



UNIVERSITÉ  
DE MONTPELLIER

**HMIN327 - RAPPORT**

---

## **Projet**

---

**Etudiant :**

*Amir SHIRALI POUR*

**Enseignant :**

Isabelle MOUGENOT

Patrice Duroux

**Février 2021**

# **Table des matières**

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Objectifs du projet</b>	<b>1</b>
<b>3</b>	<b>Outils utilisés</b>	<b>3</b>
	<b>A. Gestion de la base de données</b>	<b>3</b>
	<b>B. Conception BackEnd</b>	<b>4</b>
	<b>C. Object-Relational Mapping ORM</b>	<b>4</b>
	<b>D. Conception FrontEnd</b>	<b>4</b>
<b>4</b>	<b>La conception</b>	<b>5</b>
	<b>A. Fonctionnalités du relationnel</b>	<b>9</b>
	<b>B. Les Fonctionnalités de l'application</b>	<b>11</b>
	<b>C. Autres fonctionnalités</b>	<b>14</b>
<b>5</b>	<b>Les problèmes rencontrés</b>	<b>16</b>
<b>6</b>	<b>Conclusion</b>	<b>18</b>
<b>7</b>	<b>Annexes</b>	<b>19</b>

## **1. Introduction**

Dans le cadre de la programmation orientée objet, et afin de mettre en œuvre les connaissances acquises dans cette UE de programmation avancée, j'ai réalisé le projet proposé traitant la conception d'une application Web.

La réalisation de cette application web permet l'application d'acquis en premier lieu en programmation et en développement web. Ce projet alliant la création d'une plateforme dynamique, avec l'idée de construire un projet durable dans le temps, qui peut avoir une utilité certaine à l'utilisateur et facilement manipulable.

Dans ce rapport je détaillerai les objectifs du projet, les outils et les étapes de la conception de ce dernier avec les problématiques rencontrées et les solutions proposées, pour enfin présenter le produit final.

## **2. Objectifs du projet**

L'objectif de ce projet est de construire une application décrivant des Monuments historiques rattachés à des personnages célèbres, et fournir leur géo-localisation décrite à son tour par un Lieu et son Departement. Comme c'est représenté dans le diagramme de classe UML (figure 1), une Celebrite peut avoir plusieurs Monuments, un Monument peut représenter plusieurs Celebrite, mais un Monument ne peut être situé que dans un seul Lieu qui est composé d'un seul Departement.

Pour cela, un jeu de données est également fourni avec 4 tables (Celebrite, Monument, Lieu, Departement), ainsi que les contraintes de clé primaire et les contraintes d'inclusion. Ces données sont donc fournies comme model et il reste donc à les enrichir et étendre le modèle conceptuel sur des façon logiques et utiles.

Ce qui est des exigences techniques, il est demandé de stocker et organiser les données avec un SGBD adapté.

L'architecture de l'application devra être constituée de plusieurs couches : une couche DAO pour la validation des données, une couche métier pour l'exécution des scénarios d'utilisation, et enfin une couche Web.

Les fonctionnalités de l'application devront permettre l'exécution différents scénarios d'utilisation et notamment l'application devra permettre de consulter les données de la base en ressortissant l'aspect relationnel du jeu de données et offrir la possibilité d'effectuer des mises à jour ainsi que l'attribution des droits d'utilisation (lecture/écriture) qui donnera Lieu à des vues externe différentes.

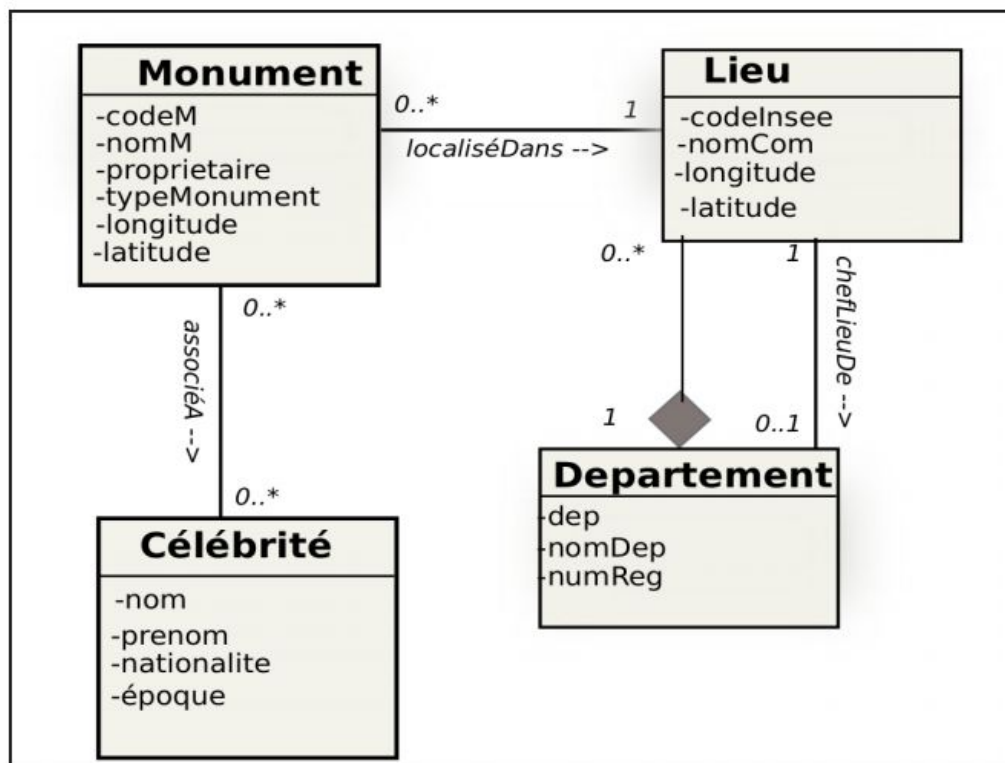


Figure 1 : Diagramme de classes UML

### 3. Outils utilisés

La conception de cette application comprend 4 parties principales qui sont : la gestion de la base de donnée, la conception FrontEnd, la conception BackEnd et enfin le coté mapping d'objet relationnel (JPA). Pour construire chaque aspect de la conception j'ai utilisé différents outils informatiques adaptés aux tâches attendues.

