Solal Gaillard

13406890

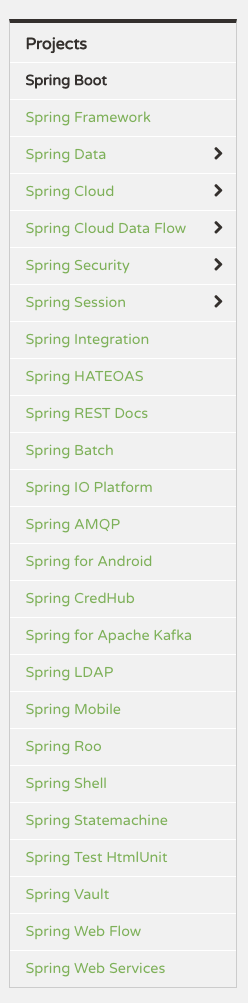
*Rapport de Réalisation de Programme : Qinetic*

Qinetic était un projet ambitieux. Il s’agissait de réaliser une Web App complexe avec un stack de différents langages de programmation très varié. Je voulais explorer et utiliser toutes les techniques de pointe qui aujourd’hui façonnent le domaine des applications web.

Le projet se devait donc d’utiliser des *Frameworks* actuels que l’on trouve dans le monde professionnel et qui peuvent parfois prendre plusieurs mois à maîtriser. Je dois avouer ne pas vraiment avoir réaliser en amont à quel point une application web était complexe en terme d’éléments impliqués. Il y a des enjeux de sécurité, de transmission de données, de représentation intermédiaire des données afin de les transférer puis de les présenter dans un navigateur. Il y a aussi la gestion d’un serveur, le déploiement d’une daemon pour lancer le programme, etc… Il n’est pas étonnant que le développement web se décompose en fait en trois métiers avec trois champs d’expertises différents : *front end*, *back end*, et *DevOps*. Il m’a fallu beaucoup de temps, de réflexions, d’aller-retour pour élaborer une architecture qui tenait la route. Je décidai de réaliser ce qu’on appelle une SPA, *Single Page App*, c’est-à-dire une page web unique qui télécharge et manipule les données dont elle a besoin pour fonctionner de manière complétement asynchrone.

Puisque gérer les *events handlers* avec ce qu’on appelle du *vanilla Javascript* aurait relevé du cauchemar, je mis mon dévolu sur *React*, une *libraire/Framework* *open source* développé par Facebook. En effet, j’aurais pu décider de suivre les patrons de design *OOP* pour développer l’application au niveau du *front end* mais *React*, en tant que *Framework*, permettait d’abstraire la conceptualisation architecturale pour se fendre dans un modèle organisationnel préétabli et diablement efficace relevant d’une approche modulaire et sémantique à la fois. Un magasin de données dans le scope global permettait de maintenir une trace des états successifs de l’application (*Redux*), les composants *React* se chargeant de les représenter. Ne connaissant rien à *React*, je pris comme référence la documentation officielle que l’on peut trouver à *https://reactjs.org/docs/getting-started.html*, le github de la communauté (*https://github.com/reactjs*) et l’excellent livre *Learning React: Functional Web Development with React and Redux* d’Alex Banks et Eve Porcello. J’y intégrais grâce à *npm* les packages permettant d’introduire le store global *Redux* et le middleware *Thunk*. J’utilisais aussi la dernière syntaxe de *JavaScript* ainsi que la syntaxe expérimentale pour définir des méthodes avec des fonctions fléchées grâce à *Babel*. Connaissant déjà un peu le monde du front end, utiliser *React* ne fut pas la plus complexe des technologies à maîtriser. En effet, *HTML5* et *CSS3* m’étaient déjà très familiers même si je laissais souvent ouvertes dans mon navigateur les pages de références du *W3Schools* (*https://www.w3schools.com/*) et de *Mozilla* (*https://developer.mozilla.org/fr/docs/Web/API*). J’utilisais un préprocesseur *CSS*, *Sass*, qui permettait une meilleure syntaxe et l’utilisation de variables et de logique au sein même des *stylesheets*.

