



Master 2 STL 2017-2018

Rapport du projet de DAR
PariSup'

Belynda Hamaz

Marie Laporte

Christopher Dionisio

Projet encadré par Romain Demangeon

TABLES DES MATIÈRES

| | |
|---|-----------|
| Introduction | 3 |
| Manuel d'utilisation | 4 |
| Aspect général | 4 |
| Création d'un compte | 5 |
| Connexion | 6 |
| Consulter / Modifier son compte | 6 |
| Rechercher une école ou une université | 7 |
| Consulter une école particulière | 8 |
| Géolocaliser une école | 8 |
| Consulter / Donner un avis sur une école | 9 |
| Ajouter un établissement en favori | 10 |
| Architecture et technologies utilisées | 10 |
| Point de vue global | 10 |
| Serveur | 12 |
| Servlets | 12 |
| JSP | 12 |
| Cookies | 13 |
| Persistance des données | 13 |
| Hibernate | 13 |
| Base de données | 14 |
| Les APIs | 14 |
| Client | 15 |
| Modèle MVC | 16 |
| Déploiement | 17 |
| Méthodes de développement | 18 |
| Résultats et perspectives d'évolution | 19 |
| Points forts et points faibles de l'application | 19 |
| Extensions possibles et améliorations | 19 |
| Conclusion | 21 |

1. Introduction

Dans le cadre de l'UE DAR (Développement d'Applications Réticulaires), nous avons réalisé une application web basée sur un réseau social.

Pour choisir le cadre de l'application, nous nous sommes posés plusieurs questions. Nous voulions à la fois une application utile, qui puisse idéalement aider bon nombre de personnes. Nous nous sommes donc basés sur nos expériences personnelles et il en est ressorti le cadre scolaire, en particulier l'orientation dans les études supérieures. En effet, combien de jeunes chaque année ne savent pas vers quelles études se diriger, que ce soit après le baccalauréat ou en étant déjà dans le supérieur ?

Nous avons donc eu l'idée de développer ce réseau social des études supérieures en Île-de-France, qui a pour but de rassembler des étudiants de tout horizon.

Deux types d'utilisateurs existent : tout d'abord, il y a celui qui souhaite se renseigner sur une université, sur une filière en particulier, et il y a celui qui donne des renseignements sur son université, sa filière.

L'objectif est l'entraide entre jeunes étudiants, qu'ils soient français ou étrangers, ces derniers étant encore plus à même d'utiliser ce genre d'applications que ce soit pour rencontrer du monde ou se renseigner sur leur future filière. Les étudiants s'organisent en réseaux, ceux-ci étant attachés à un établissement/une filière.

Dans ce rapport, nous allons présenter cette application, sa conception et son implantation. Nous décrirons aussi les résultats obtenus et les potentielles évolutions possibles pour PariSup'.

2. Manuel d'utilisation

Nous allons commencer par présenter l'ensemble des fonctionnalités que propose l'application. Chacune des fonctionnalités sera suivie de cas d'utilisation spécifiques à chaque besoin que les utilisateurs pourraient avoir.

2.1. Aspect général

Avant de commencer à énumérer l'ensemble des fonctionnalités, il est important de parler de l'aspect général de l'interface utilisateur.



Page d'accueil de Parisup'

L'ensemble des pages de l'application inclut au moins :

- Une **barre de navigation** permettant de naviguer très rapidement entre différentes pages disponibles sur le site.
- Un **bas de page** indiquant aux utilisateurs comment contacter l'équipe de développement et donnant un accès aux réseaux sociaux liés à Parisup' (facebook / twitter).

La page d'accueil quant à elle, donne un aperçu du contenu du site aux utilisateurs.

2.2. Création d'un compte

La première chose que la plupart des utilisateurs devront faire, c'est se créer un compte afin de s'identifier. Cela leur permettra notamment d'accéder à plusieurs fonctionnalités où l'authentification est nécessaire (ex: laisser un avis sur une école). Pour créer un compte, on va devoir cliquer sur le bouton "Inscription" situé sur la barre de navigation. Il faudra ensuite remplir le formulaire suivant :

| | |
|---|---|
| Prénom: | <input type="text" value="Jean"/> |
| Nom: | <input type="text" value="Rose"/> |
| Email: | <input type="text" value="j.rose@gmail.com"/> |
| Confirmation de l'email: | <input type="text" value="j.rose@gmail.com"/> |
| Mot de passe: | <input type="password" value="****"/> |
| Confirmation du mot de passe: | <input type="password" value="****"/> |
| Adresse:* | <input type="text" value="2 rue des dames"/> |
| Ville:* | <input type="text" value="Paris"/> |
| Anciennes et / ou récente(s) école(s) fréquentée(s):* | <div><input type="text" value="Université Paris Diderot - UFI"/> <input type="button" value="Ajouter"/></div> |
| Liste complète : | |
| <div><input type="text" value="Université Paris Diderot - UFR Lettres, arts et cinéma"/> <input type="button" value="Ajouter"/></div> | |

Formulaire d'inscription

Le formulaire demande un prénom, un nom, un email, et un mot de passe. On peut aussi indiquer de manière optionnelle une adresse, une ville, ainsi que la liste des universités ou écoles fréquentées récemment et / ou anciennes.

