

Projet Bibliothèque

Patrons

Ersagun Yalcintepe

Julien Remy



MIAGE M2 SID

Université de Lorraine

2016 - 2017

November 7, 2016

Abstract

L'objectif de ce projet a été de réaliser un système d'information pour la mise en place d'une bibliothèque à partir d'un cahier des charges. Après avoir conceptualisé l'application grâce à une modélisation, nous avons développé une application web en utilisant JHipster. Le bibliothécaire peut de cette manière gérer l'ensemble des ressources via ce site web.

Sommaire

1	Introduction	1
2	Conception	1
3	Modélisation	2
4	Technologies	10
5	Développement de l'application	11
5.1	Serveur	11
5.1.1	Classe POJO	11
5.1.2	Classe Repository	11
5.1.3	Classe DTO	11
5.1.4	Classe Mapper	11
5.1.5	Classe Resource	11
5.2	Client	12
5.3	Interface Graphique	12
6	Conclusion	14

1 Introduction

L'objectif de ce projet a été de réaliser un système d'information pour la mise en place d'une bibliothèque à partir de quelques données sur le fonctionnement de cette dernière.

Pour réaliser ce projet, nous nous sommes rapprochés le plus possible de la méthode agile. Nous avons donc avancé régulièrement sur le projet en essayant de se fixer des jalons pour les différentes opérations applicatives à mettre en place. Travaillant à deux sur ce projet, nous nous sommes répartis les tâches comme suit : Ersagun s'est plus penché sur la partie développement, alors que Julien sur la partie modélisation/conception. Bien entendu, nous nous faisons part de notre avancement très souvent, pour que l'application corresponde en tout point au modèle réalisé. Il a fallu en revanche revoir la conception, puisque nos technologies utilisées ne nous permettaient pas de suivre les schémas proposés à la lettre. Nous avons utilisé un gestionnaire de version (GitHub) pour pouvoir partager le code et ainsi travailler en commun. Nous nous sommes servis d'un RaspberryPi pour héberger la base de données. De cette manière, nous pouvions interagir sur les mêmes données. A savoir que le site était également hébergé sur ce micro-serveur à l'adresse <http://ersagun.ga:8080>. Afin d'avoir une base de données complète, nous avons extrait des livres à partir d'un site web : <http://www2.informatik.uni-freiburg.de/cziegler/BX/> au format txt. Nous avons ensuite normalisé les données pour qu'elles correspondent au schéma souhaité.

2 Conception

L'application est uniquement accessible depuis le bibliothécaire. Les usagers passeront donc par lui pour réaliser leurs emprunts. La bibliothèque dispose de nombreuses œuvres, de type livre ou magazine. Chaque œuvre peut avoir plusieurs exemplaires, c'est-à-dire en plusieurs, ou bien des œuvres sorties à des dates différentes ou par un éditeur différent. Un usager peut emprunter un exemplaire pour un temps donné. Nous avons choisi d'utiliser un booléen disponible pour les exemplaires afin de simplifier et d'optimiser l'application, qui passe à faux lorsqu'ils sont empruntés. Aussi, nous gérons l'état des livres, c'est-à-dire à quel point ceux-ci sont abîmés. Il y a 4 états représentés :

0 : illisible 1 : mauvais état 2 : bon état 3 : neuf. Si aucun exemplaire de cet œuvre n'est disponible, il peut réaliser une réservation. De cette manière, au retour d'un exemplaire de cet œuvre ou lors d'un achat, il pourra par la suite être notifié. S'il réalise l'emprunt, la réservation est donc annulée. Les réservations ne sont pas conservées en base de données, nous stockons uniquement l'ensemble des emprunts. Lorsqu'un usager rapporte un exemplaire, le bibliothécaire doit vérifier l'état de l'exemplaire et le modifier s'il a été abîmé. Le bibliothécaire est alerté si jamais il effectue une réservation pour un usager sur une œuvre dont un exemplaire est disponible.