Utiliser *React*, c’était prendre la décision de l’obtention des données non pas par templating mais par des appels au serveur de manière asynchrone. Malheureusement ou heureusement, *http* est un protocole qui ne garde pas les états en mémoire. Une ressource doit donc être délivrée au bon moment au sein du programme et à la bonne personne. Comment gérer les réponses individualisées par utilisateur, comment sécuriser un accès à une certaine ressource ? Il existe évidemment plusieurs réponses à ces questions et il est toujours complexe de trouver la bonne quand on conçoit une *RESTful API*. Je mis mon dévolu sur les *JSON Web Tokens*. Il ne s’agit pas de la réponse la plus sécurisée mais elle offrait un bon compromis entre nombre de requêtes nécessaires pour la livraison des données et sécurité. Afin de mieux comprendre ce qu’est une *RESTFul API* et *HATEOAS*, je lus *RESTful Web Services* de Leonard Richardson et *REST in Practice: Hypermedia and Systems Architecture* de Jim Webber. Le token est livré dans l’en-tête http. Il s’agit d’une chaîne de caractère encodé avec l’algorithme *HMAC* et *SHA-256*. Avec la clé privée, résidant sur le serveur, je pouvais d’une part vérifier la validité du token et d’autre part en extraire certaines valeurs codées dans le token même comme l’identifiant de l’utilisateur façonnant la requête par exemple. J’utilisais un service tiers pour implémenter ce système d’authentification. Leurs APIs me permettaient de récupérer la clé (charger à nouveau toutes les 24 heures sur mon serveur) et de valider les tokens. Les identifiants et mots de passes étaient quant-à-eux stockés dans leurs bases de données. Il s’agit d’*auth0 (https://auth0.com/)* dont le service est gratuit pour toute utilisation limitée à moins de 7000 utilisateurs ou pour tout projet *open source*. J’apprenais donc plus en détail les ressorts du protocole *http* et surtout *https* pour pouvoir effectuer toutes ces communications entre *front end* et *back end* ainsi que *back end* et *auth0*. Utiliser *https* voulait dire encrypter les données. J’étais donc obligé de créer un certificat auto-signé avec *openssl* stocké dans un fichier *PKCS#12* et utilisé par mon serveur Tomcat hebergé sur une machine *Ubuntu* sur *AWS*.

Bien évidemment, il me fallait donc un serveur qui réponde aux requêtes de ma *SPA* et serve les données nécessaires. Je ne connaissais pas *Java* mais comme il me semblait qu’il s’agissait d’un standard dans le monde du développement d’application web et un standard dans le monde du développement objet, je jetais mon dévolu sur ce dernier. Je pense aujourd’hui que ce fut une erreur même si l’apprentissage résultant est indéniablement plus riche que si j’avais utilisé *python, ruby*, ou *JavaScript*. Premièrement, *Java* est extrêmement rigide dans sa structure, pas question d’y déroger avec sune classe publique par fichier, son système de packages et d’importations. Deuxièmement, c’est un langage typé, hors travailler avec une application web est toujours un jeu d’équilibriste, de prototypage et d’aller-retour entre une représentation des données dans une certaine structure vers une autre. Affiner la partie *front end* demande donc, en contrepartie, qu’on affine la partie *back end*. Java se prête mal au jeu. Je lu Core 1 et Core 2. Les ressources sur Java sont faciles d’accès et le langage assez facile à comprendre. En revanche l’utilisation de Spring, un framework pour Java afin de servir des ressources en JSON aux requêtes du *front end* se révélait être un cauchemar. Il y a peu de ressources en ligne, des versions différentes avec des syntaxes différentes, des classes différentes, des annotations différentes, une documentation officielle peu lisible car trop technique et avec trop peu de recul (<https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/index.html>). Une grande difficulté d’appréhension donc pour le néophyte. J’utilisais souvent les tutoriaux de baeldung. Néanmoins je réussissais à construire mes premières routes et à suivre le MVC proposé par le framework. Nous noterons aussi l’absence de livre de référence pour le néophyte car il s’agit peut-être d’un framework trop extensif comme le suggère la collection de tout ses dérivés. Afin de compiler l’archive java, j’utilisais Gradle, un build automation tool qui me permettait d’appeler aisément mes dépendances comme justement Spring. Mon IDE était IntelliJ IDEA qui ne présentait pas énormément d’avantages. Bien qu’ayant suivi le cours d’objet orienté, construire une application complexe nécessitait de connaître les patrons de design OOP, Je fus obligé de lire l’incontournable GOF mais aussi HEADLESS FIRST qui résumait les mêmes patrons en Java.

