Développement d'Applications Réticulaires

M2 STL 2016/2017

Projet MusExpress

Auteurs
Ouiza Bouyahiaoui
Morane Gruenpeter
Lina Yahi

Responsable :
Romain Demangeon

13 novembre 2016

Table des matières

Table des figures

Introduction

Dans le cadre de l'UE Développement d'Applications Réticulaires (DAR), nous avons réalisé l'application Web « MusExpress » permettant aux utilisateurs franciliens de connaître le temps d'attente actuel des musées. La force de cette application réside dans le fait que les utilisateurs fournissent eux-mêmes les affluences durant leur propre visite. Il s'agit donc d'une solution communautaire interactive permettant aux utilisateurs de gagner du temps au quotidien.

Par ailleurs, l'application offre aux utilisateurs des suggestions de visites de musées qui s'accordent avec la météo actuelle, leur permettant d'éviter des files d'attente interminables dans des conditions météorologiques défavorables (soleil de plomb, pluie torrentielle...).

L'objectif du projet est de développer une application web en utilisant des API en Île-de-France sous une architecture client-serveur en se servant d'appels simultanés avec un traitement côté serveur.

Nous allons présenter l'application web, sa conception et son implémentation. Le premier chapitre est dédié à la conception de l'application, suivant les cas d'utilisation possibles. Le deuxième chapitre introduira l'architecture de l'application et les technologies que nous avons choisies. Enfin nous décrirons les résultats de notre travail et les perspectives de l'application MusExpress.

Chapitre 1

Manuel d'utilisation

1.1 Description de l'interface

L'utilisateur accède au site MusExpress où il peut se connecter ou créer un profil.

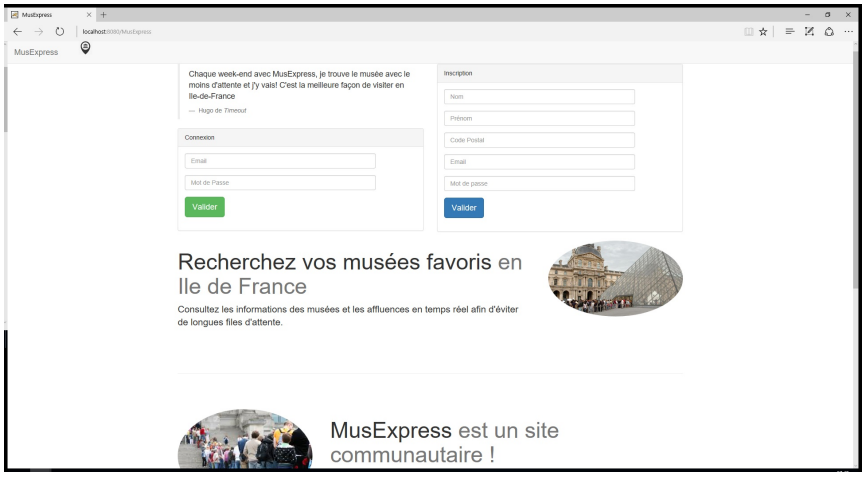

FIGURE 1.1 – Page d'inscription et de connexion, index.html

Une fois connecté, l'utilisateur a accès à une page complètement configurée pour ses besoins. Sur cette page, on retrouve les propositions classées en fonction des données météorologiques ou d'affluences, les prévision météorologique du jour ainsi qu'un champ de recherche par nom de musée.

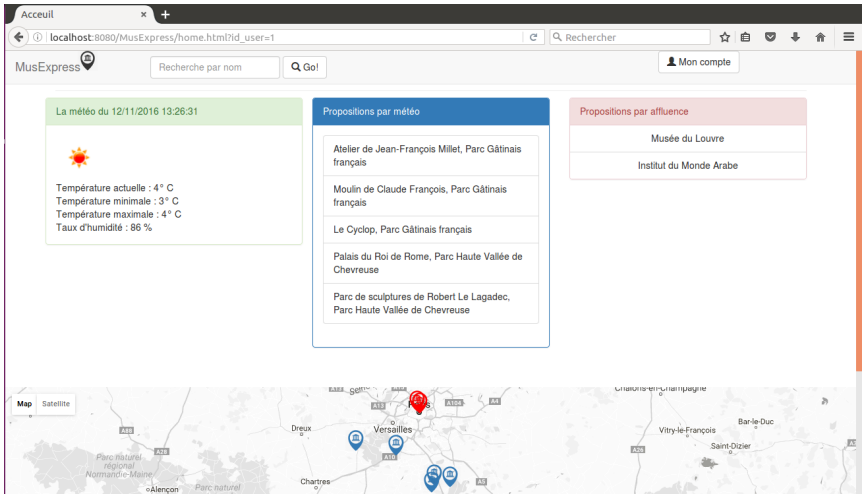

FIGURE 1.2 – Page d'accueil home.html

L'utilisateur peut également accéder à son compte. Il peut y visualiser ses musées favoris et modifier son compte.

