Programmation Avancés HMIN327
HMIN327

MonumTour: Application de gestion de monuments liés à des célébrités

Etudiant:
• Kacioui Arezki

Enseignants :
• Duroux Patrice
• Mougenot Isabelle

2020/2021

Sommaire

1. Introduction
2. Exigences fonctionnelles
3. Conception
4. Architecture technique
5. Conclusion
6. Démonstration de l’application

1. Introduction

L'application monumTour a été développé dans le cadre de l'UE HMIN327 (Programmation avancée)

Appliquer les notions vus en cours notamment la persistance des données, le mapping objet-relationnel et le modèle MVC

Application web respectant le model MVC, offrant une interface d'administration ou de consultation selon le type d'utilisateurs sur les différentes entités

Respect d'un cahier des charges et d'un modèle de données

2. Exigences fonctionnelles

- L'application doit offrir une possibilité de connexion avec différents types d'utilisateurs:
  - Un administrateur
  - Un voyagiste
  - Un touriste
- Selon le type d'utilisateur, celui-ci aura accès à des fonctionnalités spécifiques:
  - L'administrateur peut effectuer des consultations, des ajouts, des mises à jour et des suppressions sur l'ensemble des entités, y compris les utilisateurs.
  - Le voyagiste peut effectuer des consultations, des ajouts, des mises à jour et des suppressions sur l'ensemble des entités, hormis les utilisateurs.
  - Le touriste peut effectuer des consultations.
- Effectuer des prétraitement tels que le calcul de distance entre monuments ou l'affichage sur une carte.
- Respecter le modèle de données qui est donnée.