Une fois le concept de l'application bien pris en main, nous nous sommes penchés sur la modélisation. Une modélisation du système d'informations a été primordiale pour visualiser la solution à réaliser d'un point de vue global, mais aussi précis et technique. De plus, tout le monde est capable d'interpréter ses schémas afin de comprendre de quelle manière le projet va se construire et comment se déroule les processus. Nous avons donc utilisé le langage UML (Unified Modeling Language) pour représenter ces informations en suivant la méthode RUP (Rational Unified Process).

3 Modélisation

Il nous a fallu d'abord identifier les acteurs, et les interactions que ceux-ci réalisaient avec le système. Pour cela, nous avons mis en place un diagramme de cas d'utilisation. Ici, on a considéré que le seul acteur est le bibliothécaire qui demande les informations à l'usager. En effet, un usager n'a pas d'accès direct au système, il communique avec le bibliothécaire qui emprunte un livre pour lui. Une œuvre ne pourra en aucun cas être modifiée.

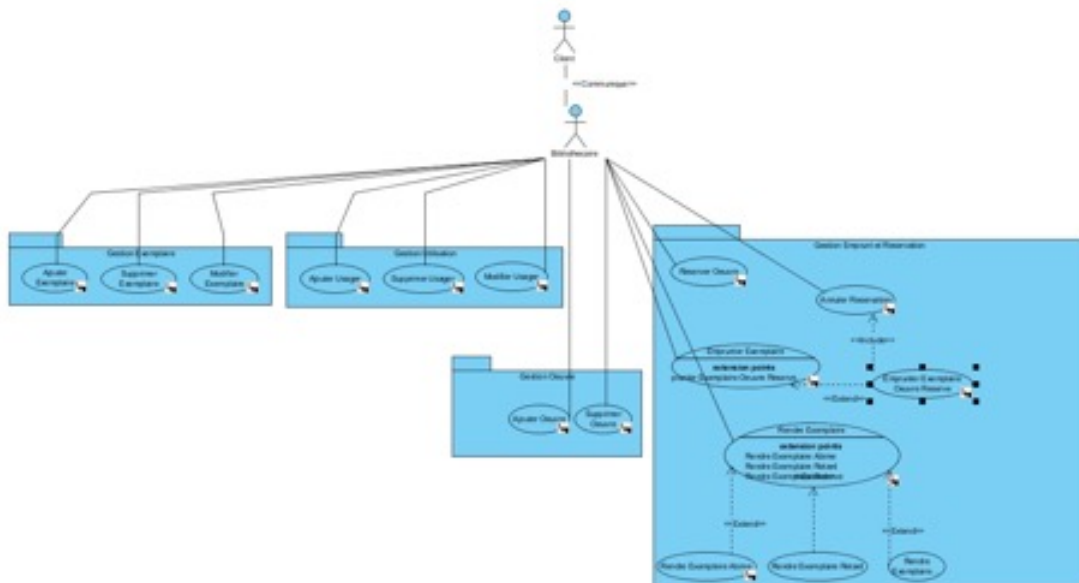


Figure 1: Diagramme de cas d'utilisation

Ensuite, nous avons réalisé des diagrammes d'activités qui permet de connaître pour chaque use case, les différentes étapes pour mener à bien l'action souhaité. La plupart de ceux-ci sont triviaux puisqu'ils correspondent à des opérations du CRUD (Create – Read – Update –Delete), nous avons donc réalisés ceux qui étaient les plus pertinent.

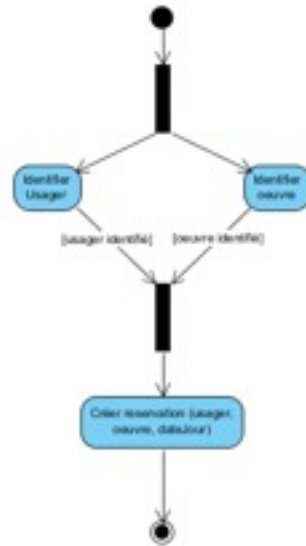


Figure 2: Exemple - Réservation d'une Œuvre

Après cela, nous avons dû identifier l'ensemble des classes ou objets pour traiter au mieux la solution. Par exemple, l'objet Usager permet de caractériser un client et faire une liaison avec un emprunt lorsque celui-ci en réalise un. A savoir que nous avons simplifié les dates par des Long dans ce diagramme (Visual Paradigm ne proposant pas de format Date). Uniquement les classes métiers sont représentées ici :