Cas d'utilisation :

Jean veut donner son avis sur une école qu'il a beaucoup apprécié, il a donc besoin d'avoir un compte. Il va alors remplir le formulaire et également donner ses coordonnées en précisant qu'il était dans cette école auparavant. Il appuie alors sur le bouton "Inscription" pour confirmer les données qu'il a rentré, mais la page indique que le mot de passe entré n'est pas le même que celui entré en confirmation de mot de passe. Il corrige alors son erreur puis ré-appuie sur le bouton "Inscription". Il est maintenant inscrit sur le site.

2.3. Connexion

Une fois que l'utilisateur a créé un compte, il pourra se connecter et se déconnecter à tout moment. Pour ce faire, il pourra tout simplement cliquer sur le menu déroulant "Connexion" disponible dans la barre de navigation (uniquement si non connecté) :

Formulaire de connexion une fois le menu Connexion déroulé

Une fois les données rentrées, on peut appuyer sur le bouton "Connexion" et l'authentification se fait si le mail et le mot de passe sont valides,, sinon le formulaire indique une erreur.

Cas d'utilisation :

Jean souhaite se connecter sur son compte. Il appuie alors sur le bouton "Connexion" puis remplit le formulaire en précisant son email et son mot de passe. Il appuie sur le bouton "Connexion" pour tenter de s'authentifier, mais le formulaire lui renvoie une erreur lui disant que son email ou son mot de passe est invalide. Il se rend compte qu'il s'est trompé de mot de passe, le rectifie, puis ré-appuie sur le bouton "Connexion" et s'authentifie correctement. Il est ensuite redirigé vers la page d'accueil.

2.4. Consulter / Modifier son compte

Quand un utilisateur est connecté, il est possible d'avoir de nouveau accès au même formulaire que celui de l'inscription afin de modifier ses informations personnelles. Pour cela, il suffit de cliquer dans la barre de navigation sur le menu déroulant affichant le nom de l'utilisateur, puis de cliquer sur le bouton "Mon compte". Le formulaire sera alors rempli automatiquement des informations personnelles actuelles du compte, que l'utilisateur pourra modifier à sa guise.

Cas d'utilisation :

Jean souhaite modifier son mot de passe. Il va alors cliquer sur le menu déroulant de la barre de navigation “Jean R.” puis cliquer sur le bouton “Mon compte”. Il voit un formulaire avec l’ensemble des informations sur son compte, puis modifie la ligne contenant son mot de passe, puis valide le changement en appuyant sur le bouton “Modifier”. Le site affiche un message lui disant que les données ont été modifiées avec succès.

2.5. Rechercher une école ou une université

Il est possible de lister l’ensemble des écoles ou universités disponibles en fonction de certains critères. Pour cela, on clique sur le bouton “Rechercher une école ou une université” et on obtiendra la liste suivante :

The screenshot shows a search interface with the following elements:

- Trier par :** A dropdown menu with "Alphabétique" selected.
- Search bar:** A text input field with "Search..." and a magnifying glass icon.
- Type d'établissement :** Four buttons: "Ecoles" (green), "Instituts" (red), "UFR" (green), and "Autres" (red).
- Results:** Two entries are displayed, each with a star rating (1 star) and a progress bar.
 - INSTITUT D'ÉTUDES JUDICIAIRES**
Paris
Unité de formation et de recherche
Université Panthéon-Sorbonne
 - INSTITUT SUPÉRIEUR DE FORMATION DE L'ENSEIGNEMENT CATHOLIQUE**
Paris
Autre établissement du supérieur
Institut supérieur de formation de

Liste de plusieurs universités ou écoles

Il est possible de trier dans l’ordre alphabétique et il est également possible de filtrer la liste en indiquant le type d’établissement recherché (écoles, instituts, ufr, ou autres), mais aussi de rechercher par mot clé.

Cas d’utilisation :

Jean recherche une excellente école d’art. Il va alors accéder à la liste des universités disponibles en appuyant sur le bouton “Rechercher une école ou une université” dans la barre de navigation. Il choisit de filtrer la liste en fonction des notes, et en filtrant uniquement les écoles. Il trouve “Créapôle ESDI” en haut de la liste.