3. Conception

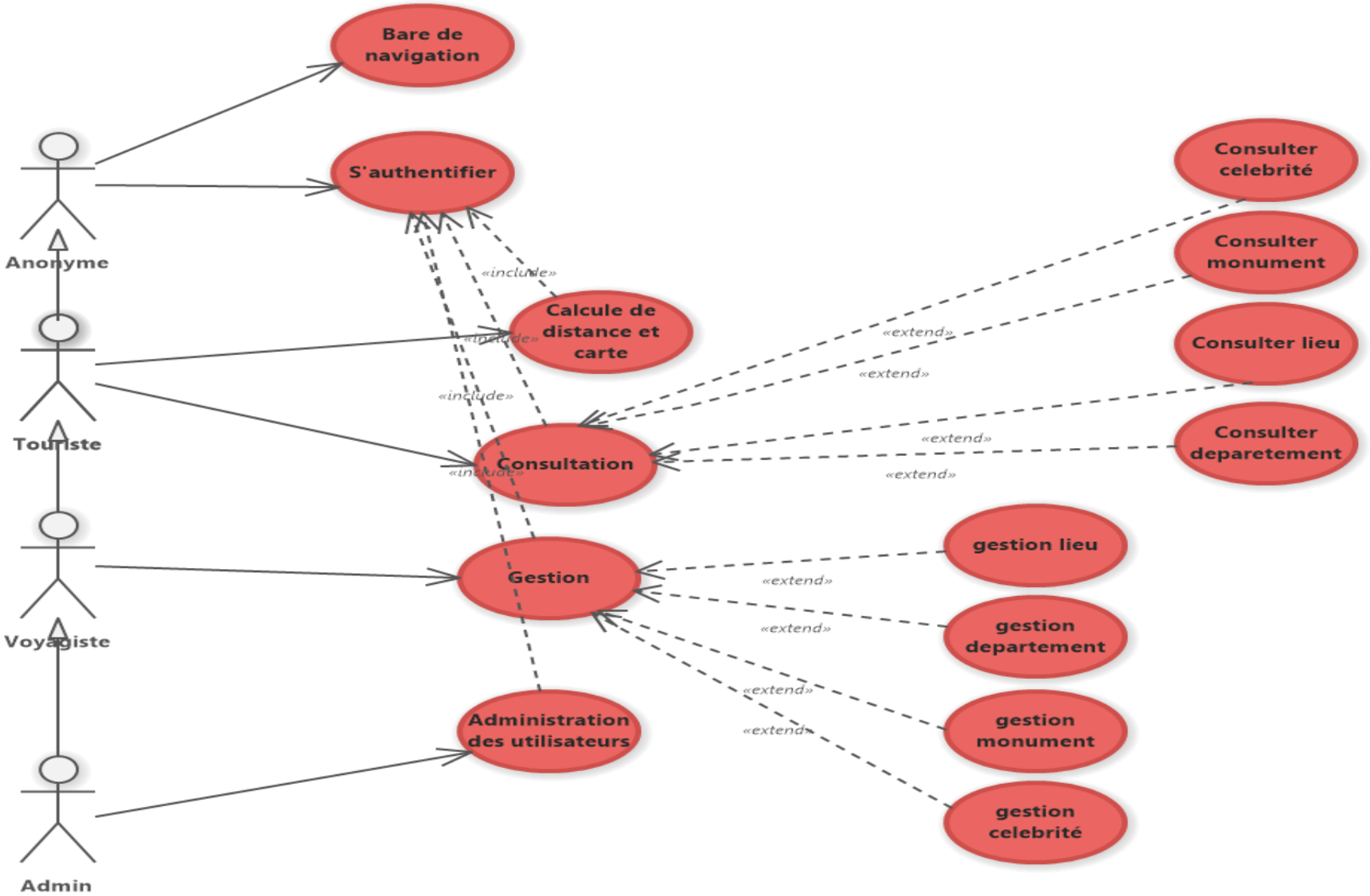


Diagramme de cas d'utilisation

