LO54 – Projet : Plateforme de cours

Table des matières

[Introduction 3](#_Toc532909717)

[La base de données 4](#_Toc532909718)

[L’application 5](#_Toc532909719)

[L’API REST J2E 5](#_Toc532909720)

[L’application web 6](#_Toc532909721)

[Tutoriel : Installation de l’environnement 7](#_Toc532909722)

[Installer la base de données 7](#_Toc532909723)

[Installer l’API 7](#_Toc532909724)

[Installer l’interface web 7](#_Toc532909725)

[Tutoriel : Utilisation de l’Application 8](#_Toc532909726)

[Authentification de l’utilisateur 8](#_Toc532909727)

[Création de cours 8](#_Toc532909728)

[Filtrage des cours 8](#_Toc532909729)

[Inscription 8](#_Toc532909730)

[Visionnage du taux de remplissage du cours 8](#_Toc532909731)

[Conclusion 9](#_Toc532909732)

[Retour d’expérience 9](#_Toc532909733)

[Utilisation de la technologie : Spring REST 9](#_Toc532909734)

# Introduction

Durant ce semestre, il nous a été confié la tâche de développer une application J2E se basant sur les techniques et technologies étudiées lors de l’UV LO54. Cette plateforme avait pour but de permettre à une école privée fictive de gérer un catalogue de cours auxquels sont rattachés des sessions effectuées à des dates prédéfinies. L’utilisateur devait donc posséder les options suivantes :

* Filtrer les cours par des mots-clés contenus dans le titre de la formation.
* Filtrer la liste des cours selon la disponibilité des sessions.
* Filtrer les formations selon le lieu de la session (à l’aide d’une liste déroulante)
* L’utilisateur doit pouvoir s’inscrire à une session.
* L’utilisateur doit pouvoir visionner le taux de remplissage d’un cours.

Les options suivantes, bien que facultatives, ont été également demandées :

* L’utilisateur doit pouvoir s’identifier.
* L’utilisateur doit pouvoir créer des cours.

Ce PDF est décomposable en plusieurs parties : Tout d’abord cette introduction rappelant l’origine du projet et les besoins auxquels il doit se conformer. Ensuite viendra la partie technique : un paragraphe sera consacré à la base de données et un autre aux technologies employées avec à chaque fois les justifications de nos choix. Suivront un tutoriel d’utilisation de l’application un guide d’installation.

Naturellement, les liens bibliographiques, le lexique ainsi que quelques liens utiles seront disponibles à la fin du document.

Bonne lecture.

# La base de données

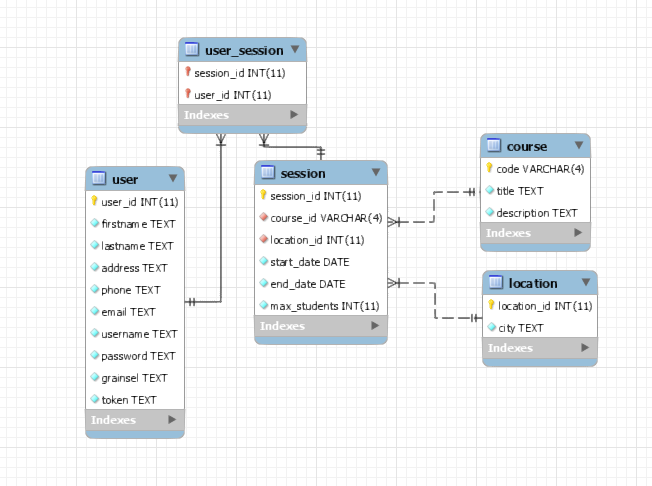


Figure : Base de donnée utilisée

Pour ce projet, nous avons utilisé la base de données **MySQL** décrite comme ci-dessus. Afin de pouvoir faire évoluer notre application et d’inclure les options facultatives décrites par le sujet, nous avons dès le départ créé une table de jointure entre les tables **User** et **Session** et inclus dans la table **User** des champs **username**, **password**, **grainsel** et **token** servant à identifier l’utilisateur et assurer la sécurité de sa connexion.

Nous avons également créé une table dédiée à la gestion des **locations** de manière à pouvoir aisément récupérer la liste des locations existantes et ainsi filtrer aisément les sessions.

Enfin, la table **Course** contient toutes les données génériques que partagent toutes les sessions d’un même cours, à savoir leur titre et leur description. Nous pensons qu’il est utile de justifier l’utilisation d’un champ **VARCHAR(4)** pour l’identifiant de cette table : nous avons pris comme modèle l’identification des cours de l’UTBM qui permet de retrouver un cours à l’aide d’un identifiant court mais explicite.