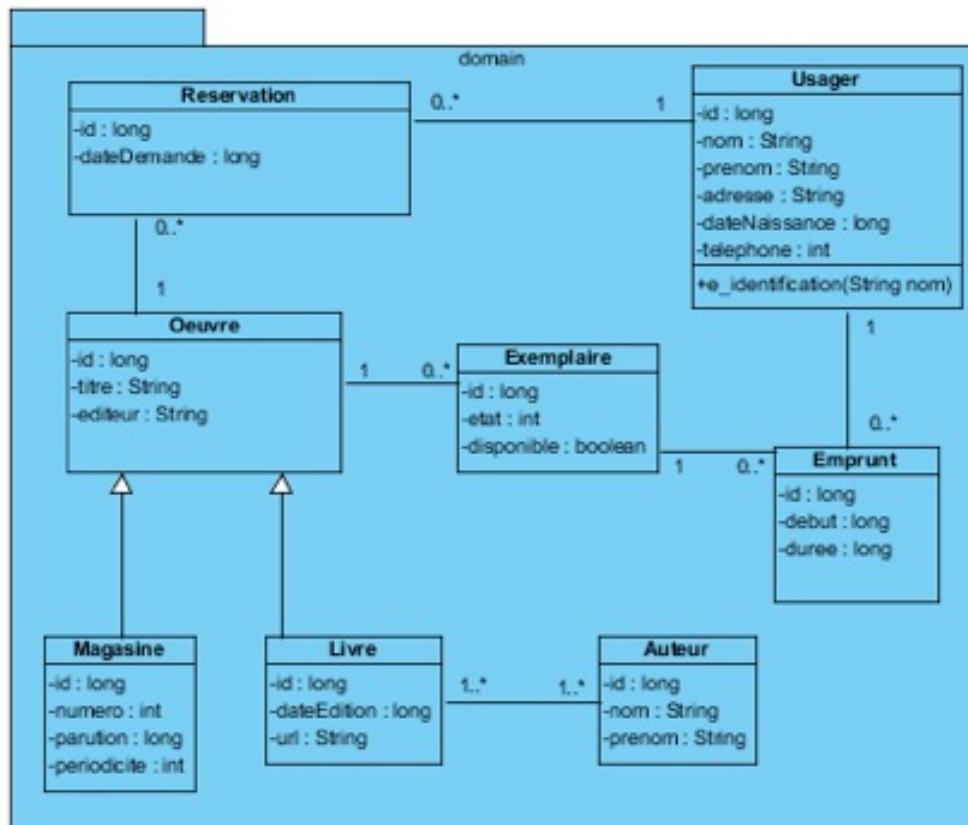


Figure 3: Diagramme de classe

Une fois le diagramme de classes établi, il a fallu mettre en place les diagrammes de séquences, qui vont mettre en évidence l'arborescence des appels méthode entre les différentes classes du système. Le diagramme de séquences nous montre aussi la nécessité d'implémenter une interface graphique, ainsi que des contrôleurs. Ces classes font en fait le pont entre un clic du bibliothécaire et le processus fonctionnel au niveau des objets puis de la base de données. Nous expliquerons dans la partie développement l'enchaînement des méthodes et des classes appelés.