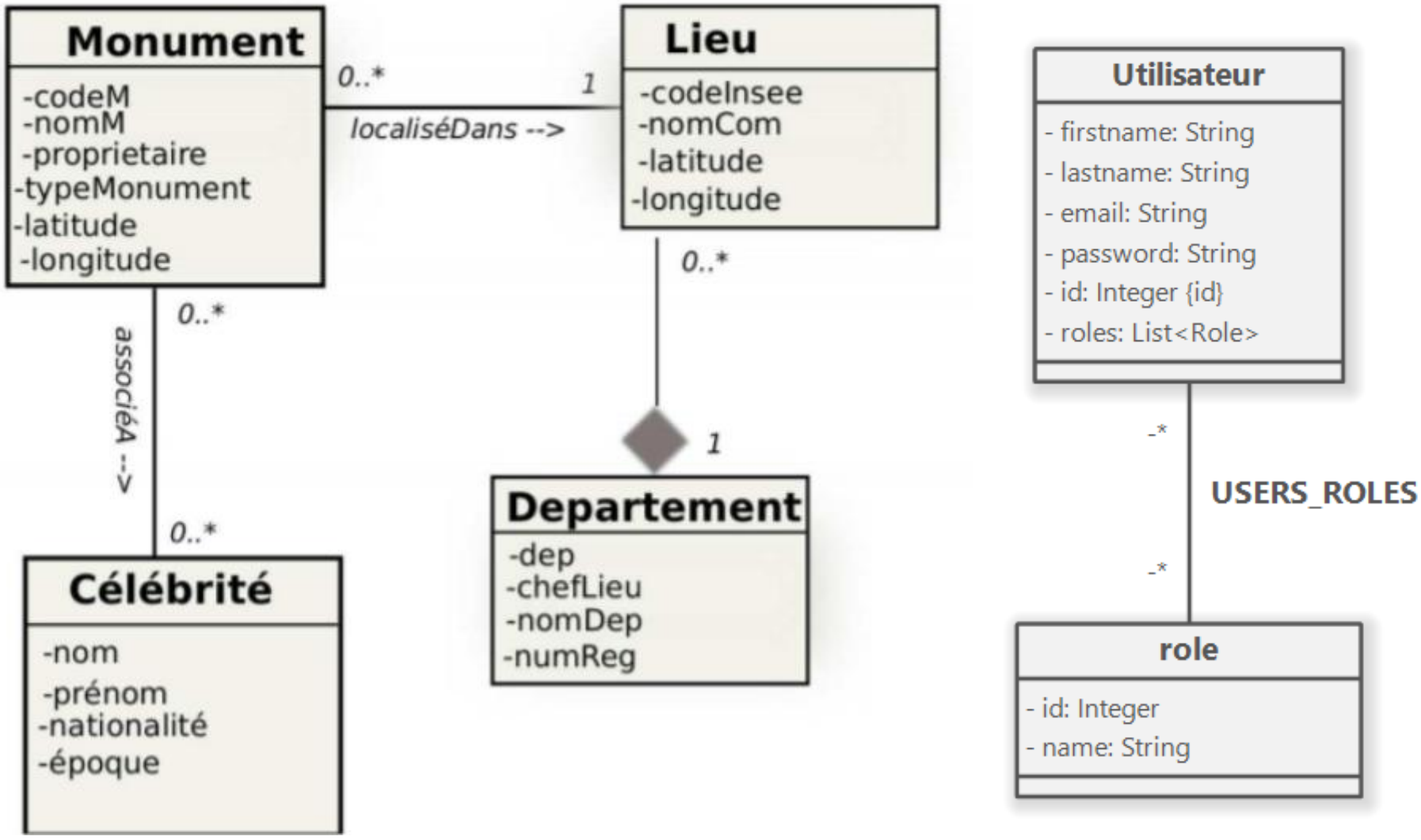
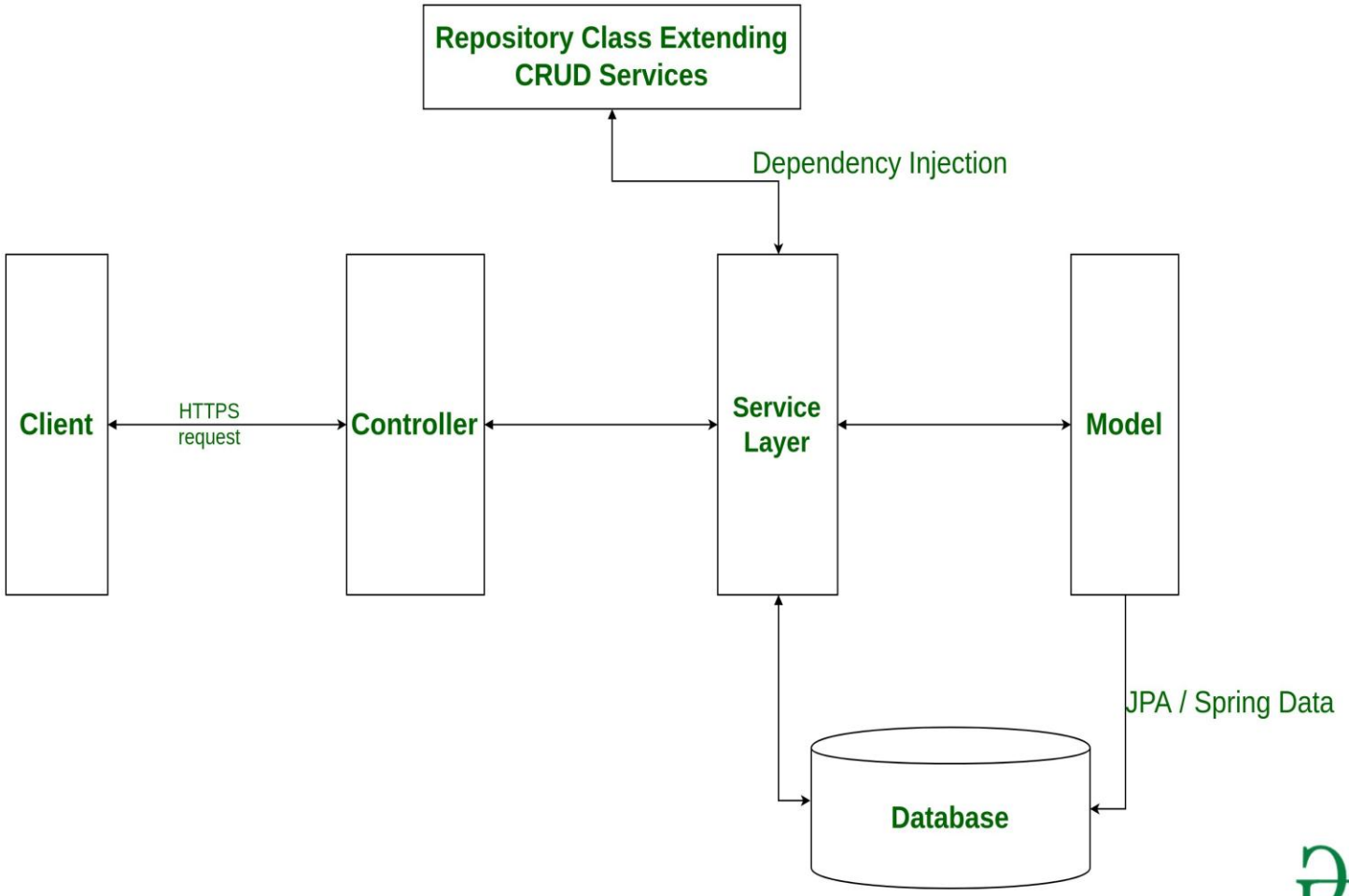