Les code de l'application est sur lien :

<https://github.com/su6i/masterIpsSemester3/tree/master/HMIN327%20%20Programmation%20avancée/Project/ProjectJEESpring>

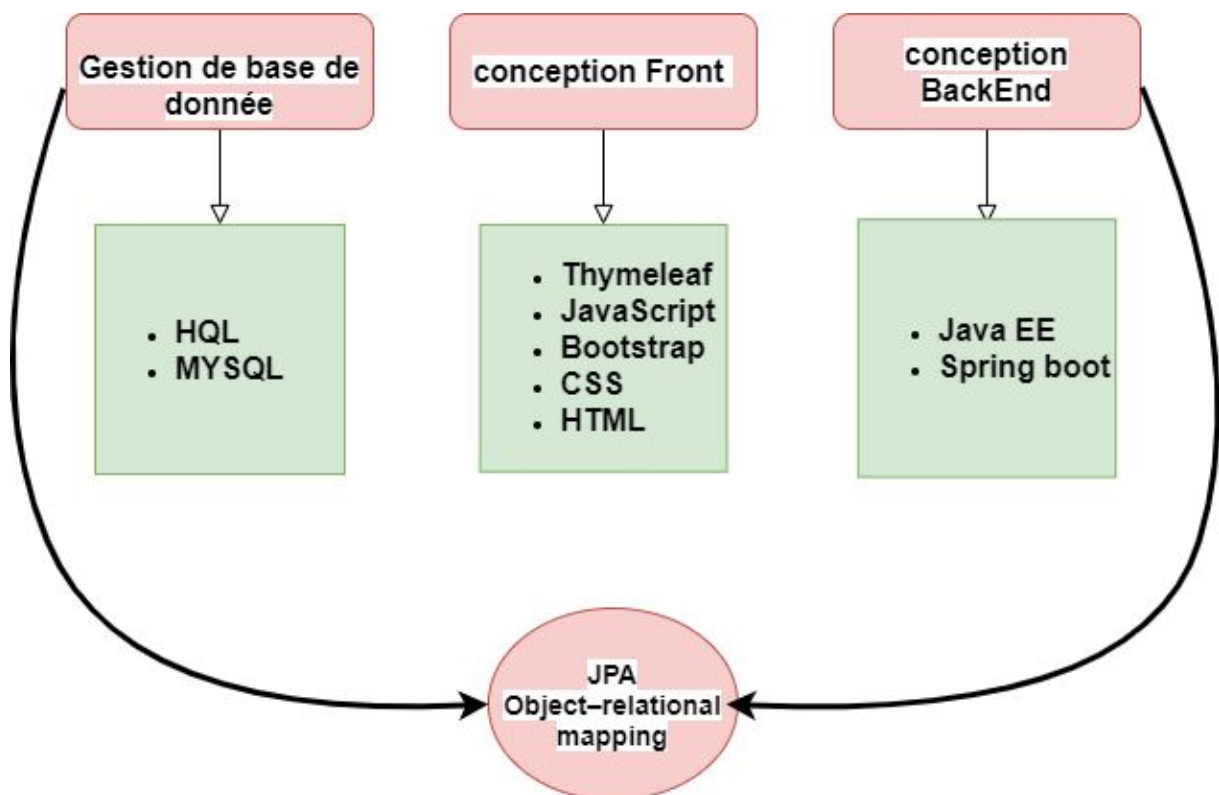


Figure 2 : Graphe des outils de conception

#### A. Gestion de la base de données

Pour cette tâche j'ai choisi d'utiliser HQL et MySQL.

- **HQL** Hibernate Query Language étant un langage puissant de requêtage et totalement orienté objet semble adapté à ce projet.
- **MySQL** SGBD orienté objet permet de gérer les relations entre les tables de mon jeu de données et de ressortir l'aspect relationnel exigé, en plus de la facilité de sa manipulation.

## **B. Conception BackEnd**

Les services BackEnd de l'application sont conçues à l'aide de Java EE et Spring boot.

- **Java EE** jugé le mieux adapté grâce à son orientation objet, et sa compatibilité avec une application-web et database-system.
- **Spring boot** choisi pour sa compatibilité avec l'architecture Java et la facilité de manipulation et configuration tout en respectant le patron de conception MVC.

## **C. Object-Relational Mapping ORM**

L'ORM va établir la correspondance entre le schéma de la base de données et la partie BackEnd du programme. Il lie donc le côté objet au côté relationnel de l'application. Pour ça j'ai utilisé Java Persistence API (Spring Data JPA).

- **Spring Data JPA** cet API de JAVA permet de réaliser toutes les associations de type objet to objet, et il est compatible avec Hibernate (HQL) et offre une flexibilité pour la manipulation d'objets.

## **D. Conception FrontEnd**

Pour le côté FrontEnd du programme j'ai utilisé différents outils conçus pour gérer la présentation d'un logiciel web : CSS, HTML, Bootstrap, JavaScript, Thymeleaf.

- **HTML, CSS, Bootstrap** : j'ai utilisé HTML pour architecturer la présentation des pages web de mon application auxquelles j'ai ajouté du style pour une bonne présentation

ergonomique du côté utilisateur avec les feuilles de style CSS et des modèles de Bootstrap.

- **JavaScript** : j'ai utilisé ce langage de script léger des pages web pour son orientation objet et aussi pour rendre les pages web plus dynamique.
- **Thymeleaf** : un Template d'environnement Java coté serveur qui apporte un aspect naturel workflow, pouvant générer du HTML et il est compatible avec les application web basées sur le model MVC ce qui justifie mon choix.

#### 4. La conception

L'architecture du programme a été construite selon les fonctionnalités qu'on cherche à concevoir comme représenté sur la figure 4 ci-dessous.