Partant de ce modèle, nous avons pu commencer la création de l’application.

# L’application

Pour réaliser l’application, nous nous sommes mis dans le contexte d’une application d’entreprise : nous avons imaginé que plusieurs serveurs étaient disponibles et que chacun était délimité par un rôle précis. Aussi nous avons scindé l’application en trois parties absolument dissociées dont chacune devait être théoriquement déployée sur un serveur différent :

1. La base de données, que nous avons vue plus tôt.
2. L’API REST J2E, permettant de requêter la base de données de façon sécurisée.
3. L’application web, offrant à l’utilisateur une interface intuitive lui permettant de manipuler l’API et par extension la base de données.

Nous détaillerons les deux derniers points de cette liste ci-dessous.

## L’API REST J2E

Le rôle de l’API J2E est d’appliquer les règles métier à la base de donnée à l’aide de requêtes http et https et de retourner des réponses sous format JSON. Pour la réaliser, nous nous sommes basés sur les frameworks et bibliothèques **Hibernate** et **Spring** ainsi que leurs dépendances respectives.

Nous avons décomposé l’application en plusieurs packages :

* **Entities** : Contenant les classes structurant les données insérées et récupérées de la base de données.
* **Repositories** : Contenant les classes requêtant directement la base de données.
* **Controllers** : Contenant les classes mappant les URLs à requêter pour obtenir des informations de l’API ainsi que les traitements associés.
* **Services**: Destiné à regrouper les classes détaillant les services. Nous ne nous en sommes finalement pas servis mais avons laissé le package et ses fichiers de façon à pouvoir les utiliser lors d’une mise à jour future.
* **Helpers**: Regroupant les classes possédant des fonctions statiques pouvant être utiles à tout endroit de l’application.
* **Validators**: Contenant une classe possédant quelques fonctions permettant une validation et vérification simplifiée des structures de données.

Grâce à cette structure, nous pouvons requêter l’API et en extraire les résultats, exemple :

https://puu.sh/CjtGv/04f6d46257.png

Figure : Requête REST

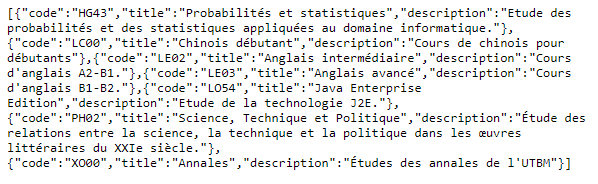


Figure : Résultat de la requête

Il est bon de noter que nous avons préféré l’usage des annotations Hibernate aux fichiers de configuration XML, ces derniers étant peu lisibles et compréhensibles.

## L’application web

Partant du principe que la partie « front-end » serait hébergée sur un serveur différent de la partie API, l’utilisation d’une autre technologie axée essentiellement sur le design et l’expérience utilisateur nous a paru particulièrement pertinente.

Nous avons donc décidé d’utiliser React.js, un Framework Node.js, dont le principal avantage est de permettre la réutilisabilité d’éléments d’interface.

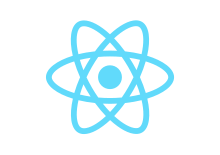


Figure : Logo React.js

Le sujet de l’UV liée à ce projet n’étant pas la conception d’interfaces graphique, nous avons décidé de passer sous silence les détails de la réalisation de l’application web et de passer au tutoriel d’installation.

# Tutoriel : Installation de l’environnement

Etant donné notre choix de développer cette application selon un modèle d’entreprise, nous devons avoir recours à plusieurs serveurs web tournant sur une même machine pour essayer l’application en local.

## Installer la base de données

Pour installer la base de données MySQL, nous avons utilisé le logiciel **XAMPP** et simplement chargé le fichier .sql fourni dans le dossier « database » des sources.

## Installer l’API

Tout comme la base de données, l’installation de l’API est courte : il suffit de compiler le code du dossier API présent dans le .zip contenant les sources de l’application.

**Attention** : le fichier de configuration (hibernate.cfg.xml) possède comme identifiants de connexion à la base de données « **lo54** » et « **lo54** » et la base de données utilisée est nommée « **borothfriplo54** ».