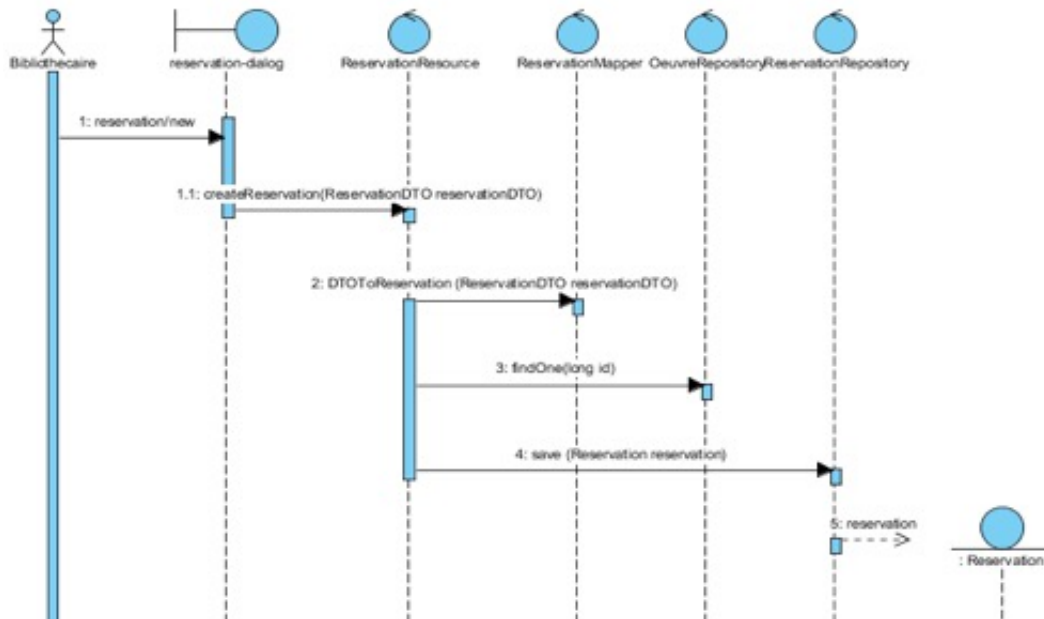


Figure 4: Exemple – Réservation d'une Œuvre

Nous avons pu ensuite définir pour chaque objet, les différents états qu'il peut prendre. Un diagramme d'états permet d'avoir une vue d'ensemble des comportements d'un objet en particulier, sans pour autant suivre un scénario (ou cas d'utilisation). Pour le diagramme d'état d'un exemplaire, nous pouvons voir que le livre change d'état (abimé ou non).

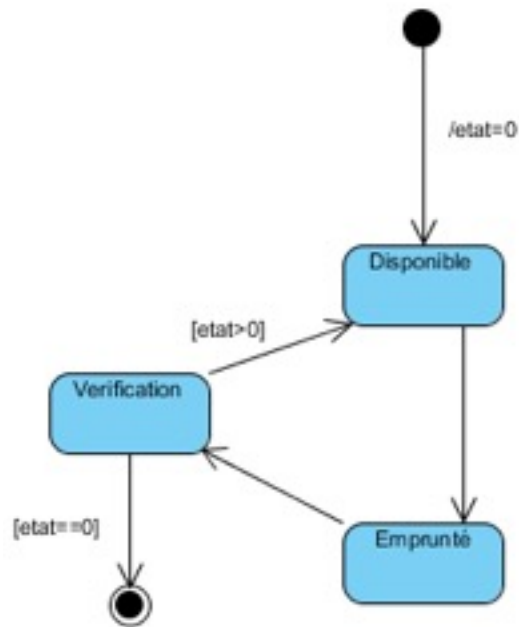


Figure 5: Exemple – Etat d'un Exemplaire

Nous avons ensuite réalisé un diagramme de package pour comprendre l'architecture global de notre programme et mettre en évidence les rôles correspondants. A savoir que les technologies que nous avons utilisées génèrent automatiquement l'organisation des classes en package. Nous avons synthétisé le système un maximum et montrer les deux packages principaux qui font fonctionner l'application.