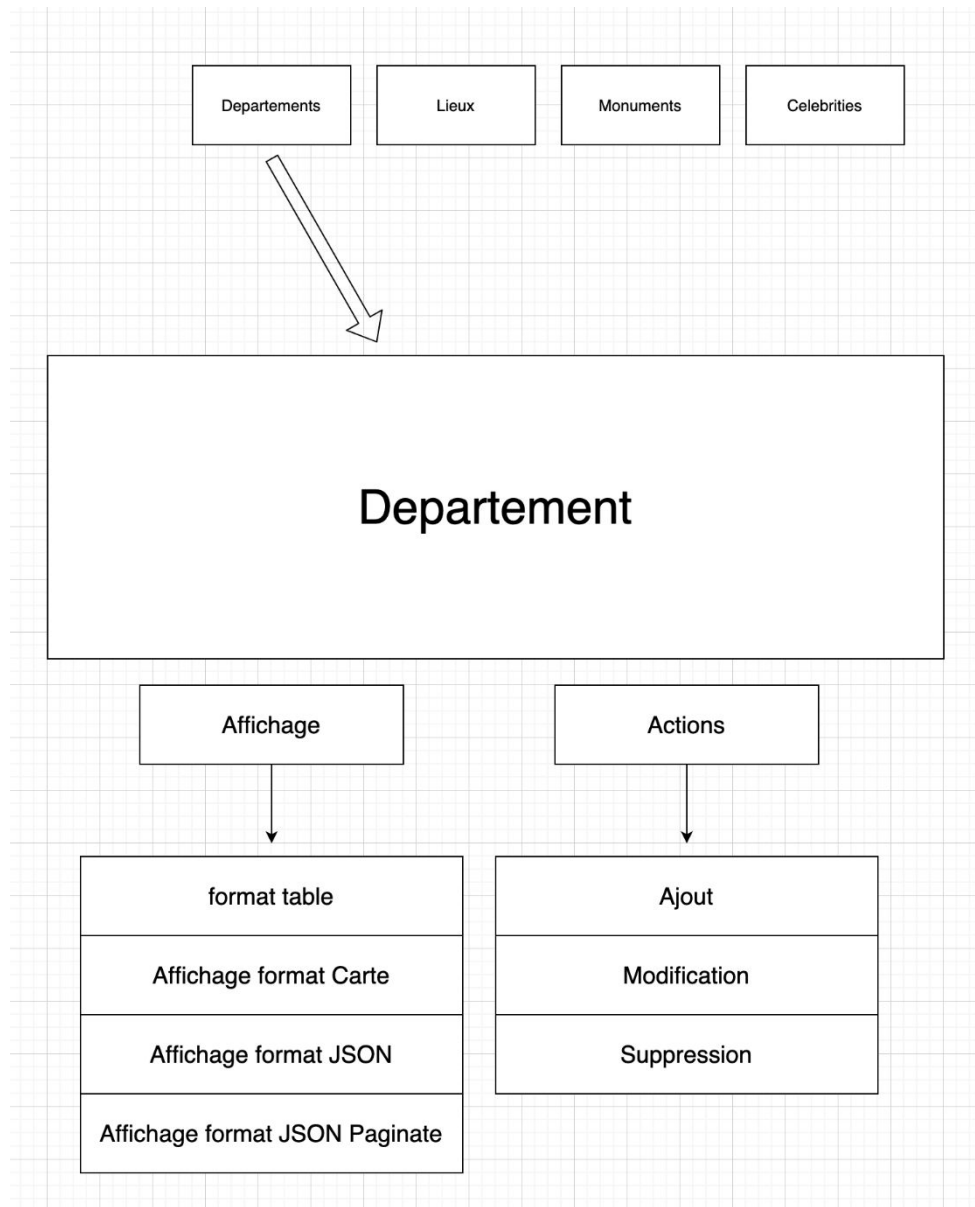
Pour chaque donnée décrite dans les tables du jeu de données le programme offrira deux fonctions principalement l'affichage et les actions possibles sur ces données.

La fonction d'affichage présente l'information au format table, carte, au format JSON et enfin au format JSON Paginate ou un affichage en format détaillé.

Les actions qu'offre l'application sur ces données est de pouvoir les modifier ou les supprimer ou d'en ajouter et cela selon le statut de l'utilisateur et les droits de lecture/écriture qui lui sont attribués.

Prenant comme exemple une instance de la classe Departement, cette information pour être afficher sur la table des Departements listés. L'utilisateur pourra consulter sa géolocalisation sur la carte, de l'afficher au format JSON. De plus de l'affichage cette instance de Departement pourra être supprimer ou modifier par les utilisateurs ayants les droits d'accès à ces actions. Respectivement toutes ces fonctionnalités sont applicables aux autres table de données de l'application.

J'ai vu nécessaire d'ajouter une autre table "User" pour gérer les données des utilisateurs. les même fonctionnalités (CRUD) peuvent s'y appliquer en ayant les droit nécessaires



**Figure 3 : fonctionnalités générales/utilisateur de l'application.**



## Diagrammes de Séquence et de cas d'utilisation

La conception de l'application repose sur la construction de l'architecture de programme en se basant sur les scénarios d'utilisations des fonctionnalités offertes. Ces scénarios sont représentés sur le diagramme de cas d'utilisation et le diagramme de séquence (figure 4-5).