2.6. Consulter une école particulière

Après avoir effectué une recherche, on peut consulter une école afin d'obtenir plus d'informations.

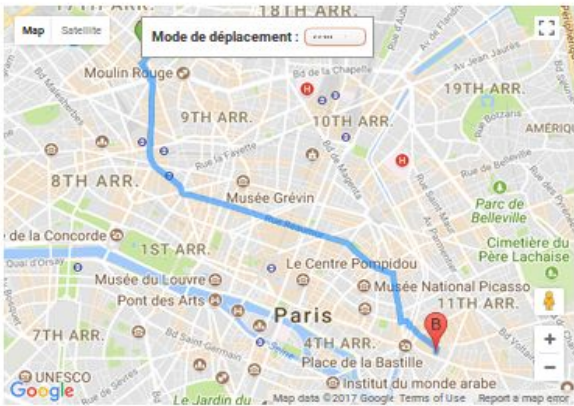
ACADÉMIE GRANDES TERRES

Informations sur Académie Grandes Terres

☆

Nom : Académie Grandes Terres
Adresse : 5 rue de Charonne
Département : 75 - Paris
Latitude : 48.852404
Longitude : 2.37417
Type d'établissement : Ecole d'art
Académie : Paris
Code UAI : 0754588E
Sigle :
Site internet : <http://www.onisep.fr/http/redirection/etablissement/identifiant/8056>

DE 1 Rue des Dames, 75018 Paris, France A 10 Rue de Charonne, 75012 Paris, France :
Trajet : 6.1 km (DRIVING)
Temps : 28 mins



AVIS

Laisser votre avis

Détails à propos de l'école d'art Créapôle ESDI

On a alors accès à plusieurs données dont l'adresse, le département, la position, le type d'établissement, le site internet de l'Onisep... C'est également ici qu'on peut connaître l'itinéraire et donner / consulter des avis.

Cas d'utilisation :

Jean a besoin de connaître l'adresse de de cette école. Il clique sur l'école dans la liste et arrive sur sa page. Il peut alors voir sur la partie gauche l'adresse de l'école en question.

2.7. Géolocaliser une école

La partie droite de la description d'une école permet de géolocaliser un établissement. l'API google maps va déterminer un itinéraire allant de l'adresse de l'utilisateur vers l'établissement sélectionné. Si l'utilisateur n'avait entré aucune adresse, ou que cette dernière n'existe pas, ou bien

que tout simplement l'utilisateur n'est pas connecté, alors l'utilisateur sera géolocalisé automatiquement.

Cas d'utilisation :

Jean souhaite connaître le temps de trajet entre son école et sa maison à vélo. Pour cela, il sélectionne le mode de déplacement "à vélo" dans le gadget Google Maps intégré et obtient un résultat.

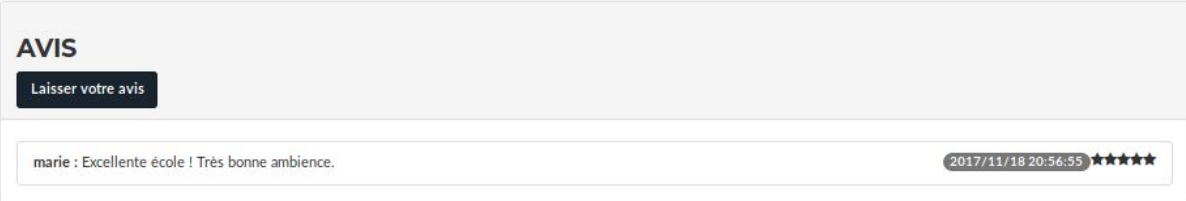
2.8. Consulter / Donner un avis sur une école

Si un utilisateur souhaite noter un établissement, il peut y laisser un avis afin d'orienter les possibles futurs étudiants. Pour cela, il suffit d'appuyer sur le bouton "Laisser votre avis" et de compléter le formulaire suivant :

A screenshot of a web form titled "LAISSER VOTRE AVIS" with a close button (X) in the top right corner. The form contains a "Note :" section with five yellow stars, a "Commentaire :" section with a text input field containing the text "Excellente école ! Très bonne ambiance.", and two buttons at the bottom right: "Fermer" and "Valider".

Fenêtre permettant de laisser un avis sur un établissement

Une fois validé, il sera ajouté à la liste des avis liés à cet établissement de cette manière :

A screenshot of a web interface showing a list of reviews. At the top, there is a header "AVIS" and a button "Laisser votre avis". Below this, a review is displayed: "marie : Excellente école ! Très bonne ambiance." followed by a date and time stamp "2017/11/18 20:56:55" and five stars.