Vous pouvez accéder à l’API par l’URL <http://localhost:8080>

## Installer l’interface web

Pour installer l’interface web, vous devrez auparavant installer **Node.js**. Une fois cela fait, utilisez l’invite de commande Node.js et exécutez les commandes suivantes dans le dossier « app » des sources :

* npm install : Installera les dépendances dont le projet a besoin pour s’exécuter.
* npm start : Crée le serveur web sur lequel l’interface s’exécutera.

Vous pouvez ensuite accéder à l’interface web par l’URL <http://localhost:3000>

En cas d’erreur lors du npm install, n’hésitez pas à utiliser la commande :

* npm cache clean --force

Une fois les trois serveurs lancés, vous pouvez passer au **Tutoriel d’utilisation**.

# Tutoriel : Utilisation de l’Application

Les paragraphes suivants se centrent autour des services demandés par le cahier des charges.

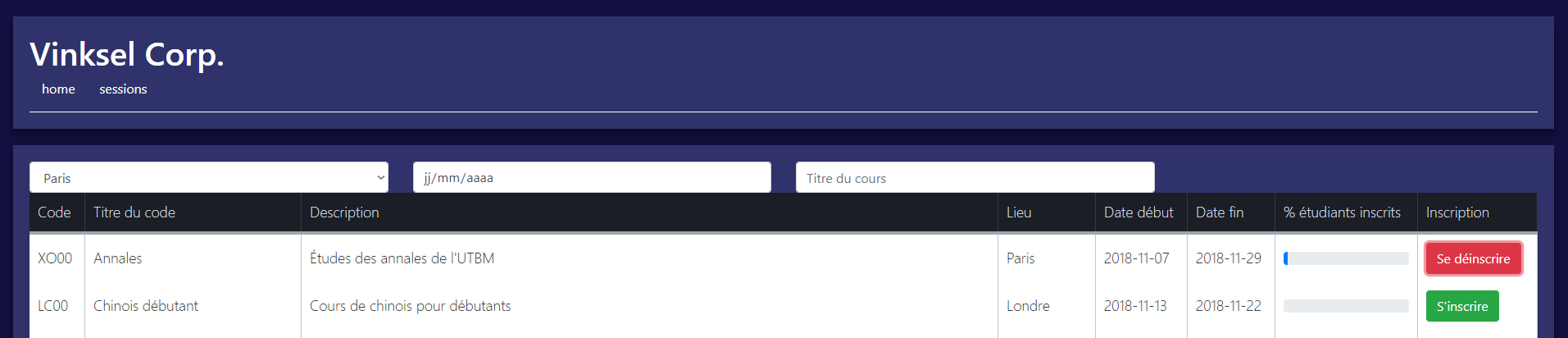
## Authentification de l’utilisateur

Une fois l’application bien installée et démarrée, rendez vous sur la page :

<http://localhost:3000/>

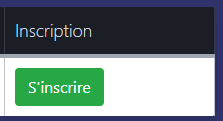
Et connectez-vous avec les identifiants : « admin » et « mdp » (respectivement nom de compte et mot de passe). Vous arriverez ensuite sur la page d’accueil.

## Filtrage des cours



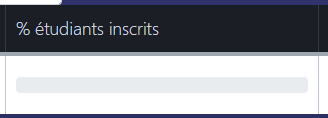
Le tableau des sessions peut être filtré par les villes où elles ont lieu, par les dates de début ainsi que par le titre des cours (Attention : Ne pas confondre Titre et Code du cours !).

## Inscription



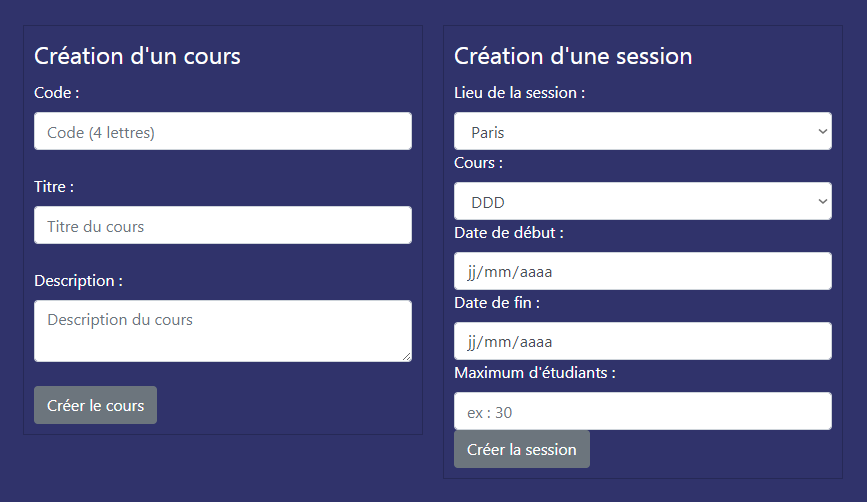
Afin de s’inscrire à un cours, l’utilisateur peut simplement cliquer sur le bouton s’inscrire (ou désinscrire pour se désinscrire) à l’extrémité droite de la ligne du cours désiré.