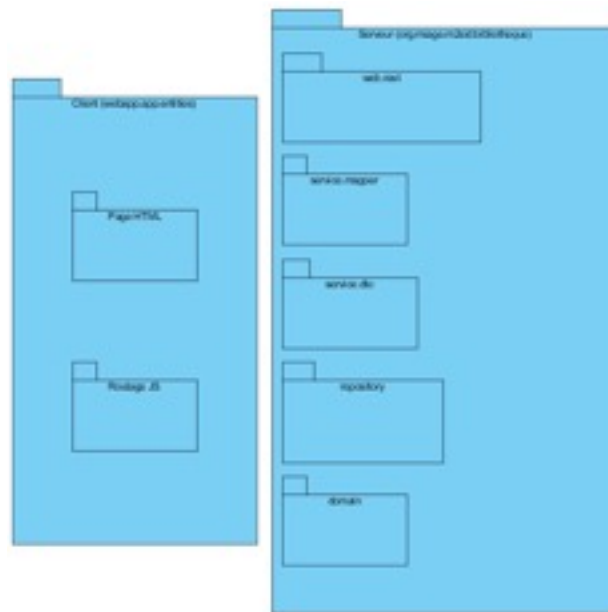


Figure 6: Diagramme de paquetage

Puis, le diagramme de composants qui permet de mettre en évidence les relations entre chaque package et ainsi comprendre leur rôle et leur place dans le fonctionnement de l'application.

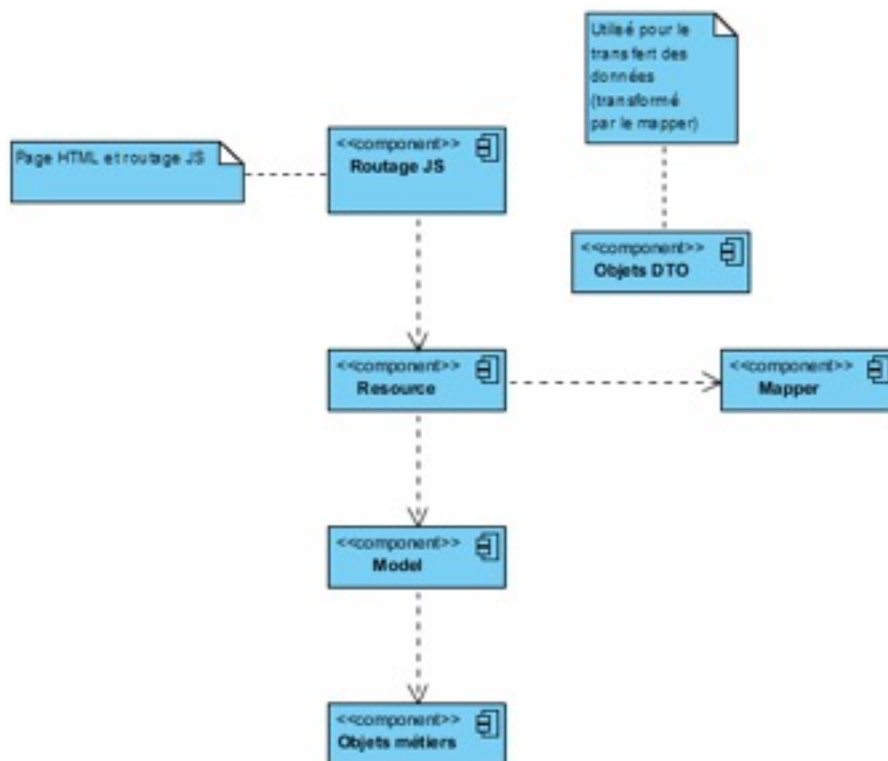


Figure 7: Diagramme de composants

Finalement, pour montrer comment l'application allait être utilisée, nous avons réalisé un diagramme de déploiement. Cela explique de manière plus physique et non fonctionnel les liens entre chaque composant.