L'ensemble des avis donnés par des utilisateurs sur un établissement

Un utilisateur doit forcément être connecté pour pouvoir donner un avis.

Cas d'utilisation :

Jean souhaite donner son avis sur l'école qu'il vient de terminer. Il consulte cette école, puis clique sur le bouton "Laisser votre avis", donne 4 étoiles sur 5 à cette école avec le commentaire suivant "Bien malgré quelques soucis d'emploi du temps.". Après avoir validé, l'avis est affiché dans la liste des avis et sera visible par n'importe quel autre utilisateur, connecté ou non.

2.9. Ajouter un établissement en favori

Il est possible, lorsqu'un utilisateur se trouve sur la page d'un établissement particulier, de mettre cet établissement en favori. Pour cela, il suffit de cliquer sur l'étoile à côté du nom de l'établissement en question. L'utilisateur pourra alors y accéder via la page de son compte personnel, et en cliquant sur l'école en question. Un utilisateur pourra également enlever un établissement de ses favoris en re-cliquant sur l'étoile.

***Note :** l'ajout et la suppression de favoris est également possible directement sur la page de modification de compte.*

Cas d'utilisation :

Jean est intéressé par une école et souhaite pouvoir re-consulter sa page plus tard. Il clique alors sur l'étoile, puis ferme le navigateur. Le lendemain, il se connecte, va sur son compte, puis clique sur l'école qu'il avait mise en favori.

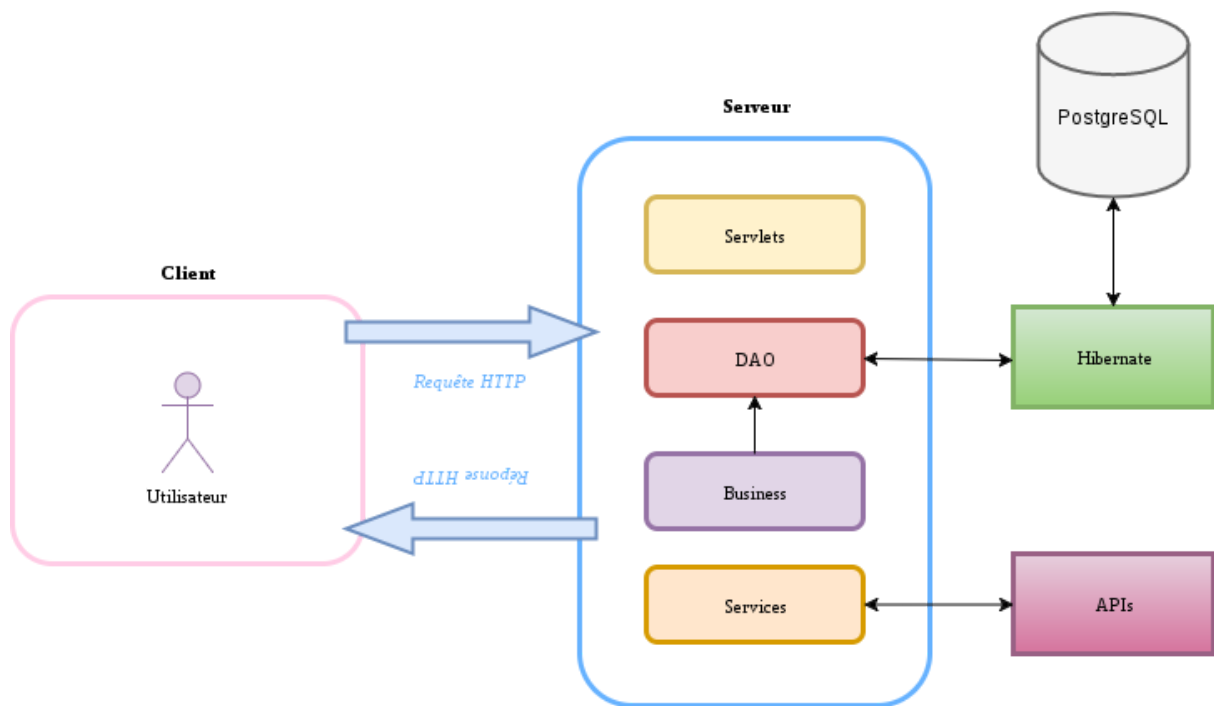
3. Architecture et technologies utilisées

3.1. Point de vue global

L'application *PariSup* est basée sur une architecture trois tiers, elle est donc composée d'une partie serveur, persistance des données et une partie client. Cette architecture logique est basée sur l'environnement client-serveur. Pour faire le lien entre la couche métier et la couche persistance des données, nous utilisons le **pattern DAO**, ceci pour plusieurs raisons :

- Centraliser les mécanismes de mapping entre la base de données et les objets métiers Java
- Prévenir un éventuel changement de base de données

Concernant ce dernier point, nous avons fait intervenir le pattern **Factory** et **Abstract Factory**. Ces deux patterns nous permettent à la fois de centraliser l'instanciation de nos *DAO* dans une seule classe et de faciliter grandement la cohabitation de plusieurs systèmes de stockage sur une même application et ainsi rendre notre application plus souple.