## Visionnage du taux de remplissage du cours

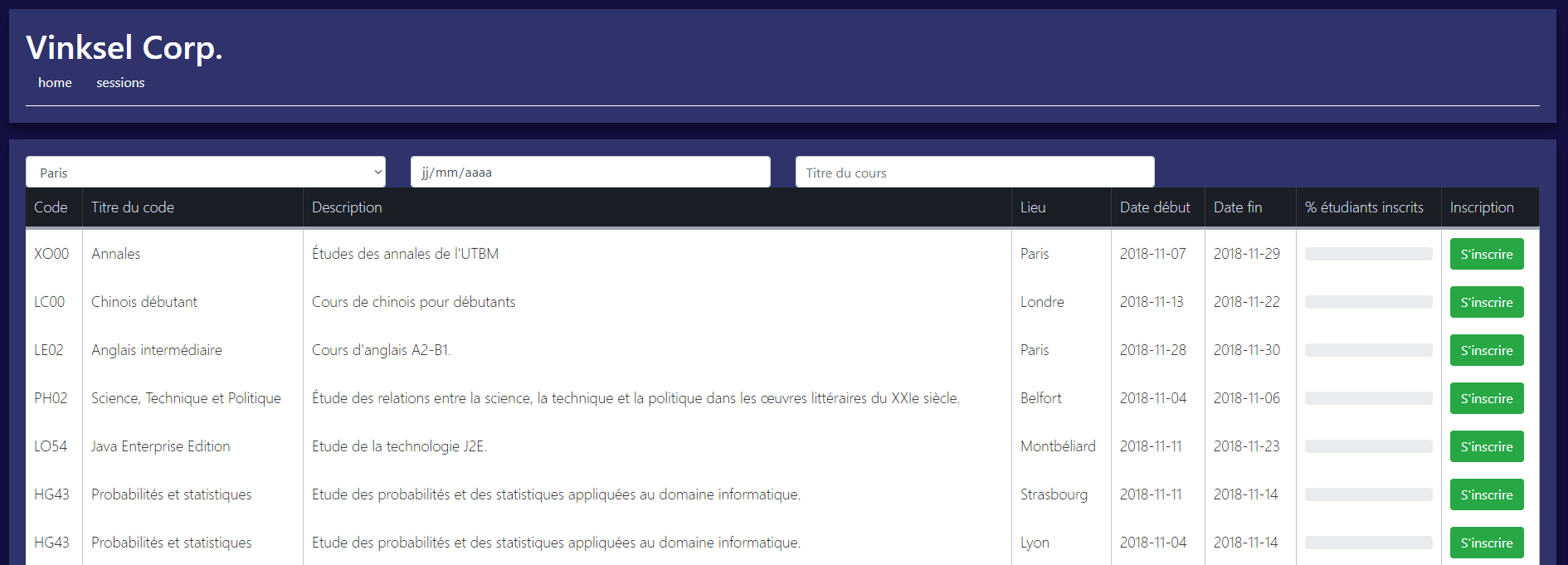


Le taux de remplissage du cours est indiqué par une petite barre sur la ligne du cours.

## Création de cours



Grâce à ce formulaire, l’utilisateur peut créer un cours puis instancier des sessions à partir de ce cours. Le cours créé apparaitra alors immédiatement dans la liste des sessions disponibles.



# Conclusion

## Retour d’expérience

Il nous a été confié le soin d’utiliser la bibliothèque **Spring** pour réaliser notre projet. Nous avons eu d’abord quelques problèmes à identifier quelle partie de Spring devait être étudiée avant de finalement décider de nous concentrer sur l’utilisation de cette technologie dans le but de créer une **API Restful**.

De ce point de vue, la bibliothèque a été très efficace et nous a parmi d’obtenir très rapidement de bons résultats. Bien que nous n’en maîtrisons pas encore tous les aspects, nous avons compris et intégré les principales fonctionnalités de Spring qui étaient nécessaires à notre projet.