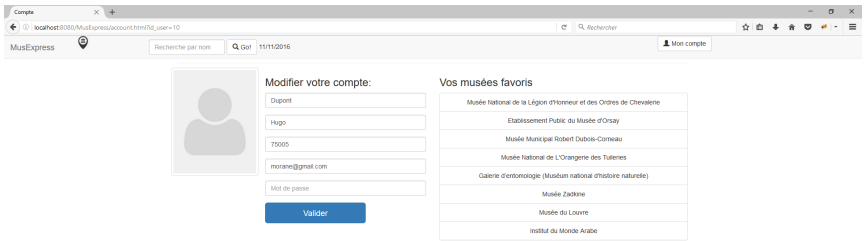

FIGURE 1.3 – Le compte de l'utilisateur Hugo Dupont sur account.html

A partir de n'importe quelle page de notre application, l'utilisateur peut chercher un musée en introduisant le nom (ou partie du nom) du musée.

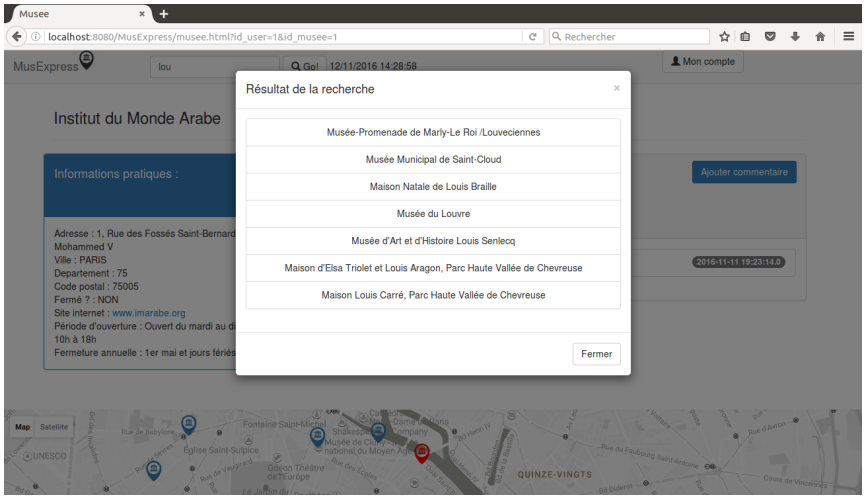

FIGURE 1.4 – Recherche d'un musée

lorsque l'utilisateur consulte un musée une page affiche les détails le concernant, sa localisation, les musées à proximité et les derniers commentaires relatif à son affluence. La figure suivante représente la page du musée du Louvre :

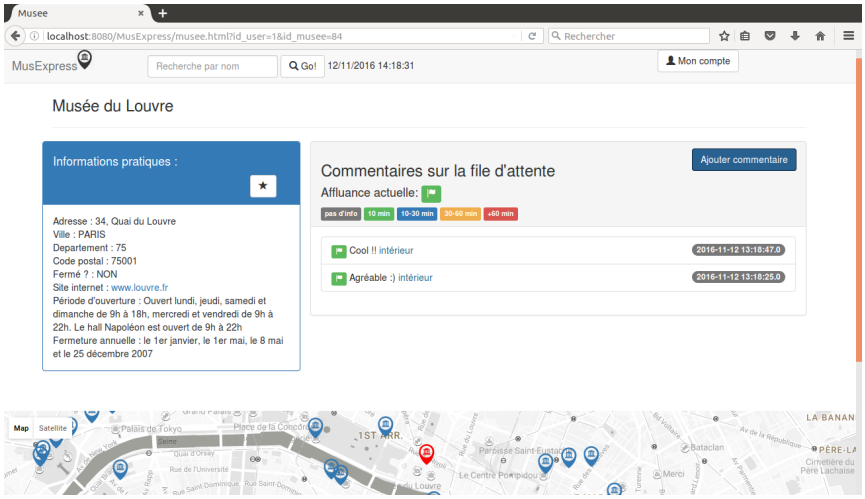

FIGURE 1.5 – La page du musée du Louvre musee.html

L'utilisateur peut ajouter le musée à sa liste de musées favoris en appuyant sur le bouton contenant une étoile. Il peut également ajouter un commentaire en appuyant sur le bouton "Ajouter commentaire". La fenêtre ci-après s'ouvre afin de lui permettre de faire cet ajout.