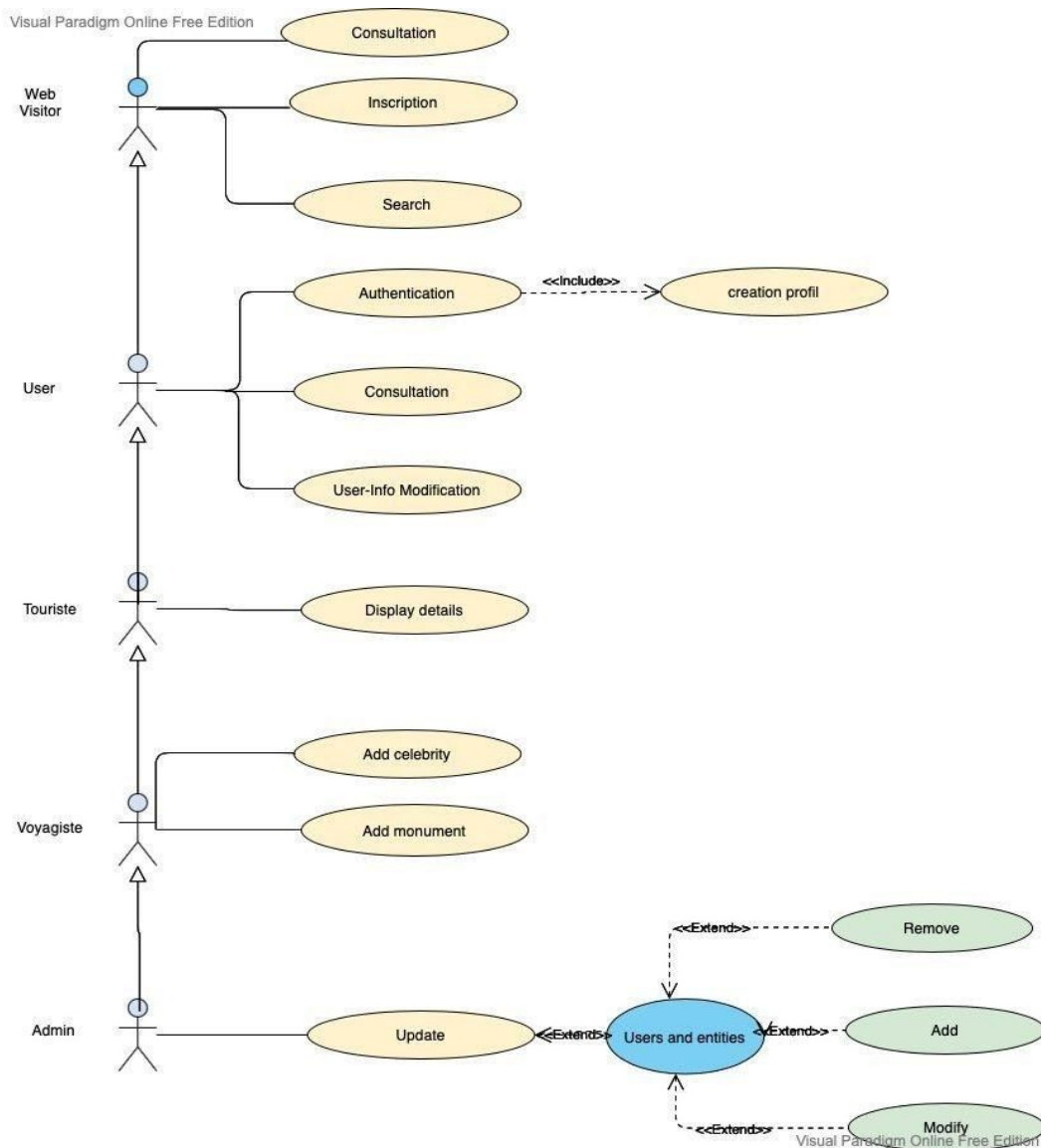
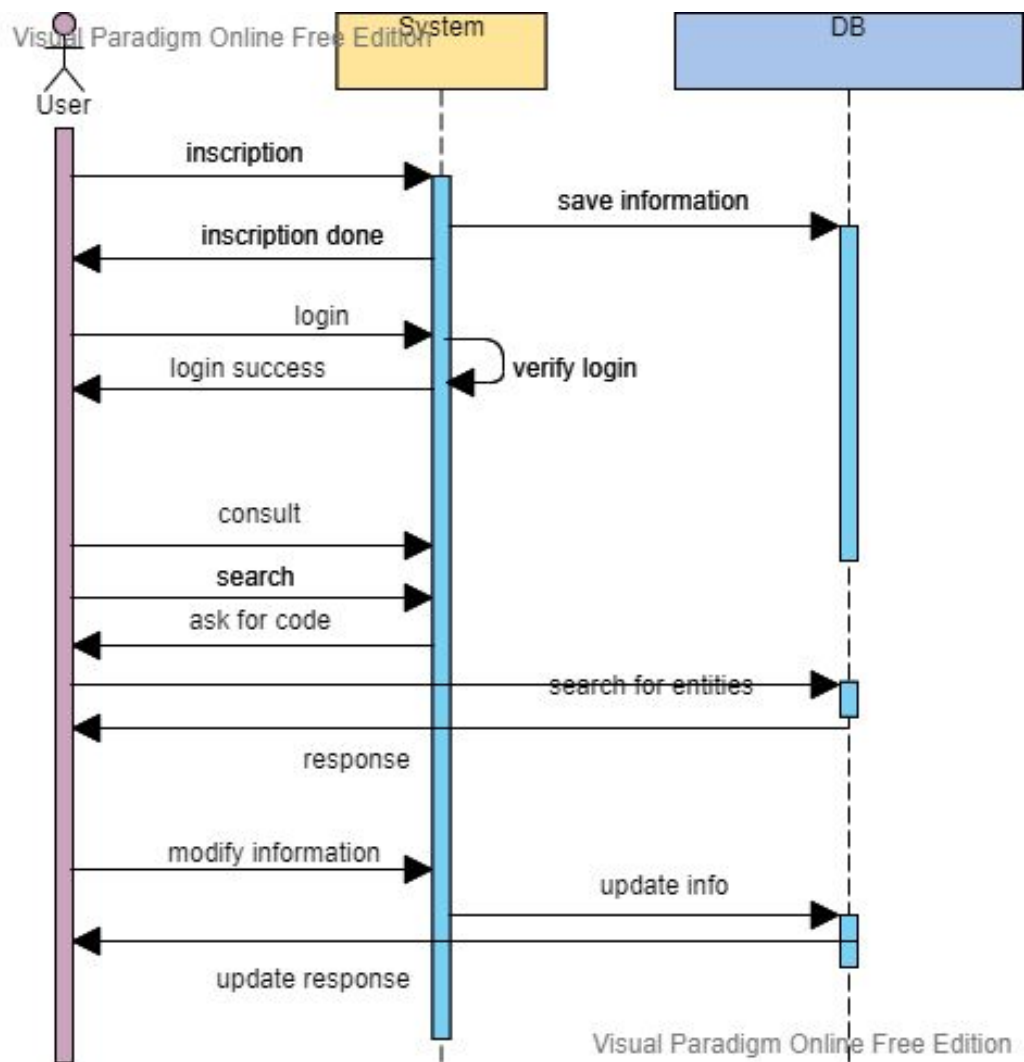


Figure 4. Diagramme de cas d'utilisation.



**Figure 5. Diagramme de séquences.**

## A. Fonctionnalités du relationnel

Pour permettre d'étendre les fonctionnalités de l'application pour une utilisation optimale, les relations entre les tables ont été élaborées pour avoir des liens cohérents entre les objets, et pertinents pour les scénarios d'utilisation.

Pour cela j'ai mis en place des relations entre les différents objets des classes du jeu de donnée que j'ai jugé les plus nécessaires et logiques à avoir pour répondre au besoin de l'utilisateur. Ces relations sont des relations de cardinalité OneToOne, OneToMany et ManyToMany.

**Relation : Departement OneToOne Lieu** : permet de lister pour chaque Departement un seul Lieu (chef-lieu). De même pour la relation **Lieu OneToOne Departement** qui va lier un seul Departement pour chaque Lieu.

**Relation Monument ManyToOne Lieu** : permet de lister plusieurs Monuments appartenant à un seul Lieu (Figure 6).

**Relation Celebrite ManyToMany Monument** permet de lister plusieurs Monuments pour plusieurs Celebrite (figure 7). Une Celebrite peut avoir plusieurs Monuments dédiés, et un seul Monument peut être dédié à plusieurs Celebrite.

```
mysql> desc departement;
```

Field	Type	Null	Key	Default	Extra
num_dep	varchar(4)	NO	PRI	NULL	
nom_dep	varchar(30)	YES		NULL	
parent_url	varchar(255)	YES		NULL	
url	varchar(255)	YES		NULL	
chef_lieu	varchar(5)	YES	MUL	NULL	

5 rows in set (0.00 sec)

Unidirectional relationship  
Department @OneToOne Lieu

```
mysql> desc lieu;
```

Field	Type	Null	Key	Default	Extra
code_insee	varchar(5)	NO	PRI	NULL	
latitude	double	NO		NULL	
longitude	double	NO		NULL	
nom_com	varchar(30)	YES		NULL	
parent_url	varchar(255)	YES		NULL	
url	varchar(255)	YES		NULL	
dep	varchar(4)	YES	MUL	NULL	
departement_url	varchar(255)	YES		NULL	

8 rows in set (0.00 sec)

Unidirectional relationship  
Lieu @OneToOne Department

Figure 6 : Relations des tables Departement et Lieu.

```
mysql> desc monument;
```