Architecture de *PariSup*

Les objets *DAO* communiquent avec le système de stockage et nous retournent l'objet métier associé en objet Java. Ces objets sont ensuite envoyés au client au format *JSON*, ce dernier pouvant effectuer des requêtes *http* au serveur. Voici un aperçu des classes Java utilisées pour implémenter le pattern DAO :

```

▼ src/main/java
  ▼ com.upmc.parisup.api.schools
    ▶ SchoolAPIConstants.java
    ▶ SchoolAPIHelper.java
    ▶ SchoolAPIRequestHelper.java
    ▶ SchoolAPIService.java
  ▼ com.upmc.parisup.business
    ▶ Rating.java
    ▶ School.java
    ▶ SelectedSchool.java
    ▶ User.java
  ▼ com.upmc.parisup.DAO
    ▶ AbstractDAOFactory.java
    ▶ DAO.java
    ▶ Factory.java
    ▶ MyPostgreSQLDAOFactory.java
    ▶ RatingDAO.java
    ▶ SchoolDAO.java
    ▶ SelectedSchoolDAO.java
    ▶ UserDAO.java
  ▼ com.upmc.parisup.DAO.DAOImpl
    ▶ MyPostgreSQLDAOImpl.java
    ▶ RatingDAOImpl.java
    ▶ SchoolDAOImpl.java
    ▶ SelectedSchoolDAOImpl.java
    ▶ UserDAOImpl.java
  
```

Implémentation du design pattern DAO

3.2. Serveur

La partie serveur est écrite en Java et va communiquer avec le client et la base de données.

3.2.1. Servlets

Nous utilisons des **servlets** pour la réception des requêtes http et l'envoi de réponses au client. Ces différents servlets sont exécutés par un conteneur de servlets, ici **Apache Tomcat**. Notamment, l'implémentation de servlet se fait par dérivation de la classe Java **HttpServlet**, et par la réécriture des méthodes `doGet` et / ou `doPost`. Ces deux méthodes récupèrent une **requête** envoyée par le client, puis lui renvoient une **réponse**.

```
1 package com.upmc.parisup.servlets;
2
3 import java.io.IOException;
4
5
6
7
8
9
10 public class Homepage extends HttpServlet {
11     private static final long serialVersionUID = -4751096228274971485L;
12
13     @Override
14     protected void doGet(HttpServletRequest request, HttpServletResponse response)
15         throws ServletException, IOException {
16         request.getRequestDispatcher("WEB-INF/index.jsp").forward(request, response);
17     }
18 }
```

Exemple de servlet utilisé pour la page d'accueil

3.2.2. JSP

Le serveur utilise également la technologie **JSP** pour générer des pages HTMLs de manière dynamique. Nous avons choisi de l'utiliser puisque très pratique à combiner avec les réponses des servlets. Pour ce faire, on a également utilisé **JSTL** (Java server page Standard Tag Library) qui permet d'ajouter une bibliothèque de balises qui seront traduites en balises HTML. Par exemple, on pourra rajouter à la réponse du servlet un attribut "schools" récupéré via le DAO et contenant l'ensemble des établissements dans la base de données :

```
List<School> schools = ((SchoolDAOImpl) AbstractDAOFactory
    .getFactory(Factory.MYSQL_DAO_FACTORY).getSchoolDAO()).getAll();
request.setAttribute("schools", schools);
```

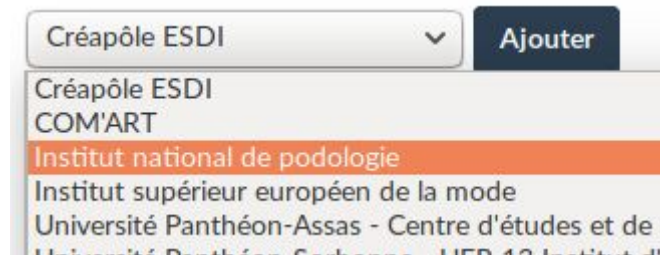
Récupération de la liste des établissements et envoi au fichier JSP