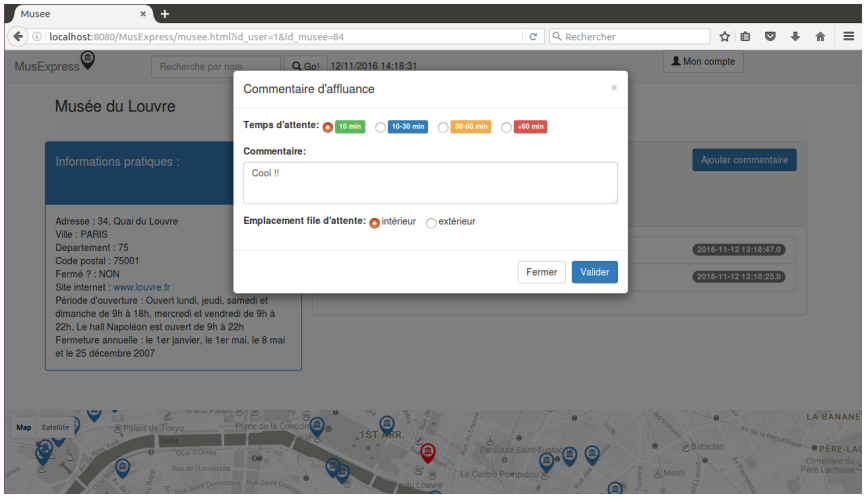

FIGURE 1.6 – Ajouter un commentaire à partir de musee.html

1.2 Description des cas d'utilisation

1.2.1 Créer un profil

Hugo consulte la page MusExpress et choisit de créer un profil. Dans un formulaire il remplit les champs suivants : nom, prénom, code postal, e-mail et mot de passe. Puis il valide le formulaire. Il arrive sur la page d'accueil de MusExpress.

1.2.2 Connexion

Hugo consulte la page MusExpress et choisit de se connecter. Dans un formulaire, il remplit les champs suivants : mail et mot de passe. S'il a précédemment créer un compte avec ces identifiant, il sera redirigé vers la page d'accueil.

1.2.3 Consulter profil

Toutes les pages, Comportent une icône de profil en haut de la page (Mon compte) permettant d'accéder à son profil. Une page affichant ses informations et ses musées favoris (correspondant au cas d'utilisation consulter musées favoris) s'ouvre.

1.2.4 Consulter les suggestions (par affluence/par météo)

Hugo vient de se connecter au site MusExpress et se trouve sur la page d'accueil. Il trouve deux listes de propositions de musées à visiter : une liste des musées avec une courte file d'attente (proposition par affluence) et une liste de musées à visiter selon la météo du moment. En sélectionnant un musée, Hugo est orienté vers la page du musée.

1.2.5 Consulter la météo

Sur la page d'accueil, Hugo peut connaître la météo du moment en consultant le panel se trouvant en haut à gauche.

1.2.6 Rechercher un musée

Hugo recherche un musée pour une éventuelle visite. Il se trouve sur n'importe quelle page de MusExpress. Il entre le nom du musée et valide, il reçoit une liste de musées qui correspondent à sa recherche. Il sélectionne un musée et le consulte (correspond au cas d'utilisation consulter un musée).

1.2.7 Consulter un musée

Hugo consulte un musée sur la page du musée. Sur cette page s'affiche la description du musée ainsi que les horaires d'ouverture et de fermeture et sa localisation.

1.2.8 Ajouter un commentaire d'affluence

À partir de la page d'un musée, l'utilisateur clique sur un bouton "ajouter commentaire". Il renseigne le temps d'attente approximatif du musée ainsi que le type de file d'attente (intérieure ou extérieure). Il peut éventuellement accompagner son choix d'un commentaire.