Field	Type	Null	Key	Default	Extra
code_m	varchar(12)	NO	PRI	NULL	
latitude	double	YES		NULL	
longitude	double	YES		NULL	
nom_m	varchar(47)	YES		NULL	
parent_url	varchar(255)	YES		NULL	
proprietaire	varchar(10)	YES		NULL	
type_monument	varchar(17)	YES		NULL	
url	varchar(255)	YES		NULL	
code_lieu	varchar(5)	YES	MUL	NULL	

9 rows in set (0.00 sec)

Unidirectional relationship  
Monument @ManyToOne Lieu  
Monument @ManyToMany Celebrité

```
mysql> desc celebre;
```

Field	Type	Null	Key	Default	Extra
num_celebrite	bigint	NO	PRI	NULL	auto_increment
epoque	varchar(4)	YES		NULL	
image	varchar(255)	YES		NULL	
nationalite	varchar(30)	YES		NULL	
nom	varchar(16)	YES		NULL	
parent_url	varchar(255)	YES		NULL	
prenom	varchar(16)	YES		NULL	
url	varchar(255)	YES		NULL	

8 rows in set (0.01 sec)

Unidirectional relationship  
Celebrité @ManyToMany Monument

Figure 7 : Relations des tables Celebrite et Monument.

## B. Les fonctionnalités de l'application

En plus des exigences techniques du projet, j'ai essayé d'ajouter des fonctionnalités que j'ai jugées utiles et qui facilitent et simplifient l'utilisation de l'application à tout utilisateur.

En gros j'ai essayé de donner à l'application une architecture RESTful, donc j'ai commencé à travailler le Backend en RESTful, en réalisant après que je ne peux pas construire un Frontend RESTful avec Thymeleaf, et en étant hésitant à utiliser angular pour ce projet j'ai finalement refait le Backend de façon à ce que ça match le frontend mais par conséquent je n'ai pas fait d'interface pour la partie JSON. Dans cette partie j'ai utilisé les méthodes : POST, DELETE, GET, et PUT.

**Affichage :** J'ai organisé l'affichage des données sous différents formats selectable depuis un menu. L'utilisateur pourra alors afficher les données en format tables, sur carte, en format JSON ou bien XML qui reste à compléter (Figure 8).

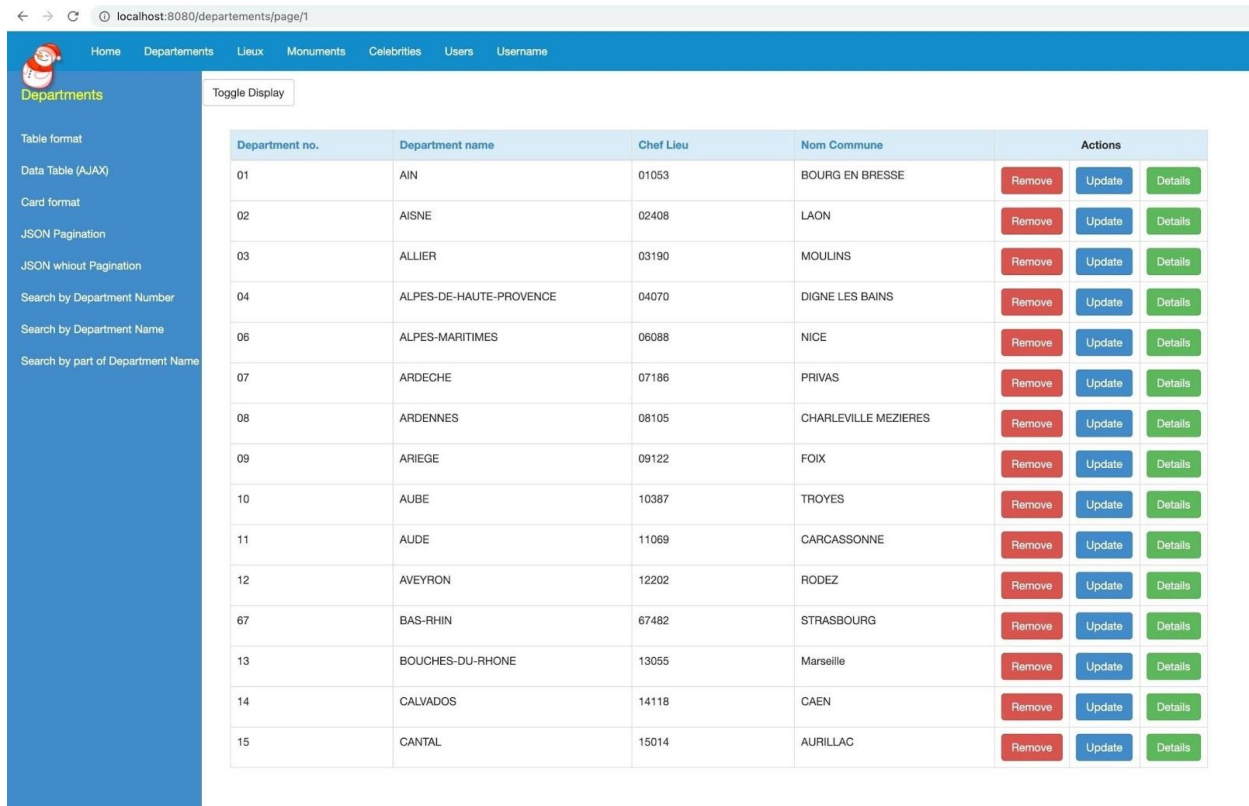
Pour l'affichage des tables j'ai fait en sorte que les éléments de chaque colonne soient organisés par ordre croissant. J'ai également ajouté deux modes d'affichage : avec ou sans pagination qui va organiser l'affichage des données d'abord pour le format JSON ou les données étaient encombrées et non lisibles. De plus, j'ai utilisé @JsonProperty pour renommer les attributs en leur donnant des noms plus compréhensibles. Ceci ajoute de la cohérence et permet de donner une meilleure organisation du code (figure 9-10).

```

- {
  Department Number: "34",
  Department Name: "MONTPELLIER",
- Chef Lieu: {
  Code Insee: "34172",
  Commune Name: "MONTPELLIER",
  Longitude: 43.6108,
  Latitude: 3.87672,
  URL: "http://localhost:8080/json/lieux/34172",
  Lieu URL: "http://localhost:8080/json/lieux/page/1",
  Departement URL: "http://localhost:8080/json/departements/34"
},
  URL: "http://localhost:8080/json/departement/34",
  Department URL: "http://localhost:8080/json/departements/page/1"
},
- {
  Department Number: "56",
  Department Name: "MORBIHAN",
- Chef Lieu: {
  Code Insee: "56260",
  Commune Name: "VANNES",
  Longitude: 47.659749377,
  Latitude: -2.75714329498,
  URL: "http://localhost:8080/json/lieux/56260",
  Lieu URL: "http://localhost:8080/json/lieux/page/1",
  Departement URL: "http://localhost:8080/json/departements/56"
},
},

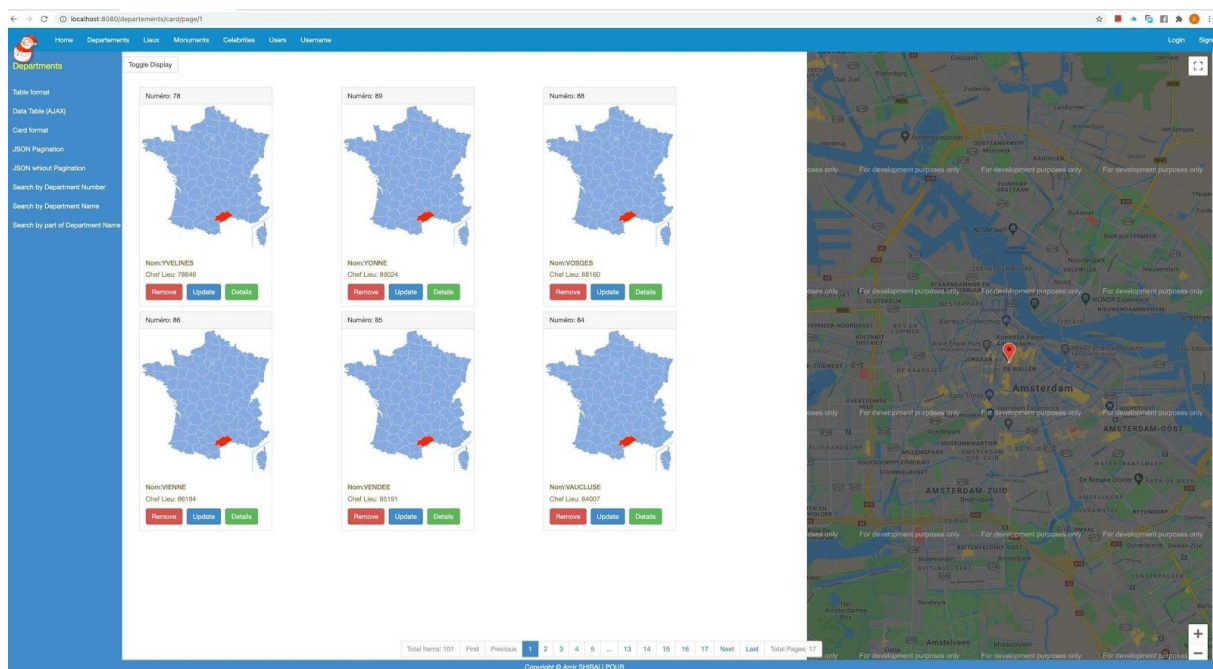
```