## Utilisation de la technologie : Spring REST

Nous avons principalement utilisé Spring web pour définir les controllers à requêter sur notre API, notamment à l’aide de l’annotation « @RequestMapping("url") » et limiter l’accès à l’API avec « @CrossOrigin(origins = "http://localhost:3000") ». De cette façon, seule l’interface web que nous avons conçu a accès à l’API et régler les problèmes de type CORS. Nous avons également employé les annotations telles que « @RequestParam » ou « @PathVariable » pour récupérer les paramètres passés dans les URL ou plus généralement dans les requêtes.

## Améliorations futures

Nous avons prévu un certain nombre d’améliorations futures à apporter à ce projet et avons mis en place la structure pour les accueillir le plus aisément possible au niveau de l’API. Malheureusement le temps nous a manqué pour pouvoir les implanter au niveau de l’interface graphique.

Parmi ces améliorations :

* **La gestion des utilisateurs** : Actuellement seuls sont disponibles les utilisateurs créés par nos soins (deux profils utilisateurs), mais nous aimerions pouvoir permettre à un profil administrateur (détaillé plus loin) de pouvoir ajouter ou valider l’inscription d’utilisateurs.
* **La gestion des droits** : Afin de permettre de restreindre les droits de façon à ce que tout le monde ne puisse pas modifier, supprimer ou créer des cours.
* **La visualisation des cours courants**: Qui permettrait à un utilisateur de facilement trouver les sessions auxquelles il est inscrit.
* **L’instanciation de nouvelles sessions**: Actuellement, par manque de temps, seule la session créée avec la création du cours est disponible. Nous souhaiterions que, par la suite, nous puissions rajouter autant de sessions que nous le voulions.

# Annexes

## Tableau des points d’entrée de l’API

|  |  |  |  |
| --- | --- | --- | --- |
| Point d’entrée | Type de requête | Paramètres | Description |
| */courses* | **GET** |  | Retourne la liste des cours |
| */course/add* | **POST** | String **code** : ID du cours  String **title** : Titre du cours  String **desc** : Description du cours | Crée un nouveau cours |
| */course/edit* | **POST** | String **id** : ID du cours  String **code** : Nouvel code du cours  String **title** : Titre du cours  String **desc** : Description du cours | Edite un cours |
| */course/delete* | **POST** | String **id** : ID du cours | Supprime le cours |
| */course/view* | **POST** | String **id** : ID du cours | Renvois les données d’un cours |
| */sessions* | **GET** |  | Retourne la liste des sessions |
| */session/add* | **POST** | String **courseID** : ID du cours  Integer **locationID** : ID du lieu  String **start\_date** : Date de début de de la session (yyyy-MM-dd)  String **end\_date** : Date de fin de la session (yyyy-MM-dd)  Integer **max\_student** : Maximum d’inscrits au cours. | Crée une nouvelle session |
| */session /edit* | **POST** | Integer **id** : ID de la session  String **courseID** : ID du cours  Integer **locationID** : ID du lieu  String **start\_date** : Date de début de de la session (yyyy-MM-dd)  String **end\_date** : Date de fin de la session (yyyy-MM-dd)  Integer **max\_student** : Maximum d’inscrits au cours. | Edite une session |
| */session/delete* | **POST** | String **id** : ID de la session | Supprime la session |
| */session/view* | **POST** | String **id** : ID de la session | Renvois les données d’une session |
| */session/unregister* | **POST** | Integer idSession : ID de la session  Integer idUser : ID de l’utilisateur | Désinscrit l’utilisateur de la session |
| */session/register* | **POST** | Integer idSession : ID de la session  Integer idUser : ID de l’utilisateur | Inscrit l’utilisateur à la session |
| */locations* | **GET** |  | Retourne la liste des lieux |
| */location/{id]* | **GET** | Integer **id** : ID du lieu | Retourne un lieu précis grâce à son id |
| */location/add* | **POST** | Integer **id** : ID du lieu | Crée un nouveau lieu |
| */location/edit/{id]* | **POST** | Integer **id** : ID du lieu | Édite un lieu existant |
| */location/delete/{id]* | **POST** | Integer **id** : ID du lieu | Supprime un lieu existant |
| */login* | **POST** | String **login** : Nom d’utilisateur  String **password** : Mot de passe | Authentification à l’API |
| */newUser* | **POST** | String **lastname** : Nom  String **firstname** : Prénom  String **address** : Adresse  String **phone** : Téléphone  String **email** : Email  String **username** : Nom de l’utilisateur  String **password** : Mot de passe | Création d’un nouvel utilisateur |
| */profile/{id]* |  | Integer **id** : ID du profil | Retourne les informations d’un utilisateur |