1.2.9 Consulter l'affluence d'un musée

Hugo prend connaissance de l'affluence actuelle d'un musée qui est représentée par un drapeau de couleur : verte pour une fréquentation faible (moins de 10 minutes), bleue pour une fréquentation moyenne (10-30 minutes), orange pour une fréquentation soutenue (30-60 minutes) et rouge pour une forte fréquentation (+60 minutes). Si toutefois aucune information sur l'affluence n'a été renseigné par l'utilisateur, un drapeau gris sera affiché.

Sous la rubrique affluence sont affichés les commentaires des internautes triés par ordre décroissant à partir du plus récent.

1.2.10 Consulter les musées à proximité

Sur la page du musée sélectionné, et plus précisément sur la carte de localisation, Hugo consulte tous les musées à proximité du musée sélectionné. Le musée recherché est alors représenté d'un marqueur rouge et les musées se trouvant à proximité en marqueurs bleus.

Chapitre 2

Architecture et technologies mises en œuvre

2.1 Architecture globale de l'application

Le protocole HTTP est utilisé pour la communication entre la partie client et la partie serveur. Les requêtes envoyées du client vers le serveur sont capturées par les contrôleurs. Le contrôleur interrogé appelle le service associé à la requête qui va consulter la base de données par l'intermédiaire des composants DAO. La couche service joue le rôle d'intermédiaire entre la couche DAO et les servlets. Ainsi elle permet d'abstraire les appels aux traitements d'accès à la base de données. Les dao vont alors interroger la base de données et retourner un résultat en objet Java. Ces objets Java vont être transformés en objet JSON par le service et renvoyés au contrôleur qui envoie ce contenu au client.

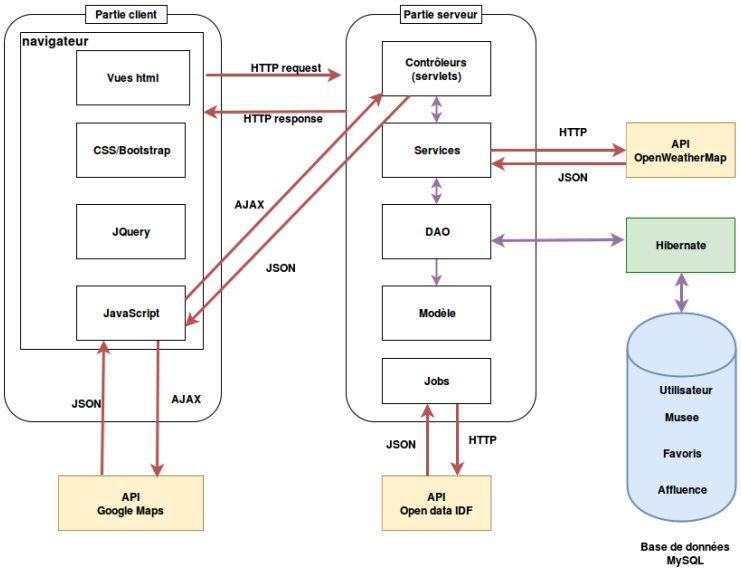

FIGURE 2.1 – L'architecture globale de l'application

2.1.1 La communication entre côté client et côté serveur

Nous avons utilisé Ajax pour la communication entre le client et le serveur ce qui permet de rendre les interactions asynchrones. La servlet renvoie des données en JSON qui vont être traitées et affichées dans les vues.

Exemple d'une requête Ajax pour l'affichage d'un musée

La fonction readMusee fait un appel GET au serveur (ConsulterMuseeServlet) en passant en paramètre l'identifiant du musée idMusee. Si l'appel ajax aboutit, le paramètre success prend en charge une fonction qui sera exécutée. Cette fonction récupère l'objet musée retourné et affiche ses informations sur la page musee.html.

```
/**
**Afficher les informations du musee
**/
function readMusee(idMusee)
{
  $.ajax
  ({
    type: "GET",
    url : "ConsulterMuseeServlet",
    data : {id : idMusee},
    dataType : 'JSON',
    success : function(data)
    {
```

```
14        var museeFromDB = data.musee;
15        if (resultat.message==1)
16        {
17          musee.setmusee(museeFromDB);
18        }
19      },
20      error : function(XHR, testStatus, errorThrown)
21      {
22        console.log("status: " + XHR.status + ", erreur: " + XHR.
            responseText);
23      }
24    });
25  }
```

La navigation dans l'application

La navigation dans l'application se fait avec un objet routeur défini en javascript. Celui-ci récupère les paramètres dans l'URL et assure le déplacement d'une page à une autre.