Figure 8. Données format JSON



Department no.	Department name	Chef Lieu	Nom Commune	Actions
01	AIN	01053	BOURG EN BRESSE	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
02	AISNE	02408	LAON	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
03	ALLIER	03190	MOULINS	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
04	ALPES-DE-HAUTE-PROVENCE	04070	DIGNE LES BAINS	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
06	ALPES-MARITIMES	06088	NICE	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
07	ARDECHE	07186	PRIVAS	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
08	ARDENNES	08105	CHARLEVILLE MEZIERES	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
09	ARIEGE	09122	FOIX	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
10	AUBE	10387	TROYES	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
11	AUDE	11069	CARCASSONNE	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
12	AVEYRON	12202	RODEZ	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
67	BAS-RHIN	67482	STRASBOURG	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
13	BOUCHES-DU-RHONE	13055	Marseille	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
14	CALVADOS	14118	CAEN	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>
15	CANTAL	15014	AURILLAC	<a href="#">Remove</a> <a href="#">Update</a> <a href="#">Details</a>

Figure 9. Affichage format table ordre croissant.



**Figure 10. Affichage des données au format carte.**

**Pagination** : la pagination des données en format table qui va afficher : - toutes les pages disponibles - première page - dernière page - prochaine page - page précédente. Et bien sûr on peut cliquer sur chaque page séparément (figure 11).



**Figure 11. Fonctionnalités de paginations.**

**Recherche** : J'ai fait une fonctionnalité qui permet de trouver/rechercher les données à afficher selon leur ID, leur nom et une partie du nom. La partie Backend est achevée mais il me reste à finaliser la partie Frontend pour que ça soit totalement fonctionnel (Figure 12).





```

[
  {
    Department Number: "60",
    Department Name: "OISE",
    Chef Lieu: {
      Code Insee: "60057",
      Commune Name: "BEAUVAIS",
      Longitude: 49.436552332,
      Latitude: 2.08616123661,
      URL: "http://localhost:8080/json/lieux/60057",
      Lieu URL: "http://localhost:8080/json/lieux/page/1",
      Departement URL: "http://localhost:8080/json/departements/60"
    },
    URL: "http://localhost:8080/json/departement/60",
    Department URL: "http://localhost:8080/json/departements/page/1"
  },
  {
    Department Number: "95",
    Department Name: "VAL-D'OISE",
    Chef Lieu: {
      Code Insee: "95500",
      Commune Name: "PONTOISE",
      Longitude: 49.051373785,
      Latitude: 2.09487928948,
      URL: "http://localhost:8080/json/lieux/95500",
      Lieu URL: "http://localhost:8080/json/lieux/page/1",
      Departement URL: "http://localhost:8080/json/departements/95"
    },
    URL: "http://localhost:8080/json/departement/95",
    Department URL: "http://localhost:8080/json/departements/page/1"
  },
  {
    Department Number: "38",
    Department Name: "ISERE",
    Chef Lieu: {
      Code Insee: "38185",
      Commune Name: "GRENOBLE",
      Longitude: 45.1821,
      Latitude: 5.72133,
      URL: "http://localhost:8080/json/lieux/38185",
      Lieu URL: "http://localhost:8080/json/lieux/page/1",
      Departement URL: "http://localhost:8080/json/departements/38"
    },
    URL: "http://localhost:8080/json/departement/38",
    Department URL: "http://localhost:8080/json/departements/page/1"
  }
]

```

Figure 12. Résultat de recherche pour le mot “ise”.

**Page multitask:** J’ai fait en sorte que le contenu des pages “details”, “Add” et “Update” soient pareils. Donc j’ai utilisé une seule page “update.html” pour faire les trois actions.

**Modular fragments:** J’ai utilisé les différents fragments d’une façon modular. Par exemple le morceau du code `<th:block th:fragment="remove-update (entity, id)">`, ajout deux boutons dans notre code, et si j’ajoute le mot “details”, `<th:block th:fragment="remove-update-details (entity, id)">`, sans changer le reste du code, j’aurai trois button qui marche. Pareil si on ajoute le mot “back” ou “td”, on aura le bouton “back” ou on aura tous ses boutons dans le format “td” de table, sans avoir à réécrire tout le code.

