field="*{etudiant.email}" id="email" name="email"
                        placeholder="votre email" required />
                </div>
```

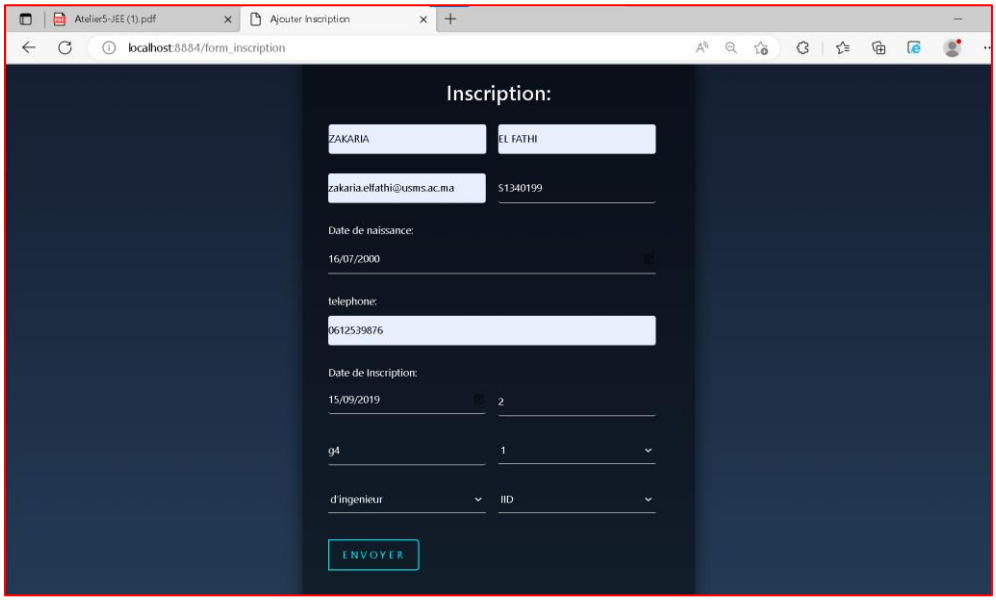
```
                <div class="user-box">
                    <input type="text" class="form-control"  th:field="*{etudiant.cne}" id="cne" name="cne"
                        placeholder="votre CNE" required />
                </div>
                <label class="user-box">Date de naissance:</label>
                <div class="user-box">
                    <input type="date"  th:field="*{etudiant.dNaiss}" id="dNaiss" name="dNaiss" required/>
                </div>
                <label class="user-box">telephone:</label>
                <div class="user-box">
                    <input type="tele" class="form-control"  th:field="*{etudiant.tele}" id="tele" name="tele"
                        placeholder="votre numero de telephone" required />
                </div>

            <label class="user-box">Date de Inscription:</label>
            <div class="user-box">
                <input type="date" class="form-control"  th:field="*{dInscription}" id="dInscription"
name="dInscription" placeholder="votre date inscription">
            </div>
            <div class="user-box">
                <input type="number" class="form-control" th:field="*{dInscription}"  id="nbrAnneeInscription"
name="nbrAnneeInscription" placeholder="Nombre d'ans d'inscription">
            </div>
            <div class="user-box">
                <input type="text" class="form-control"  th:field="*{dInscription}"  id="groupe" name="groupe"
placeholder="votre groupe">
            </div>



            <div class="user-box">
                    <select name="niveau"  th:field="*{niveau}" class="custom-select" id="niveau" >
                      <option value=" ">Select niveau..</option>
                      <option value="1">1</option>
                      <option value="2">2</option>
                      <option value="3">3</option>
                    </select>
            </div>
            <div class="user-box">
                    <select name=cycle  th:field="*{cycle}" class="custom-select" id="cycle" >
                      <option value=" ">Select cycle..</option>
                      <option value="preparatoire">preparatoire</option>
                      <option value="d'ingenieur">d'ingenieur</option>
                    </select>
            </div>
            <div class="user-box">
                    <select name="filiere" th:field="*{filiere}" class="custom-select" id="filiere" >
```

```
                    <option value=" ">Select filiere..</option>
                    <option value="cp">CP</option>
                    <option value="iric">IRIC</option>
                    <option value="iid">IID</option>
                    <option value="gi">GI</option>
                    <option value="ge">GE</option>
                    <option value="gpee">GPEE</option>
                  </select>
            </div>
            <button  type="submit">envoyer</button>
        </div>
        </form>
    </div>
    </body>
</html>
```

Affichage des donnees saisies :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">

<head>
    <meta charset="utf-8" />
    <title>Inscriptions Dashboard</title>
    <link rel="stylesheet" type="text/css" th:href="@{css/styleAffichage.css}" />
</head>

<body>
    <div class="login-box">
            <h3>Inscription QUE VOUS AVEZ SAISIE:</h3>
        <hr/>
        <div class="input-group">
            <div class="user-box">
                <h1><strong>PRENOM:</strong>
                    <h3 th:text="${etudiant.firstName}"></h3>
                </h1>
            </div>
            <div class="user-box">
                <h1><strong>NOM:</strong>
                    <h3 th:text="${etudiant.lastName}"></h3>
                </h1>
            </div>
            <div class="user-box">
                <h1><strong>EMAIL:</strong>
                    <h3 th:text="${etudiant.email}"></h3>
```

```
        </h1>
    </div>
    <div class="user-box">
        <h1><strong>CNE:</strong>
            <h3 th:text="${etudiant.cne}"></h3>
        </h1>

    </div>
    <div class="user-box">
        <h1> <strong>DATE DE NAISSANCE:</strong>
            <h3 th:text="${etudiant.dNaiss}"></h3>

        </h1>
    </div>
    <div class="user-box">
        <h1><strong>TELEPHONE:</strong>
            <h3 th:text="${inscription.etudiant.tele}"></h3>

        </h1>

        <div class="user-box">
        <h1><strong>id ins:</strong>
            <h3 th:text="${inscription.id_insc}"></h3>

        </h1>
    </div>
                    <a  th:href="@{/form}">Ajouter une autre inscription</a>
             </div></div></div></body></html>
```



CONCLUSION

Cet atelier a permis d'aborder plusieurs aspects liés à la mise en place d'une application web Java EE avec Spring. Tout d'abord, nous avons vu comment créer et configurer un projet Spring à l'aide de l'outil Spring Boot. Nous avons également examiné la structure d'un projet Spring et comment créer des entités JPA pour stocker des données dans une base de données.

Nous avons ensuite examiné la création de pages HTML à l'aide du moteur de template Thymeleaf, la communication avec les contrôleurs Spring, la validation des entrées de formulaire et la gestion des erreurs

En somme, cet atelier a fourni une vue d'ensemble de la création d'une application web Java EE avec Spring, en insistant sur les bonnes pratiques et les outils de développement modernes pour faciliter la création et la maintenance d'applications web robustes et évolutives.