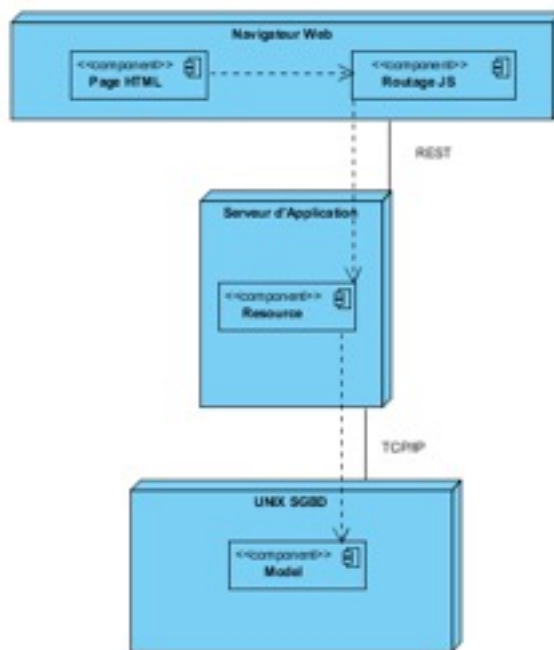


Figure 8: Diagramme de déploiement

4 Technologies

Pour pouvoir implémenter facilement la solution à partir du modèle de données, nous avons décidé d'utiliser JHipster. C'est un générateur d'application web très complet et puissant qui allie Yeoman, SpringBoot et AngularJS.

JHipster fournit JDL studio, un outil qui génère un schéma UML et l'ensemble des classes à partir d'une syntaxe simpliste. La génération des classes, des interfaces de classes se génèrent automatiquement une fois le modèle importé. Yeoman est un ensemble de fonctionnalités, fonctionnant sous Node.js qui permet de créer la structure du projet, avec modularité et simplicité. SpringBoot est un micro-framework qui permet de mettre en place une application Spring facilement configurable avec un serveur embarqué. AngularJS est un framework javascript qui permet de développer la partie client aisément basé sur le modèle MVC (Model-View-Controller).

JHipster nous permet donc finalement d'obtenir un projet Java côté serveur très complet et performant. Nous avons également utilisé le framework Hibernate qui permet de faire la relation entre la base de données et les objets Java, ce que l'on appelle le mapping relationnel. Doté d'une surcouche JPA (Java Persistent API), des annotations permettent de directement faire le mapping sur les objets POJO (Plain Old Java Object) côté Java. De cette manière, on manipule des objets persistants, c'est à dire qu'ils correspondent exactement à ceux présents en base de données. Nous avons utilisé MariaDB comme SGBD (Système de Gestion de Base de Données), étant open-source et s'apparentant à MySQL.

A noter que l'ensemble du projet fonctionne aussi avec maven, c'est à dire que toutes les dépendances et les exécutions sont gérés par un fichier de configuration "pom.xml". Grâce à ce choix de technologie, nous avons certes une application quelque peu lourde côté serveur, mais très facilement maintenable, modulable et avec une grande scalabilité. En effet, le modèle MVC permet de bien différencier les différents membres de l'application et ainsi aucune répercussion n'a lieu si on modifie l'un d'entre eux. Aussi, JHipster permet de déployer une application web en très peu de temps une fois la configuration faite. Petit bémol cependant, l'architecture générée par SpringBoot est trop complexe néanmoins pour proposer des diagrammes de séquences clairs car de nombreuses opérations s'exécutent mais toujours en respectant le modèle MVC.

Voici une brève présentation du fonctionnement du code de l'application. JHipster fournit en effet énormément de classes et nous allons donc nous pencher sur celles qui prennent vraiment part au processus. Il faut savoir que l'application est découpée en deux parties : le côté client qui correspond aux pages web et donc à l'interface homme machine et le côté serveur qui lui s'occupe de tous les traitements (de la base de données au renvoi des valeurs côté client). A savoir que le client et le serveur communique via des appels REST qui contiennent du format JSON. Le package du projet : `org.miage.m2sid.bibliotheque`

5 Développement de l'application

5.1 Serveur

Le code du serveur est contenu dans `src/main/java/`

5.1.1 Classe POJO

Les POJO (Plain Old Java Objects), c'est à dire les classes de base comme `Emprunt`, `Auteur`, `Usager` ou `Exemplaire` sont dans le package `domain`. Elles sont serializable et elles représentent l'état de l'objet dans la base de données. Ce sont ces classes qui sont persistés dans la base de données grâce à JPA et Hibernate.