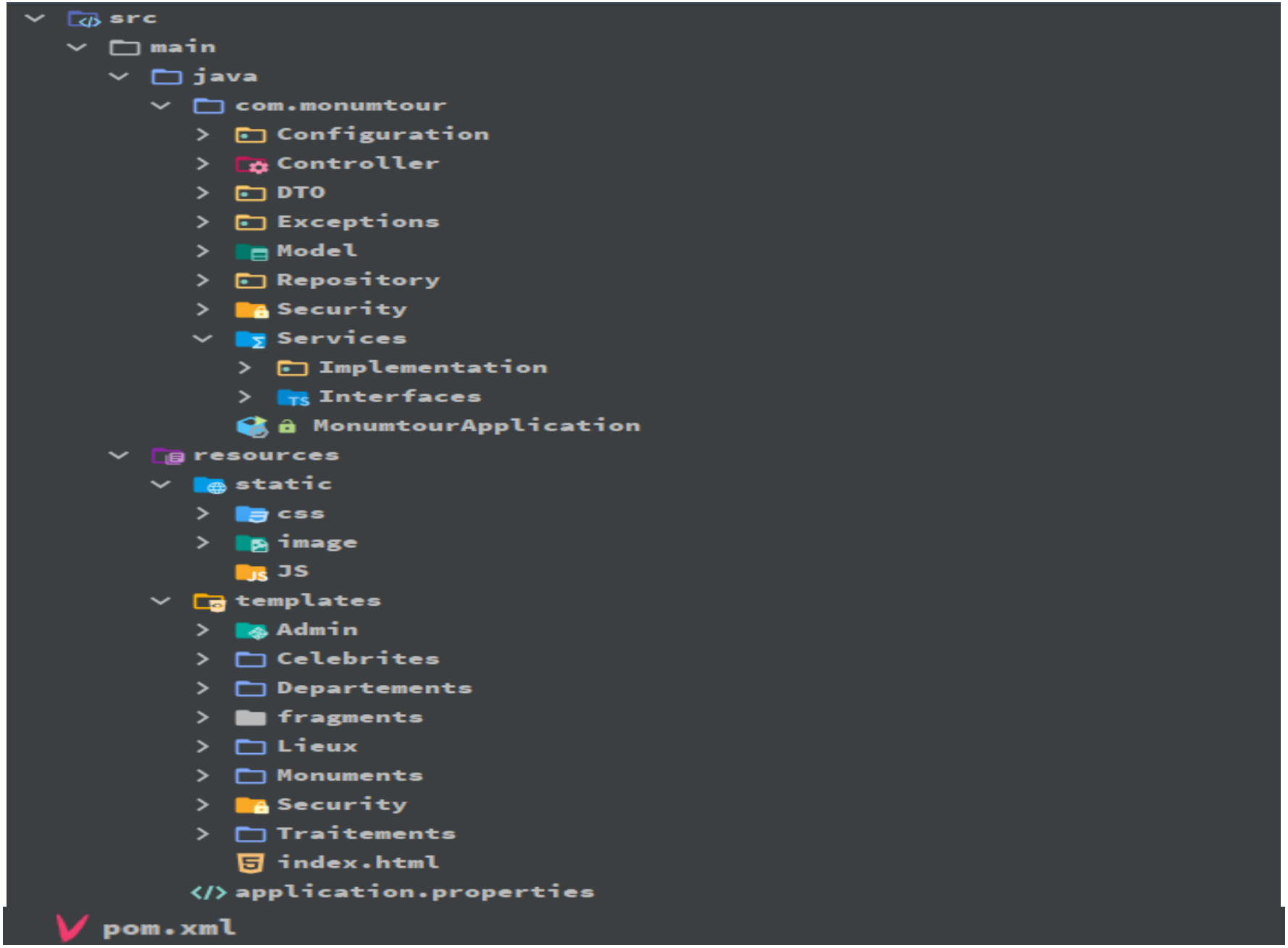
Diagramme de classes

4. Architecture

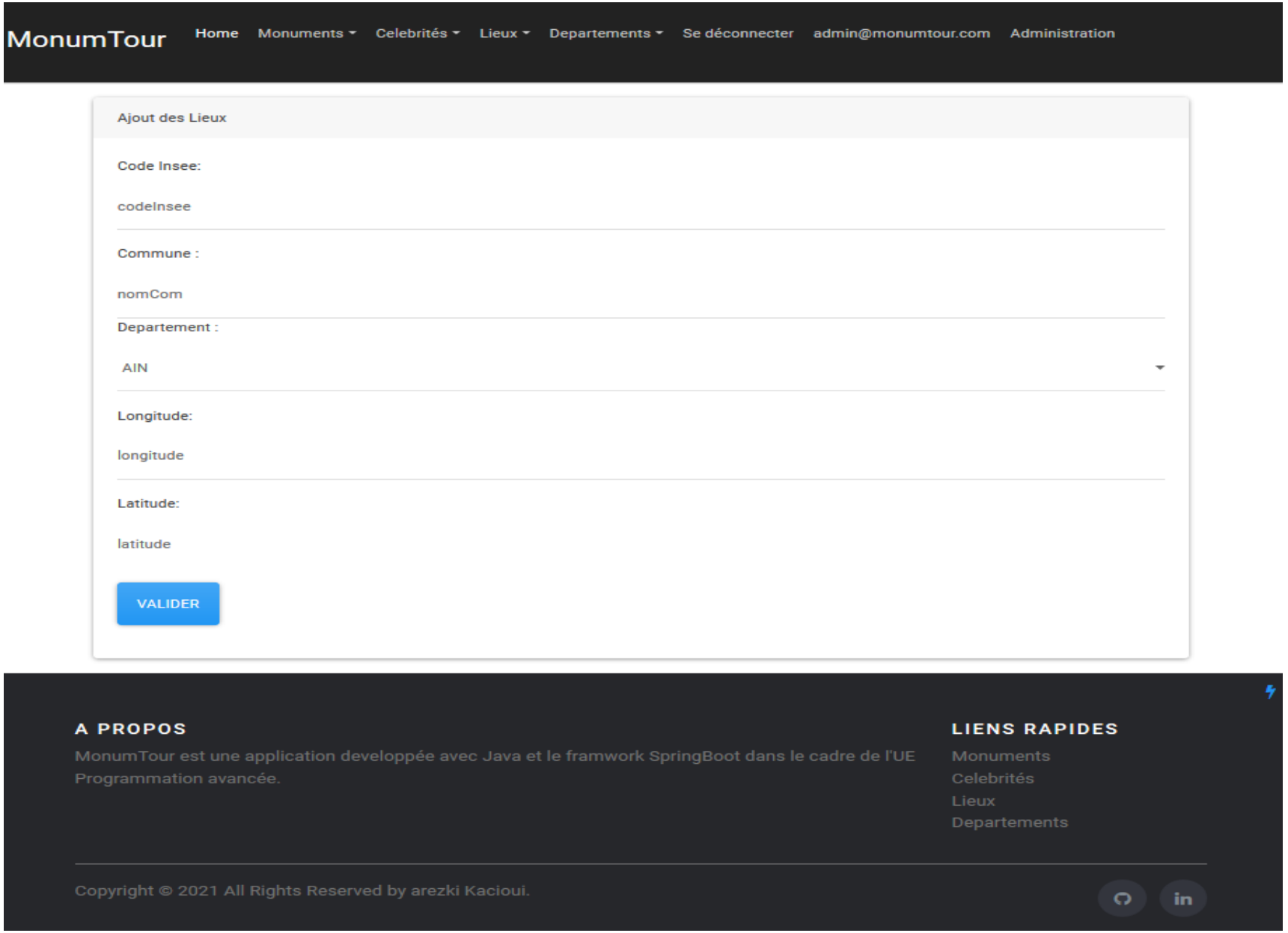
Technologies utilisées

Spring Boot flow architecture





Arborescence du projet



- A la validation, une requête de type post est envoyé à la route /savelieu

La requête envoyée est reçue par MonumentController et effectue les traitements en faisant appel aux différents services qui y sont injectés, et renvoie ensuite la Template allMonuments.html

```java
package com.monumtour.Controller;


import ...

@Controller
public class MonumentController {

    //injection de dependances
    @Autowired
    private IMonumentService monumentService;
    @Autowired
    private ILieuService lieuService;
    @Autowired
    private ICelebriteService celebriteService;

    @Secured(value = {"ROLE_ADMIN"})
    @RequestMapping("/saveMonument")
    public String saveMonument(@ModelAttribute("monument")Monument m, ModelMap modelMap){
        monumentService.saveMonument(m);