Puis l'utiliser à l'aide des balises JSP afin de générer une liste déroulante contenant l'ensemble de ces établissements :

```
<select class="fixed-size" id="allSchoolsList" name="school">
  <c:forEach items="${schools}" var="school">
    <option value="${school.id}"><c:out value="${school}" /></option>
  </c:forEach>
</select>
```

Le code JSP permettant de lister l'ensemble des établissements

Ce qui donnera le résultat suivant :



La liste déroulante visible par le client

3.2.3. Cookies

Côté serveur, il est possible de manipuler les **cookies**. Dans notre cas, les cookies ont été utilisés uniquement dans le but d'authentifier un utilisateur. On utilise alors un cookie ayant pour clé "email". Un servlet peut par exemple avoir besoin de l'objet métier User correspondant, pour cela, on utilisera la méthode Java suivante recherchant le cookie "email", puis retournant le User associé au mail via une requête à la base de donnée :

```
public static User getCurrentUser(HttpServletRequest request) {
    Cookie[] cookies = request.getCookies();

    if (cookies != null) {
        for (Cookie cookie : cookies) {
            if (cookie.getName().equals("email")) {
                String email = cookie.getValue();
                return ((UserDAOImpl) AbstractDAOFactory
                    .getFactory(Factory.MYSQL_DAO_FACTORY).getUserDAO())
                    .getByAttribute("email", email);
            }
        }
    }

    return null;
}
```

La fonction récupérant l'utilisateur courant

3.3. Persistance des données

3.3.1. Hibernate

En ce qui concerne la persistance des données, nous avons opté pour le choix d'utiliser un *ORM*. Cette technique crée un lien entre nos objets métier Java et notre base de données SQL (chaque table correspondant à un objet Java). Cela permet entre autres de nous affranchir de beaucoup de code SQL et d'interagir très facilement avec nos objets persistants. Nous avons opté pour le framework Hibernate qui est un *ORM* très utilisé dans la communauté Java. Son installation a nécessité un moment, cependant le temps gagné ensuite rend l'opération rentable.

Pour utiliser Hibernate, il faut tout d'abord mapper nos objets Java avec les tables dans la base de données à l'aide d'un fichier nommé `<nom_de_la_classe>.hbm.xml` pour chaque objet. Ces fichiers vont permettre à Hibernate de savoir comment mapper nos classes Java en tables dans la base de données. Enfin, un fichier `hibernate.cfg.xml` va contenir la configuration de l'accès au système de stockage.

3.3.2. Base de données

Nous avons choisi d'utiliser une base de données *PostgreSQL*. Pourquoi ce choix ? Tout d'abord car PostgreSQL est un *SGBD* relationnel et nos objets étant liés, le choix d'un système de stockage relationnel nous paraissait le plus judicieux. De plus, Hibernate est particulièrement adapté pour les bases de données relationnelles.

Au départ, nous avons fait le choix d'utiliser une base de données *MySQL*, cependant il s'est avéré que Heroku, le service de cloud computing qui héberge notre application utilise nativement PostgreSQL, c'est pourquoi nous avons dévié vers ce choix, afin de nous affranchir de conversions et de l'utilisation de *add-ons* supplémentaires.

3.3.3. Les APIs

Pour remplir notre base de données, nous avons une API externe, **l'API Open Data Île-de-France**, afin de récupérer l'ensemble des établissements scolaires Franciliens. Étant donné que cette API est statique, autrement dit qu'elle n'est pas régulièrement mise à jour, nous avons fait le choix d'intégrer directement dans la base de données toutes les données de l'API lors du déploiement, ainsi l'utilisateur dispose d'une grande variété de choix d'établissements scolaires.

L'API Google Maps quant à elle, est utilisée directement depuis le navigateur à l'aide de JavaScript. Nous faisons appel à cette API lorsqu'un utilisateur se trouve sur la page d'un établissement. En utilisant les coordonnées GPS de l'établissement ainsi que l'adresse de l'utilisateur, nous plaçons des marqueurs afin qu'il puisse juger de la distance entre son domicile et l'établissement. L'utilisateur a la possibilité de changer le moyen de déplacement.

3.4. Client

Côté client, nous avons essentiellement utilisé du **HTML**, **Bootstrap**, **JavaScript**, et **jQuery**. Les balises HTML sont combinées avec les balises de la technologie JSP, mais elles restent très présentes. Du code CSS est également présent afin de gérer l'ensemble des styles des différents éléments du site. Le JavaScript est très fréquemment utilisé avec JQuery puisque cette dernière librairie va nous permettre de faire des requêtes au serveur.