Il y avait aussi la question de la permanence des données, une base de données PostgreSQL me semblait tout ce qu’il y avait de plus correct et moderne. Je mettais d’abord en ligne cette base de données grâce à AWS, les services d’Amazon. Je m’y connectais ensuite avec DataGrip, un logiciel lui aussi de JetBrains. Mon serveur s’y connecte grâce à Hibernate, une implémentation de JPA. Hibernate fournit ce qu’on appelle un ORM, *Object Relational Mapping*. Cela permet une correspondance entre les objets Java et les données de la base de données. Cela permet aussi une interface simplifiée pour récupérer, écrire, détruire, modifier les rangs des tableaux de la base.

J’aurais aimé pousser plus loin le projet. J’ai commencé seul, j’ai eu deux étudiants qui se joints à moi. J’avais rencontré Kamel à Paris, et eu Théo au téléphone. Malheureusement, ils ont abandonné le projet sans jamais l’avoir commencé. J’ai essayé de les pousser, de les impliquer. Je leur avais assigné des tickets sur Jira, avait organisé un rendez-vous hebdomadaire au téléphone qui ne s’est produit qu’une fois. Je crois que le projet était bien trop ambitieux et qu’il leur a fait peur. Ils n’étaient pas non plus intéressés par le développement web. Théo était censé être responsable de la mise en place de tests unitaires pour *React* et *Java*, Kamel, de réaliser les premiers *stubs* serveurs pour faire fonctionner un *front end*. Comme Github peut en attester, ils n’ont pas fait un seul commit. J’ai tout fait, du devops, back end, front end et base de données. Je suis assez fier d’avoir pu maîtriser et comprendre une vaste partie des outils du dévelopement d’une application web même si je suis déçu de n’avoir pu pousser le projet plus loin.

J’aurais aimé comprendre mieux l’authentification et la gestion de session au lieu de déléguer à auth0 la validation des tokens ainsi que l’emmagasinage des identifiants et mots de passes. J’aurais aussi aimé connaître mieux Spring et tous les autres aspects du framework auquel je n’ai pas touché. Je me demande aussi si une autre approche que top down est possible avec un projet pareil. Aurais-je pu avoir une approche bottom-up, cela me semble difficile, tant toutes les parties s’imbriquent les unes avec les autres et sont co-dépendentes.

Il est difficile de faire un tour d’horizon de toutes les ressources explorées en ligne pour la réalisation de ce projet. J’en cite déjà beaucoup dans le corps du rapport. Bien entendu, sans stackoverflow ou stackexchange, il n’y aurait à ce jour toujours rien à voir sur le serveur AWS. Medium m’a aussi permis de prendre des décisions plus théoriques grâce à des articles qui questionnent les technologies elles-mêmes plus que leurs implémentations.

En ce qui concerne mon parcours professionnel, il m’est très facile de discourir et de justifier l’intérêt que représentait pour moi un tel projet. J’ai commencé à travailler en 2013 à New York comme *Interactive Designer and Developer*. A l’époque, j’avais réalisé deux sites internet et fais un peu de graphisme papier. C’est ce qui me permit de devenir développeur *front end*. Difficile néanmoins de cacher mes limites et mon ignorance profonde de l’informatique. Néanmoins, j’avais un travail et je pouvais faire un peu mes armes. Je commençais à maîtriser HTML5 et CSS3 mais cela ne suffisait pas. Voilà pourquoi je me suis inscrit en licence à Paris 8. Depuis je continue à développer mes compétences au niveau *front end* et *back end*, aidé par des connaissances théoriques. Il est important de noter l’usage intensif du cours de système d’exploitation pour la gestion du serveur par exemple, ou du cours sur les bases de données pour la gestion de PostgreSQL. Il en va de même pour le cours de réseau ainsi que tous les autres cours qui même plus théoriques m’ont permis d’acquérir une compréhension plus holistique et profonde de la science informatique. Je veux à terme avoir une véritable expertise comme développeur *fullstack*. Bien entendu, la licence de Paris 8 ne me permettra pas d’acquérir, seule, cette expertise mais elle est complétement nécessaire pour me mettre sur son chemin. Ce projet m’a donc permis d’explorer plusieurs facettes du développement d’applications web en considérant les technologies en vogue sur le marché du travail et me laissant les manipuler chacune.