        List<Monument> monuments = monumentService.getAllMonuments();
        modelMap.addAttribute( attributeName: "monuments",monuments);
        return "Monuments/allMonuments";
    }
```

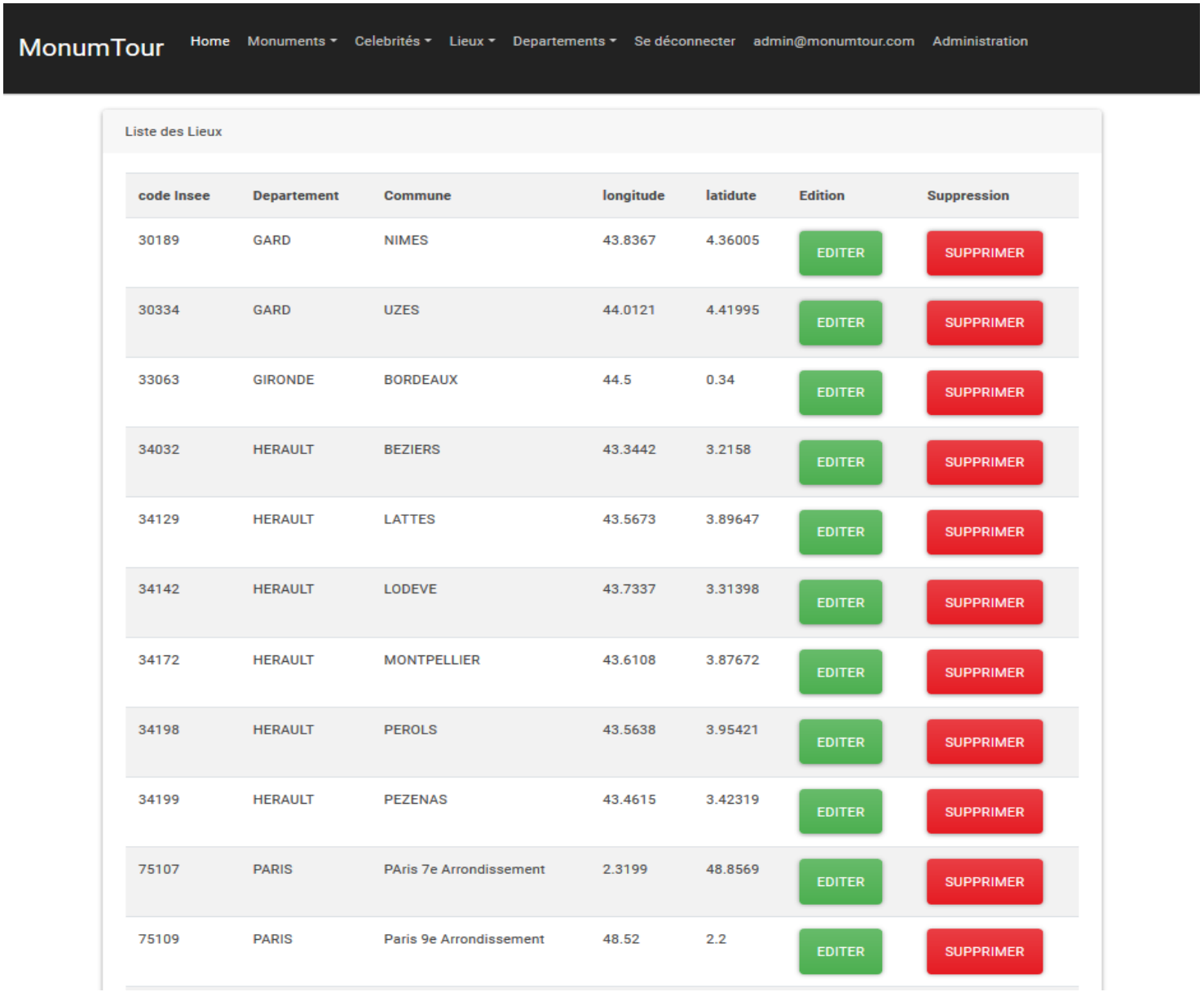
Le service MonumentService implémente l'interface IMonumentService.
Utilise le MonumentRepository interagissant ainsi avec l'entité Monument et la base de donnés.

```java
@Service
public class MonumentService implements IMonumentService {
    @Autowired
    private MonumentRepository monumentRepository;

    @Override
    public Monument saveMonument(Monument m) { return monumentRepository.save(m); }
    @Override
    public Monument updateMonument(Monument m) { return monumentRepository.save(m); }
    @Override
    public void deleteMonumentById(String id) { monumentRepository.deleteById(id); }
    @Override
    public Monument getMonument(String id) { return monumentRepository.findById(id).get(); }
    @Override
    public List<Monument> getAllMonuments() { return monumentRepository.findAll(); }
    @Override
    public float distance(String codeM1, String codeM2) {
        final int R = 6371; // Radius of the earth
        Monument m1 = monumentRepository.findById(codeM1).get();
        Monument m2 = monumentRepository.findById(codeM2).get();
        float latDistance = (float) Math.toRadians(m2.getLatitude() - m1.getLatitude());
        float lonDistance = (float) Math.toRadians(m2.getLongitude() - m1.getLongitude());
        float a = (float) (Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
                + Math.cos(Math.toRadians(m1.getLatitude())) * Math.cos(Math.toRadians(m2.getLatitude()))
                * Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2));
        float c = (float) (2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a)));
        float distance = R * c * 1000; // convert to meters
        distance = (float) Math.pow(distance, 2);
        return (float) (Math.sqrt(distance)/1000);
    }
```