Ex:

```

<th:block th:fragment="remove-update-details-td (entity, id)">
<th:block th:fragment="remove-update-details-back (entity, id)">

```

C’est assez cool comme fonction car ça me fait gagner en temps et en ligne de code.



### C. Autres fonctionnalités

- J'ai ajouté une fonctionnalité "color()", qui permet de changer les couleurs dans l'affichage de données après chaque actualisation de la page (figure 13). De plus, j'ai développé la partie backend pour permettre d'avoir la distance entre deux Monuments dans une seule ou deux villes différentes. Ainsi que l'affichage des tous les Monuments et Celebrite disponible dans une seule région donnée.

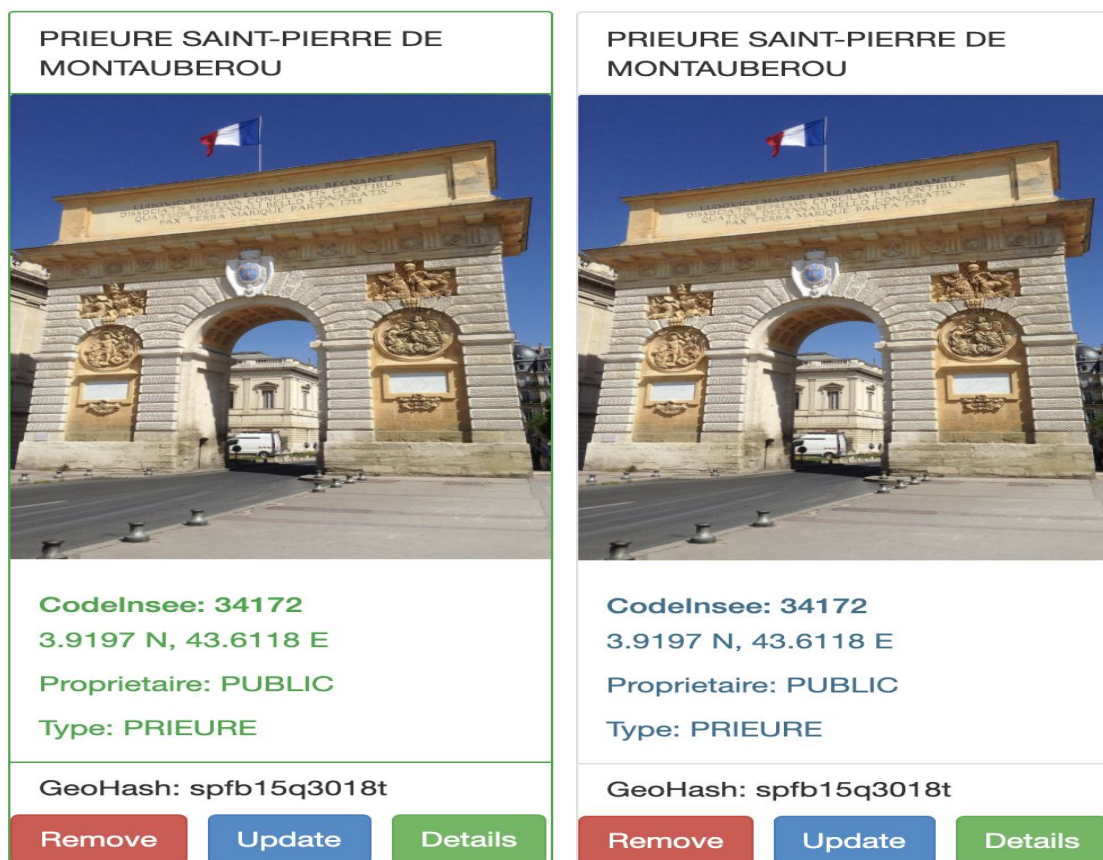


Figure 13. Changement de couleur des données affichées.

- J'ai mis en place une fonctionnalité qui gère les erreurs : si on a une erreur elle revoie les exceptions dans classes EntitiesNotFoundException qui étend RuntimeException, celle-ci va agir dans le cas où on ne trouve pas une entité et elle envoie un message avec l'ID non trouvée. Dans le cas d'erreur "object in object", elle envoie un message d'erreur approprié. J'ai également utilisé **Toast Message** pour envoyer des messages d'erreurs ou de succès à l'utilisateur. Et également, en cas d'entrée d'une URL non valide on aura une page d'erreur.

La deuxième classe `NotFoundAdvice @ControllerAdvice` renvoi un message personnalisable selon le type d'erreur

- `REST.rest` un outil contenant toutes les routes de la partie backend de RESTful API permet d'examiner les méthodes et les routes utilisées. J'ai aussi organisé l'arborescence des fichiers de façon compréhensible.
- J'ai aussi ajouté des types d'utilisateurs selon les rôles : Admin, Voyageur, Touriste, (figure 14). Seulement Admin a accès aux données de tous les utilisateurs. Et enfin la suppression d'un objet conduit à la suppression de tous les objets qui y sont liés. Par exemple, si on efface un Département, tous les Lieux qui sont dans ce Département seront effacés .

### Amir's Profile



Amir SHIRALI POUR

	Email	amir@gmail.com
	Role	[ROLE_ADMIN]
	Active	<input checked="" type="checkbox"/>

[Remove](#) [Back](#) [Update](#)

**Figure 14. Données utilisateur.**

## 5. Les problèmes rencontrés

Comme pour tout processus de conception de logiciels, j'ai rencontré quelques problèmes techniques en plus de la contrainte du temps et la réalisation de plusieurs projets en une même période.

- Premièrement j'ai eu des difficultés lors de la manipulation de Thymeleaf vu que c'est la première fois que je l'utilise, ça m'a donc pris un moment à m'y habituer.
- Un conflit entre les différentes applications utilisées de plus d'être ma première manipulation de certaines d'elles, des problèmes qui surgissent en pleine conception m'ont souvent freiné.
- Un problème d'actualisation pour l'application Eclipse et Spring Tool Suite m'a bloqué pour un moment, en effet l'interaction lors de la conception entre ces outils ne se faisait pas convenablement cela été à cause de la nécessité d'actualiser à chaque manipulation.
- En gestion de la base de données, j'ai rencontré quelques "problèmes" lors de l'ajout/suppression des données de tables à cause des contraintes posées, il fallait donc les gérer ce qui a consommé beaucoup de temps du projet.
- Des difficultés liées à l'ajout d'un objet dans un autre. Les deux tables Lieu et Departement ont des dépendances mutuelles qui est difficile à gérer lors de la création d'une nouvelle instance. Dans le formulaire d'objet Lieu si on rentre le codeInsee du Lieu ça va vérifier l'existence de ce Lieu, si ce Lieu existe il va récupérer ses données et l'ajouter dans Departement. Par contre si le Lieu n'existe pas ça va créer un nouveau Lieu avec le codeInsee entré par l'utilisateur, puis il va l'ajouter dans l'objet Departement.