5.1.2 Classe Repository

Les interface `Repository` sont dans le package `repository`. Ces interfaces extends de la classe `JpaRepository<NomDePOJO, Long>`. La classe `JpaRepository` contient les méthodes qui interagissent avec la base de données. Nous pouvons trouver les méthodes `save`, `delete`, `update`, `findAll`, `findOne` ainsi que des requêtes préétablis. L'application fait donc appel à ces classes pour interagir avec la base de données. Elles sont de type `NomDeClasseRepository`.

5.1.3 Classe DTO

Les DTO (Data Transfer Objects) sont dans le package `service.dto`. Les objets de transfert de données ont pour objectif d'encapsuler les données. Ce sont ces types d'objets qui sont envoyés à travers le réseau au format JSON en REST. Pour chaque objet, on a des donc des classes `NomDeClasseDTO`.

5.1.4 Classe Mapper

Nous disposons donc également de `Mapper` dans le package `service.mapper` qui nous permet, à partir d'un DTO, "d'instancier" l'objet correspondant. Ils sont écrits sous la forme `NomDeClasseMapper`.

5.1.5 Classe Resource

Lorsque nous effectuons une action côté client (sur le site web, par le bibliothécaire), le clic sur des boutons redirigent en fait dans la sous-arborescence de l'adresse pour effectuer des appels REST vers le serveur. Les classes qui traitent et font donc l'objet de web service se trouvent dans le package `web.rest`. Ils se présentent sous la forme `NomDeClasseResource`. Ce sont ces classes qui effectue la véritable gestion de l'application. Chaque classe de ressource contient un attribut de type `Repository` comme vu plus haut et récupère ou enregistre les données à partir

de la base de données en faisant appel aux méthodes de cet objet. Ensuite il envoie le résultat à travers le réseau.

5.2 Client

Le code du client est contenu dans `src/main/webapp/app/entities/`

Notre client utilise le framework AngularJS. Le dossier `entities` contient l'ensemble des classes de notre modèle. On retrouve des pages html qui vont correspondre à différentes actions grâce à des appels REST. Par exemple, `adresse:8080/reservation/new` permet de créer une nouvelle réservation. Point important: lorsque l'on effectue une réservation ou un emprunt par exemple, le livre et l'usager est sélectionné directement depuis la page web. C'est à dire qu'une fois la sélection réalisée, l'API REST va envoyer au format JSON un objet `ReservationDTO` qui va déjà contenir les informations pour cet objet (Usager et Oeuvre). C'est pourquoi nous n'avons pas à identifier usager et œuvre côté serveur, elle est réalisée en amont côté client. En revanche, nous vérifions que l'œuvre est bien présente en base de données. A noter que ce sont des entités javascript qui vont faire le routage vers les différents web services du serveur après cela.

Voici un schéma illustrant de manière générale l'architecture de notre application :



Figure 9: Architecture de l'application

5.3 Interface Graphique

Le bibliothécaire pourra accéder au système d'informations via un site web. Il est nécessaire de se connecter en temps qu'admin pour pouvoir gérer les usagers et les œuvres. L'interface rassemble un ensemble d'onglets permettant d'accéder aux livres, magazines, emprunts, réservations et usagers présents dans la base de données. Pour chacune de ces entités, on peut en ajouter, en modifier et en supprimant (sauf quelques exceptions). Par exemple, lorsque le bibliothécaire réalise un emprunt pour un usager, il doit se diriger vers la section Livre et sélectionner l'exemplaire que l'usager souhaite emprunter. Il sera ensuite redirigé vers une fenêtre modale et il pourra choisir la durée et l'usager.

Depuis la section Livre, le bibliothécaire peut voir l'ensemble des exemplaires disponibles. Il peut donc directement effectuer un emprunt. Si aucun exemplaire n'est disponible, il aura la possibilité d'effectuer directement une réservation depuis cette page.