Nous avons créé 4 pages différentes :

1. index.html
2. home.html
3. musee.html
4. account.html

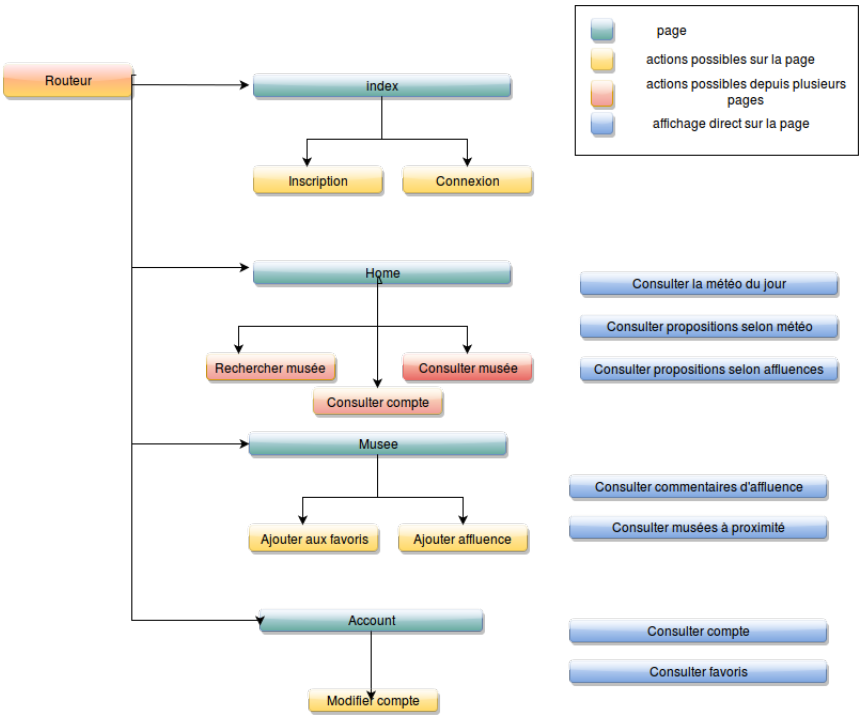

FIGURE 2.2 – La navigation dans MusExpress

Au lancement de l'application, on accède à la page index.html. Après inscription ou connexion, on passe à la page home.html avec notre *id_user* en paramètre pour pouvoir afficher le contenu adapté à notre profil. Puis nous avons le choix à partir de cette page d'aller vers notre compte sur account.html ou de continuer la navigation en allant sur la page d'un musée sur musee.html. L'affichage statique est écrit en html et le contenu dynamique est injecté après que le serveur réponde à nos requêts ajax.

```
1  var routeur = {
2      idUser: "",
3
4      index: function(){
5        window.location=('index.html');
6      },
7
8      init: function(){
9        this.idUser = GetURLParameter('id_user');
10     },
11
12     home: function(id_user){
13       this.idUser = id_user;
14       if(this.idUser=="" || this.idUser==undefined){
15         this.index();
16       }else{
17         window.location=('home.html?id_user='+this.idUser);
```

```
18        }
19      },
20
21      account: function(){
22        window.location=('account.html?id_user='+GetURLParameter('id_user')
            );
23
24      },
25
26      musee: function(idMusee){
27        window.location=('musee.html?id_user='+this.idUser+'&id_musee='+
            idMusee);
28
29      },
30  }
```

2.1.2 Base de données : MYSQL

Nos entités étant liées, nous avons opté pour une base de données relationnelle. MySQL permet de structurer les informations et de les réutiliser facilement. De plus, son interface d'administration phpMyAdmin facilite le travail.