Typiquement, une requête au serveur (asynchrone afin de ne pas figer le navigateur) pourra avoir la forme suivante :

```
$('#sign').click(function(e) {
    e.preventDefault();

    $.ajax({
        type : 'POST',
        url : 'signup',
        data : getDatas()
    }).done(function(data) {
        if (data.success) {
            var user = JSON.parse(data.user);
            window.location.assign("/");
            login(user);
        } else {
            $('#div #error').html(data.message);
            $('#div #error').show();
        }
    }).fail(function() {
        alert("Server not responding.");
    });
});
```

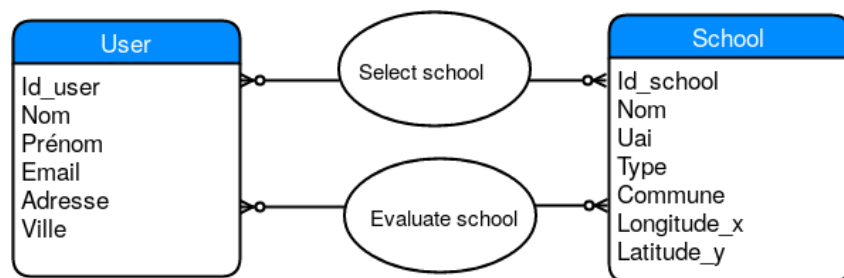
La requête AJAX permettant de réagir au bouton d'inscription

Dans cet exemple, on va réagir lors de l'appuie sur le bouton "Inscription". On va ici demander au serveur de répondre à une requête HTTP POST à l'URL "signup" en fournissant en paramètre l'ensemble des données contenues dans le formulaire d'inscription. Le servlet associé à cette URL va alors exécuter la fonction `doPost()` implémentée. Dans cette méthode, il va récupérer les données envoyées par le client et faire des appels à la base de données afin de les enregistrer. Il va aussi vérifier si les données sont correctement entrées. Le servlet va alors renvoyer une réponse sous le format **JSON** et réagir en fonction. Dans notre cas, s'il y a eu des erreurs, alors elles seront affichées dans un div spécifique. sinon alors on redirige sur la page *home* et on connecte l'utilisateur.

3.5. Modèle MVC

Le pattern DAO est en relation avec le pattern Model-View-Controller, où les modèles sont les objets métiers, les contrôleurs sont les servlets, et les vues sont les pages HTML générés par le serveur via JSP. Une fois implémenté, on a le résultat qui suit.

Modèles



Les objets métiers utilisés

Vues

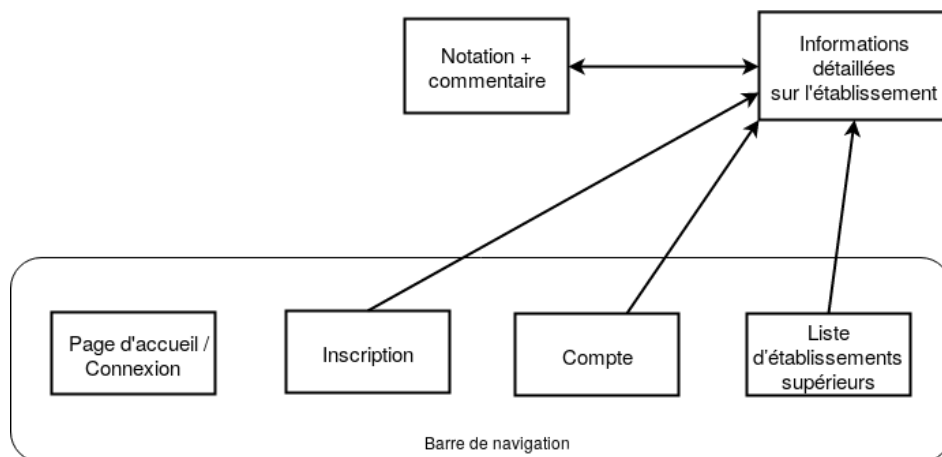


Schéma de navigation

Controleurs



Les servlets utilisés

3.6. Déploiement

Le déploiement de l'application s'est fait au moins un mois avant le rendu final de l'application, nous voulions être certains que l'application soit déployée en temps et en heure. Au départ, nous souhaitions utiliser un serveur et nom de domaine personnalisé à l'aide d'un compte personnel d'un contact. Cependant, le manque d'accès direct aux options du compte nous a rapidement fait changer d'idée. Nous avons alors opté pour une solution gratuite et simple : **Heroku**.