Voici l’affichage de la liste des livres présents dans notre base de données. Nous pouvons depuis cette page ajouter une œuvre. Le clic sur un de ces livres nous permet d’accéder à l’œuvre et voir l’ensemble des exemplaires disponibles.

Livres

[+ Ajouter un livre](#)

Titre	ISBN	Date d'édition	Auteurs	Resume
Clara Callan	2005018	2000	Richard Bruce Wright	Resume du livre
Classical Mythology	195153448	2001	Mark P. O. Morford	Resume du livre
Flu: The Story of the Great Influenza Pandemic of 1918 and the Search for the Virus That Caused It	374157065	1998	Gina Bari Kolata	Resume du livre

Figure 10: Affichage des livres

Cette page correspond à l’ensemble des emprunts réalisés actuellement sur la base de données. On voit les id des usagers, l’exemplaire correspondant ainsi que la durée. Le bibliothécaire peut rendre l’emprunt depuis cette page lorsque l’usager retourne son exemplaire.

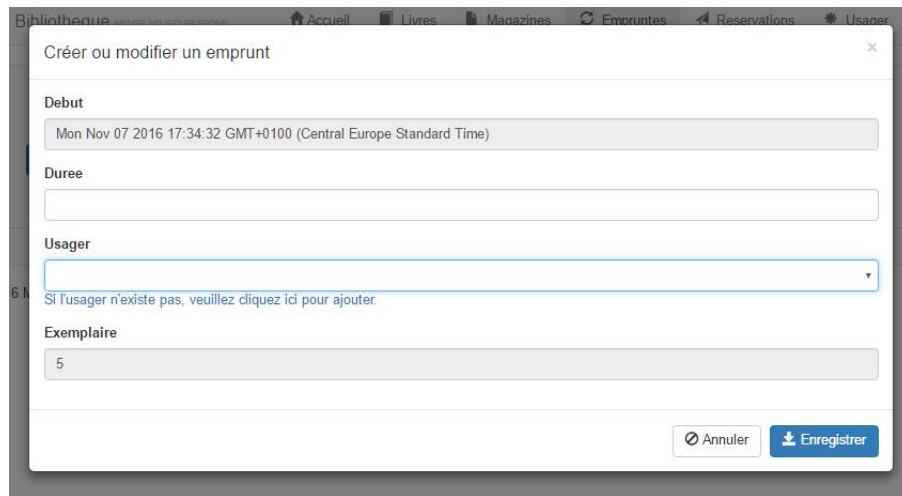
Emprunts

[+ Créer un Emprunt](#)

ID	Debut	Duree	Usager	Exemplaire	
1	Nov 7, 2016 5:37:33 PM	5	1	3	Rendre
2	Nov 7, 2016 5:43:11 PM	4	1	43	Rendre

Figure 11: Affichage d’un emprunt

On voit ici la fenêtre modale ouverte pour la création d'un emprunt. Ici, le bibliothécaire peut sélectionner la durée (la date de début étant fixée à la date actuelle), et l'utilisateur correspondant. L'exemplaire est choisi au préalable dans la liste de ceux disponibles.



Créer ou modifier un emprunt

Debut
Mon Nov 07 2016 17:34:32 GMT+0100 (Central Europe Standard Time)

Duree

Usager

Si l'utilisateur n'existe pas, veuillez cliquer ici pour ajouter.

Exemplaire
5

Annuler Enregistrer

Figure 12: Création d'un emprunt

6 Conclusion

Nous sommes tout d'abord satisfaits d'avoir pu mettre le site en ligne depuis un RaspberryPi. L'application peut s'apparenter à un projet concret, et chaque testeur va donc obtenir les mêmes données. Ce projet a été enrichissant puisque nous avons développé la modélisation jusqu'au déploiement de l'application, ce que nous n'avions jamais fait auparavant. Nous avons aussi appris à gérer de nombreuses technologies (JHipster, AngularJS, SpringBoot) qui sont dans la mouvance actuelle informatique.