## **6. Conclusion**

En gros ce projet a été bien adapté pour appliquer la programmation orienté objet ainsi que l'implémentation de l'aspect relationnel et lié les deux concepts tout en manipulant une base de données.

La conception du projet m'a apporté une idée de comment adapter les données et manipuler les outils en main pour arriver à construire un produit efficace et facilement utilisable par les utilisateurs.

Enfin ce projet m'a permis de découvrir de nouveaux outils de programmation qui facilitent les tâches.

Ce qui pourrait améliorer l'application serait d'utiliser Google API pour trouver les coordonnées géographiques, de finir l'implémentation de la table AssocieA et de la mettre en utilisant Angular en FrontEnd à la place de Thymeleaf. Utiliser AJAX pour récupérer les données liées d'une nouvelle entité entrée dans le formulaire d'ajout et de modification, et si l'objet est trouvé il affichera instantanément ses informations.

## 7. Annexes

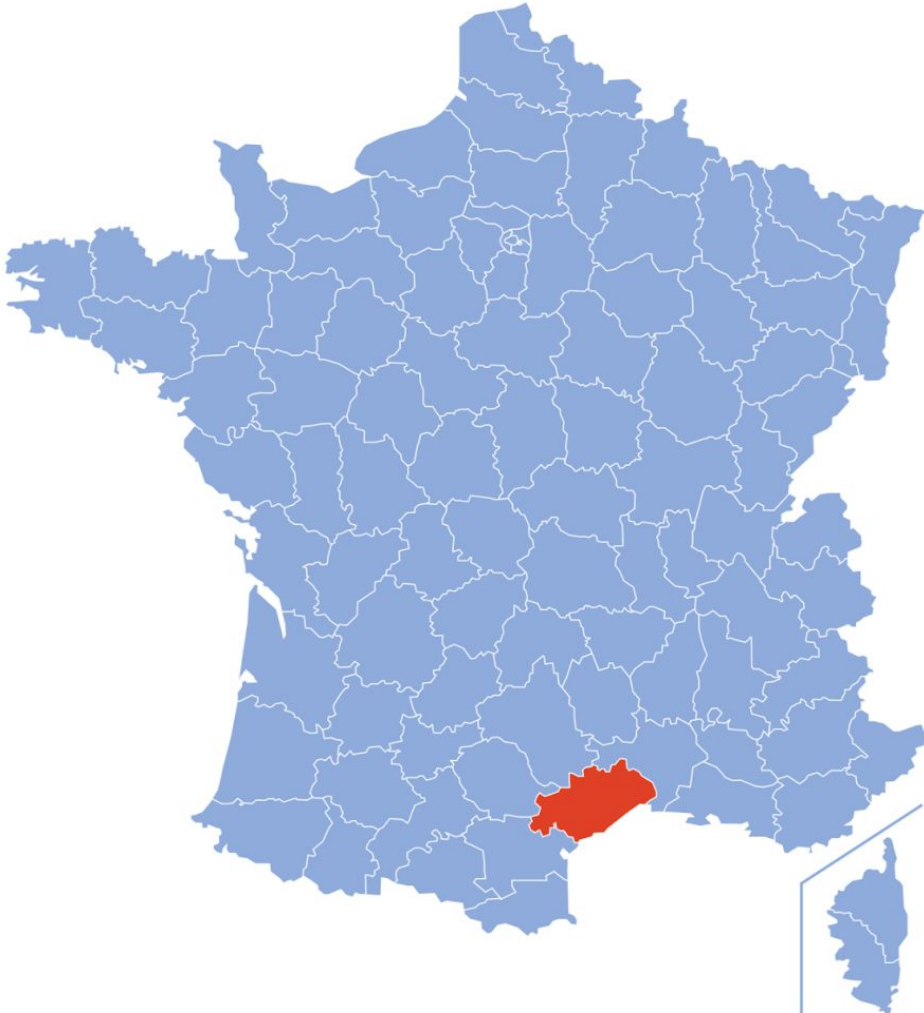
```
mysql> select * from user;
+-----+-----+-----+-----+-----+-----+
| email          | first_name | image          | is_active | last_name    | password |
+-----+-----+-----+-----+-----+-----+
| amir@gmail.com | Amir      | /image/amir.jpg | 0x01      | SHIRALI POUR | $2a$10$IIfjQWVljayFWvPh/CO.0Ig3AidwJY05t5I81gxro1zAnLacLWu |
| fahima@gmail.com | Fahima    | /image/fahima.jpg | 0x01      | MOKHNACHE    | $2a$10$rJD6T/bxtvkdKxId/JDDv0YTUDxT9JsC4hiPJPhzMzGCpkZT1P/xu |
| hassina@gmail.com | Hassina   | /image/hassina.jpg | 0x01      | BOUFATIS     | $2a$10$sVD9MbV9NBiuvp8Xdt6010uZU8zbdSFPP0figMb9M9TcFG8TK/fI. |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from role;
+----+-----+
| id | name      |
+----+-----+
| 1  | ROLE_ADMIN |
| 2  | ROLE_VOYAGISTE |
| 3  | ROLE_TOURISTE |
+----+-----+
3 rows in set (0.00 sec)

mysql> select * from users_roles;
+-----+-----+
| user_email | role_id |
+-----+-----+
| hassina@gmail.com | 3 |
| fahima@gmail.com | 2 |
| amir@gmail.com | 1 |
+-----+-----+
3 rows in set (0.00 sec)
```

Figure 15 : Données des tables user, role et users\_roles.

## Details Departement with ID: 100



**Name:**

**Number:**


**Chef Lieu:**

**URL:**

**Parent URL:**

Figure 16 : Ajout d'un nouveau Departement.

# Details Lieu with ID: 99999



**Name:**

**CodeInse:**

**Departement:**

**Latitude:**

**Longitude:**


**URL:**

**Parent URL:**

Figure 17 : Ajout d'un nouveau Lieu.



## Details Monument with ID: newCodeMonum



**Name:**

**Code Monument:**

**Lieu:**

**Latitude:**

**Longitude:**

**Proprietaire:**

**Type:**

**URL:**

**Parent URL:**

Figure 18 : Ajout d'un nouveau Monument.



## Details Celebrite with ID: 1887560983



/image/Saint\_Roch.JPG

**Number:** 1887560983

**Name:** Saint Roch

**Family:** Rocco

**Nationality:** Française

**Year of Birth:** 1295

**URL:** <http://localhost:8080/json/celebrities/1887560982>

**Parent URL:** <http://localhost:8080/json/celebrities/page/1>

Remove

Update

Figure 19 : Ajout d'un nouveau Celebrite.