On utilise les annotations afin de spécifier les relations entre les entités

```java
@Entity
public class Monument implements Serializable{
    private static final long serialVersionUID = 1L;
    @Id
    private String codeM;
    private String nomM;
    private String proprietaire;
    private String typeMonument;
    private float latitude;
    private float longitude;
    @ManyToOne
    @JoinColumn(name="FK_CodeInsee")
    private Lieu localite;
    @ManyToMany
    @JoinTable(name="AssocieA",joinColumns= @JoinColumn(name="codeM"),
            inverseJoinColumns=@JoinColumn(name="codeCelebrites"))
    private Collection<Celebrite> celebrites;
```

La Template allMonument est retournée à l'uilisateur



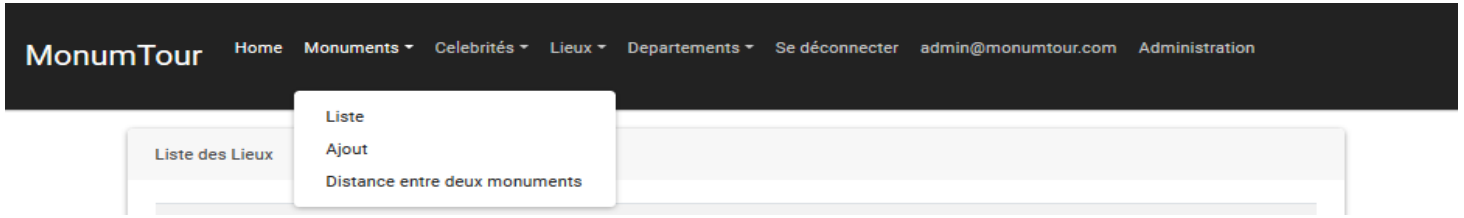
Spring Security permet de gérer les rôles ainsi que l'accès aux routes