Pour cela, l'ensemble de l'équipe a installé Heroku en ligne de commande, et s'en est ensuite servi également pour tester le site localement. Nous avons installé l'add-on **Postgres** afin d'avoir une base de donnée à notre disposition en ligne. Une fois que tout le processus de déploiement a correctement fonctionné, nous sommes tous repartis sur une base de donnée locale afin d'y effectuer des tests unitaires.

3.7. Méthodes de développement



Afin de travailler efficacement sur le projet, notre équipe a principalement utilisé les 4 outils suivants :

- GitHub : Pour gérer le code source de l'application web.
- Trello : Afin de lister l'ensemble des tâches à accomplir.
- Slack : Pour toujours pouvoir communiquer instantanément au sein de l'équipe, poser des questions, demander un avis, etc.
- Google Drive : Pour partager l'ensemble des fichiers nécessaires à la rédaction du rapport et à la présentation du projet.

4. Résultats et perspectives d'évolution

4.1. Points forts et points faibles de l'application

Notre application a pour avantage :

- D'avoir une interface facile à utiliser, permettant aux utilisateurs de découvrir facilement son contenu.
- D'utiliser le modèle MVC qui permet d'avoir une architecture claire, et ainsi faciliter la maintenance ou les améliorations qui peuvent être apportées par d'autres développeurs.
- De pouvoir passer de n'importe quel type de base de données à une autre, en changeant uniquement les traitements d'interaction avec la base.
- D'utiliser des APIs externes qui enrichissent son contenu, soit en les interrogeant dynamiquement, soit pour remplir la base de données.
- De proposer un service original.
- De garantir une certaine sécurité des données de l'utilisateur par un cryptage du mot de passe donné.

Notre application a également les points faibles suivants :

- Sa dépendance à l'hébergeur Heroku.
- Une sécurité insuffisante.
- Pas assez de tests unitaires effectués.
- Le code JavaScript n'est pas minimisé/obfusqué.

4.2. Extensions possibles et améliorations

Avec plus de temps, diverses extensions ou améliorations auraient pu être implémentées afin d'améliorer l'expérience utilisateur :

- Une première extension aurait pu être **d'internationaliser l'application**. En effet, les étudiants non francophones qui voudraient venir étudier en France apprécieraient sûrement un site disponible au moins en anglais. La librairie **JSTL** permet justement de faire cela, à l'aide de fichiers *properties*, chacun contenant tous les textes de l'application traduits dans la langue associée au fichier.
- Nous aurions pu améliorer **l'affichage au niveau des appareils mobiles** car malgré l'utilisation d'un template Bootstrap, l'affichage n'est pas parfait sur ces terminaux.
- **La sécurité** aurait elle aussi mérité d'être améliorée. Outre le fait que les mots de passe soient sécurisés de manière efficace, nous ne vérifions pas assez les input de l'utilisateur.

- En ce qui concerne l'application elle-même, il aurait été agréable pour l'utilisateur d'avoir accès à une sorte de **messagerie instantanée** afin de pouvoir communiquer directement avec les autres membres, qui seraient organisés sous forme de réseaux par établissement, et ainsi d'avoir les réponses à ses questions plus rapidement quant à l'établissement.
- Le service **d'envoi d'emails** manque à l'appel. En effet, lorsqu'un utilisateur oublie son mot de passe, il a la possibilité de le récupérer via un lien qui lui aura été envoyé par mail. Cependant, nous nous sommes rendus compte trop tard que Heroku ne prenait pas en charge ce service gratuitement (cela marchait parfaitement localement). Nous avons donc été contraint d'annuler cette fonctionnalité au dernier moment.
- Enfin, l'utilisation de l'**API RATP** aurait été un plus et aurait permis à l'utilisateur de savoir quelles stations de métro sont à proximité de l'établissement/de son domicile.

5. Conclusion

Le projet de Développement d'Applications Réticulaires nous a permis de mettre en pratique les connaissances acquises durant l'UE mais aussi de voir une manière un peu différente d'envisager une application web. En effet, nous avons déjà eu l'occasion dans le passé de créer une application web, cependant du côté serveur, nous avons utilisé le framework open-source *Jersey* afin de développer des services web selon l'architecture REST, entre autres.

Nous avons pu lors de ce projet mettre à profit et renforcer notre créativité ainsi que compétences techniques. Ce projet nous a donc globalement permis d'approfondir nos connaissances dans le développement avec J2E et de découvrir de nouvelles technologies tels que les ORMs ainsi que le déploiement d'une application, cet aspect de la programmation web qui était encore nouveau pour nous jusqu'à aujourd'hui.

Bien qu'il y ait des améliorations à effectuer, l'application fonctionne bien et répond au sujet ; nous considérons donc l'objectif comme accompli.