La figure suivante représente le schéma relationnel de notre base de données :

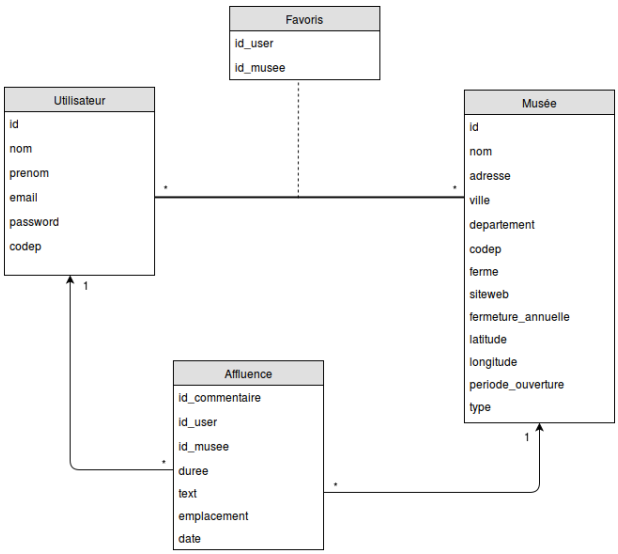

FIGURE 2.3 – Le schéma relationnel de l'application

2.1.3 ORM : Hibernate

Object-relational mapping est une technique de programmation de conversion des données d'un système orienté objet à une base de données relationelle. Hibernate est un ORM framework pour Java qui permet la gestion de la persistance des objets dans des bases de données relationnelles. Nous avons jugé intéressant de nous y familiariser et de l'utiliser dans le développement de notre application. L'utilisation d'Hibernate se fait en trois temps :

1. La création des classes qui encapsulent les données (dans notre architecture ce sont les models)
2. Le mapping entre les classes objets et les tables en base de données soit par la création d'un fichier de correspondance en format XML appelé *mapping file* (comme c'est les cas pour les classes User et Musee), soit en ajoutant les annotations de mapping directement sur les models (comme pour la classe Affluence).
3. Spécifier la configuration d'hibernate dans un fichier nommé hibernate.cfg.xml

2.1.4 Les API utilisées

Les API (*Application Programming Interface*) suivantes nous ont permis de remplir notre base de données et créer une application beaucoup plus ergonomique et intéressante.

— Premièrement un cache de l'API **Open Data Île-de-France** (Liste des musées franciliens et Parcs Naturels Régionaux d'Île-de-France) : pour l'implémentation nous appelons cette API pour remplir la table des lieux (musées et parcs naturels) que nous proposons comme contenu à nos utilisateurs.
Nous avons donc décidé de faire seulement un appel à l'API au moment du déploiement afin d'éviter des appels trop fréquents ce qui permet d'éviter des échecs de connexion et ainsi, assurer aux utilisateurs un accès complet aux données.
Par ailleurs, cette API n'étant pas régulièrement modifiée (dernière modification en 2013), nous n'avons pas créé de tâche crône qui permettrait la mise à jour de ces données.

— L'API **Google Maps** est utilisée directement à partir du navigateur (javascript). En utilisant la liste des musées à proximité et leurs localisation (latitude,longitude) on lance un appel à l'API Google Maps. La fonction initMap() affiche la carte sur la page avec les marqueurs du musée actuel et des musées à proximité.

— L'API **OpenWeatherMap** : la servlet GetMeteoServlet fait appel à cette API qui retourne des données en JSON sur la météo du moment. Ces données sont utilisées afin d'afficher à l'utilisateur la météo du moment. Grâce à ces données nous proposons des musées à visiter. En effet, s'il fait beau, MusExpress propose à l'utilisateur une liste de parcs et/ou de musées dont la file d'attente est extérieure (donnée recueillie à partir des commentaires des utilisateurs). Si par contre le temps

est mauvais, une liste de musées dont la file d'attente est intérieure est proposée afin de lui éviter une attente dans des conditions météorologiques inadéquates.

2.2 Modèle MVC

L'architecture du système est divisée en trois parties selon l'architecture du modèle MVC[1].