➢ Utilisation du dialecte sec de Spring security afin de gérer les accès au sein des templates, et des annotations @Secured au sein des Controller

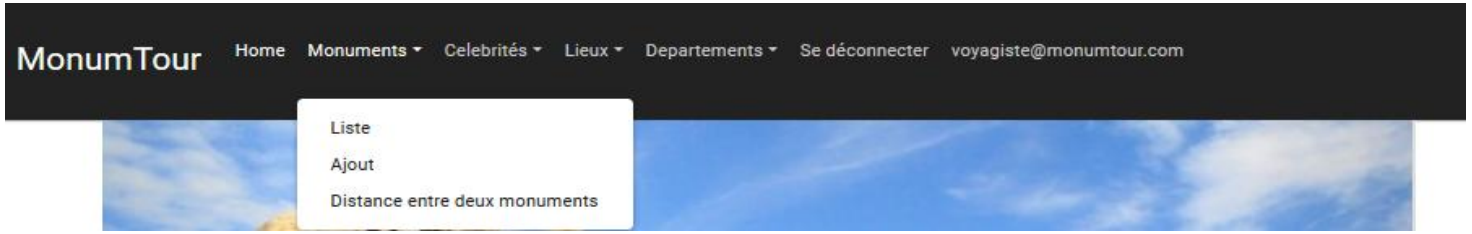
```
@Secured(value = {"ROLE_SUPER_ADMIN"})
```

```
    <div class="dropdown-menu" aria-labelledby="navbarDropdownDepartement">
        <a class="dropdown-item" href="#" th:href="@{/allDepartements}">Liste</a>
        <a sec:authorize="hasRole('ROLE_ADMIN')" class="dropdown-item" href="#" th:href="@{/addDepartement}">Ajout</a>
    </div>
</li>

<li class="nav-item" sec:authorize="isAnonymous()">
    <a class="nav-link " href="#" th:href="@{/registration}">S'inscrire</a>
</li>
<li class="nav-item" sec:authorize="isAnonymous()">
    <a class="nav-link " href="#" th:href="@{/login}">Se connecter</a>
</li>

<li class="nav-item" sec:authorize="isAuthenticated()" >
    <a class="nav-link" href="#" th:href=" @{/logout} " >Se déconnecter</a>
</li>
<li class="nav-item" sec:authorize="isAuthenticated()">
    <p class="nav-link" sec:authentication="name"></p>
</li>
<li class="nav-item" sec:authorize="hasRole('ROLE_SUPER_ADMIN')">
    <a class="nav-link" href="#" th:href="@{/admin}">Administration</a>
</li>
```

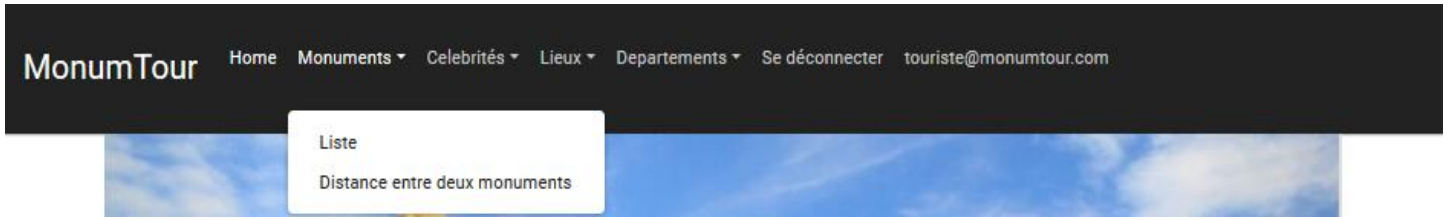
➢ L'administrateur (ROLE_SUPER_ADMIN) a accès a tout.



➢Le voyagiste (ROLE_ADMIN) a accès à la gestion des entités hormis les utilisateurs.



➢Le touriste (ROLE_USER) a accès à la consultation



5. Conclusion

Expérience enrichissante au niveau technique

Conception d'application Web avec Java SpringBoot, Thymeleaf et MySQL

Concevoir un modèle de données et le respecter

Respecter un cahier des charges

6. Démonstration sur machine