# **Présentation de l’application**

Nous avons réalisé une application de gestion de devis. Nous pouvons gérer deux types de devis : immobilier et automobile.

Avant de pouvoir créer de nouveaux devis, l’utilisateur doit s’authentifier. Les utilisateurs sont enregistrés en base. Vous trouverez plus d’informations à ce sujet dans la partie « Présentation de l’architecture ».

Une fois authentifié, l’utilisateur peut choisir le type de devis qu’il veut créer. Le déroulement de la création d’un devis est toujours le même : l’utilisateur remplit un formulaire puis passe à l’étape suivante. A tout moment, il peut revenir sur les étapes précédentes pour modifier les informations déjà entrées. Il peut également quitter la création du devis en cours et la reprendre par la suite.

Un écran listant tous les devis est disponible. Il contient les devis en cours et terminés. Les devis terminés ne sont disponibles qu’à la consultation. Il est possible d’éditer un devis en cours.

Bien entendu, l’utilisateur peut également se déconnecter.

# **Présentation de l’architecture**

L’application se base sur les technologies suivantes : Java, JEE, Javascript (pour quelques appels AJAX) et Vert.x. Elle est découpée en 2 applications :

- D’une part il y a une application « front » qui est réalisée avec JEE, c’est elle qui fournit l’IHM. Cette application possède sa propre base de données dans laquelle sont enregistrés les devis et leurs états. A chaque fois qu’un utilisateur passe à un nouveau wizard, l’état du devis est enregistré en base. Dans la BDD sont aussi sauvegardés les différents types de contrats (tous risques etc…)

- D’autre part, il y a une application « back » réalisée avec Vert.x et Java. Nous parlons ici simplement d’une application qui fournit une API qui est attaquée par l’autre app. Cette appli a aussi sa propre base qui contient les utilisateurs et toutes les données nécessaire à la création des devis (les modèles de véhicules, les types de carburants et d’habitations etc)

Ces deux applications dialoguent souvent.

**Lors de la connexion**

Lorsqu’un utilisateur se connecte, la requête est envoyée à Spring Security. Un objet de configuration définit comment Spring Security se comporte : ce dernier fait appel à une api fournie par Vert.x (/api/login). Vert.x va se charger de générer un token unique (via JWTAuth) et d’authentifier l’utilisateur. Une fois cela terminé, il répond à Spring Security et une sauvegarde de l’utilisateur connecté en Session est réalisée.

**Lors des différents wizards**

Les informations nécessaires aux wizards étant sauvegardées dans la BDD qui va avec Vert.x, l’application Spring doit demander des informations à vert.x en fonction des inputs des utilisateurs. Quand un utilisateur sélectionne une option (par exemple un modèle de voiture), une requête AJAX est exécutée pour récupérer les informations relatives à ce modèle (par exemple le type de carburant) et ainsi de suite.

## **Architecture de la partie Spring**

### Spring dispose d’une architecture MVC, les vues sont appelées par des controllers. Les controllers appellent des modèles qui eux même appellent des services (connexion à une BDD par exemple) afin de retourner les données et les injecter dans les vues.

### **Les vues**

En ce qui concerne les vues de type ‘non formulaire’ le controller renvoie simplement la page jsp concernée.

Pour les vues de type ‘formulaire’ un modelWizard spécifique (habitation ou véhicule) est envoyé à la page. Il est instancié lors de la première étape du devis puis il est surchargé entre chaque nouvelle étape.

Une fois le devis validé par l’utilisateur, l’objet Wizard est converti en entité de BDD pour être inséré en base.

Pour peupler les formulaires, une requête http est effectuée côté client pour récupérer des données via Vertx / Spring comme par exemple récupérer les marques des voiture, les modèles...

### **Spring Security**

Le système d'authentification côté IHM utilise Spring Security, il est configuré dans WebSecurityConfig.java. Nous avons protégé les pages nécessitant une authentification en préfixant leurs URL de "/private".