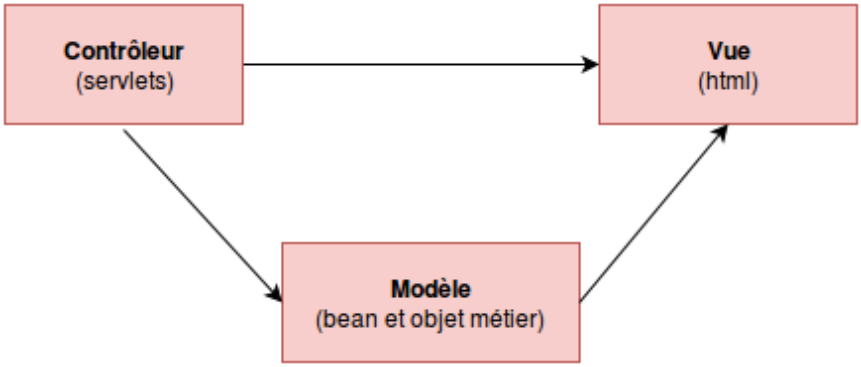

FIGURE 2.4 – Le modèle MVC

La première partie, le modèle, s'occupe de la gestion des données. Il représente les **Beans** qu'on manipule à savoir User, Musee et Affluence et les objets métiers associés : UserDao, MuseeDao et AffluenceDao.

La deuxième partie, la vue, définit le côté client de l'application web. Cette partie n'a aucun contrôle sur le système. Néanmoins la navigation entre les vues et l'interaction entre l'utilisateur et le système sont définies par des objets **Javascript**.

La troisième partie, les contrôleurs, représente nos **Servlets** qui se chargent de mettre à jour le modèle et les vues.

Nous avons opté pour une approche basée service où chaque servlet est associée à un service. Le tableau ci-dessous fait l'association entre les fonctionnalités et le découpage en modèle MVC de notre application.

1. MVC : Modèle-Vue-Contrôleur est une architecture permettant de concevoir des applications interactives. Le modèle représente les données de l'application, la vue représente l'interface d'interaction entre l'utilisateur et l'application, et le contrôleur représente le gestionnaire des événements qui se produisent sur l'interface.

Table 2.1 – Découpage en modèle MVC

Fonctionnalités	Servlet	DAO	Page
créer profil	InscriptionServlet	UserDAO	index.html
connexion	LoginServlet	UserDAO	index.html
consulter profil	ConsulterUserServlet	UserDAO	account.html
modifier profil	UpdateUserServlet	UserDAO	account.html
ajouter musée aux favoris	AjoutMuseeFavServlet	UserDAO	musee.html
consulter musées favoris	AfficherFavorisServlet	UserDAO	account.html
rechercher musée	RechercheMuseeServlet	MuseeDAO	home.html
consulter musée	ConsulterMuseeServlet	MuseeDAO	musee.html
ajouter commentaire affluence	AjoutAffluanceServlet	AfluenceDAO	musee.html
consulter commentaires affluence	AfficherAffluanceServlet	AffluenceDAO	musee.html
consulter musées à proximité	AfficherMuseesProximiteServlet	MuseeDAO	musee.html
propositions par affluence	AfficherPropAffServlet	AffluenceDAO	home.html
propositions par météo	AfficherPropMeteoServlet		home.html
consulter météo	GetMeteoServlet		home.html

2.3 Tester la communication côté client

Le plus important à tester au moment du déploiement sont les réactions des **Servlets** et de s'assurer que la communication entre le côté client et le côté serveur se passe bien. C'est pourquoi nous avons utilisé un framework de test **Qunit** pour les tests en javascript. Dans la page web test.html présentée ci-après, le nom du test indique la Servlet testée et result le résultat du test. En rouge sont les tests qui ont échoué. Certains sont définis pour échouer, comme l'inscription d'un utilisateur qui existe déjà en base de données.

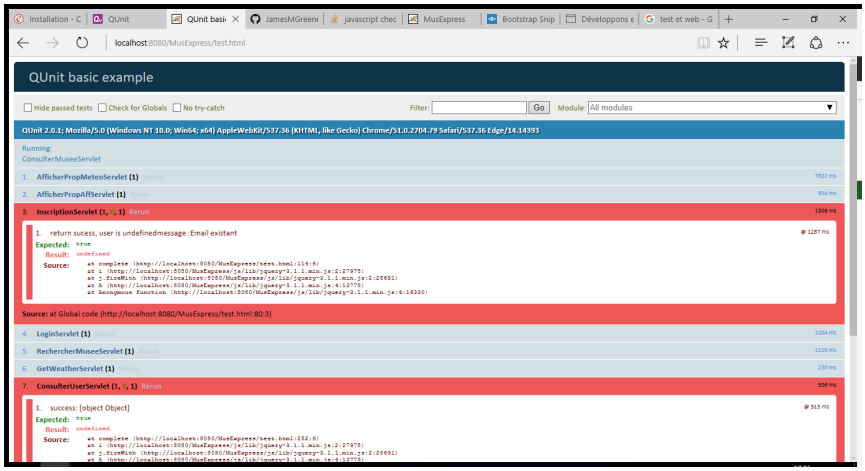

FIGURE 2.5 – Les tests sur test.html

2.4 Méthode de développement agile

Une méthode de développement agile est un ensemble de principes et de pratiques qui aident les équipes de développement à livrer des produits à cycles courts, ce qui permet une rétroaction rapide, une amélioration continue et une adaptation rapide au changement.

Nous avons développé notre projet en essayant de suivre au mieux la méthodologie Scrum en effectuant des « sprints » de deux semaines. A cet effet, nous avons utilisé :

1. L'application **Trello** pour la planification de nos sprints et l'organisation du projet.
2. L'application **Slack** pour la communication au sein du groupe.
3. Le logiciel de gestion de versions **GitHub** qui facilite le travail en équipe sur un même projet.
4. L'application en ligne **Overleaf** pour travailler en simultanée sur la rédaction du rapport en **LaTeX**.

Chapitre 3

Résultats et perspectives

3.1 Points forts

Parmi les avantages que présente notre application, nous pouvons citer :

— **l'interface** : nous avons conçu une interface ergonomique avec **Bootstrap** afin de faciliter l'utilisation.

— **l'architecture** : l'application est facilement maintenable grâce à l'utilisation d'une architecture MVC.

— **la base de données relationnelle** : choix de MySql pour assurer la persistance et l'intégrité des données .

— l'utilisation de plusieurs **API externes** : l'utilisation de l'*API Google Maps* et de L'*API OpenWeatherApi* pour une meilleure expérience utilisateur (UX). Ainsi et l'**API Open Data Ile-de-France** pour remplir la base de données.

— **l'agilité** : développer cette application en agile nous a donné une grande visibilité sur l'avancement du projet.

Néanmoins, un certain nombre de limites ont été identifiées.

3.2 Points faibles

— **la sécurité** : il n'y a pas eu d'étude de la sécurité de l'application. Seule la validation des champs des différents formulaires a été prise en compte. Il faudrait prévoir un encodage des mots de passe en base de données mais aussi l'utilisation de sessions...etc.

— le manque de **tests unitaires** : notre application gagnerait à utiliser des outils de tests unitaires côté serveur.

3.3 Perspectives d'amélioration

La liste ci-dessous présente une liste non-exhaustive des extensions que nous pouvons envisager pour notre application :

- **amélioration de la sécurité** :mettre en place des dispositif de cryptage des données envoyées au serveur et vice versa.
- **mise en place de tests unitaires côté serveur** : Définir une couverture de tests appropriés pour toutes les classe java et pour les requêtes appelées sur la BDD.
- **amélioration de l'UX** :La mise en page sur smartphone et tablette est à revoir avec le système de grid de Bootstrap.
- **amélioration des propositions** : les algorithmes de calcul des propositions pourraient prendre en considération des éléments que l'on utilise pas pour l'instant. Par exemple, la liste des musées favoris, les horaires d'ouverture etc. De plus, pour la version mobile de l'application, la localisation de l'utilisateur par GPS pourrait affecter les propositions et amènerait à une expérience utilisateur améliorée.
- **analyser les commentaires** introduits par les utilisateurs à travers un traitement automatique du langage afin d'améliorer les propositions faites par affluence.
- faire appel à l'**API de la RATP** pour fournir les itinéraires lors de la visualisation des musées à proximité.
- **une application évolutive** : proposer l'utilisation de plusieurs langues pour les visiteurs étrangers ou ajouter plusieurs base de données afin de récupérer les données qui concernent la France entière ou même plusieurs grandes villes dans le monde.

Conclusion

Ce projet nous a permis de mettre en pratique les connaissances acquises dans le cadre de l'UE Développement d'Applications Réticulaires. Nous avons ainsi effectué nos premiers pas dans le développement web en Java EE.

Nous avons pu d'une part, développer nos capacités de réflexion, de créativité et de création d'idées, et d'autre part, acquérir un sens de l'organisation et du travail en équipe. Par ailleurs, ce projet était le moyen pour nous de découvrir et d'expérimenter les bonnes pratiques du web actuel.

Nous considérons que les objectifs initiaux ont été atteints (obtenir un site avec des fonctionnalités bien implémentées). Toutefois, des pistes d'améliorations restent envisageables.