".antMatchers("/private/\*\*").authenticated()" permet donc de spécifier que l'accès aux routes contenant "/private" nécessite d'être authentifié.

Dans cette configuration sont également spécifiées les pages concernant l'authentification : page de login, action en cas de mauvaise authentification, page de logout, redirection en cas d'accès non autorisé...

L'authentification passe par un AuthenticationProvider personnalisé, la connexion devant être réalisée par VertX. Cet AuthenticationProviderService récupère les *credentials* afin de les envoyer à VertX. Si ce dernier autorise la connexion, les informations de l'utilisateur ainsi qu'un *token* généré par VertX sont renvoyés à l'AuthenticationProviderService. Ces données sont récupérées dans une classe VertXResponse.

En cas de succès, les attributs "token" et "user" sont renseignés, tandis qu'en cas d'échec ce sont "message" et "error" qui sont spécifiés.

Après avoir vérifié que l'attribut "error" est bien à null, le service ajoute le *token* et l'objet "user" dans la session afin qu'ils soient accessibles dans l'application. Enfin, l'AuthenticationProviderService retourne un objet Authentication, finissant d'authentifier l'utilisateur. Sinon, en cas d'échec d'authentification il rejette un événement de type "UsernameNotFoundException".

### **Base de données**

Nous avons fait le choix d’un serveur de base de données unique sur un serveur dédié afin d’être certains de partager les mêmes données et types de données.

Il y a une base dédiée à Vert.x (groupe4vertx) et une base dédiée à Spring (groupe4spring).

Il s’agit de 2 bases MySQL, moteur InnoDB pour la gestion des clés étrangères.

## **Architecture de la partie Vert.x**

La partie Vert.x n’a strictement rien de compliqué. En synthétisant au maximum, nous avons un verticle qui handle les différentes URL de l’API, exécute une requête sur la BDD et retourne un résultat. Comme c’est un pattern qui se répète énormément, nous avons créé une classe générique (entre autres) qui nous facilite la vie : Connector.

Cette classe offre une méthode qui abstrait les instructions suivantes :

- Récupération du client SQL

- Ouverture de la connexion

- Gestion de la réussite/échec de la connexion

- Récupération de l’objet SQLConnection

- Exécution de la requête passée en paramètre

- Gestion des erreurs en rapport avec la requête/son exécution

- Transformation du résultat pour correspondre avec la classe passée en paramètre.

- Retour du résultat

- Fermeture de la connexion au client SQL

D’autres classes ont été développées pour factoriser un peu le code. Par exemple, ReqError a une méthode ReqError.hurl() (je suis très fier du nom de cette méthode) qui permet de retourner une erreur un peu plus facilement. Une class Auth se charge de l’authentification.

Des models ont été créés. Ceux-ci correspondent avec la structure de la BDD côté Vert.x. Ils sont utilisés pour être passés au Connector et ainsi avoir des données correctement formatées, facilement.

# **Travail réalisé par les différents membres du groupe**

Comptes GitHub par personne :

Blaire Cador : **Termiton**

Nicolas Melin : **NicolasMelin**

Clément Flodrops: **mildful**

Joëlle Ferrari : **Jferra**

**Clément Flodrops**

Conception et développement de la partie Vert.x (seulement le code, pas la base), rédaction de ce document, et petits coups de main à droite à gauche.

**Joëlle Ferrari**

Spring Security, authentification, relecture de ce document.

**Nicolas Melin**

Création des wizards, quelques services, vues jsp.

**Blaise Cador**

Conception des BDD (Spring et Vert.x), peuplage des bases, enregistrement des devis en base.

# Bilan du projet par membre

## **Clément Flodrops**

### **Difficultés rencontrées**

Un petit temps d’adaptation à l’environnement Java. Ce n’était pas vraiment une difficulté en soit mais ça fait toujours perdre quelques précieuses minutes. Le plus compliqué fut de développer la classe Connector. La raison est simple : je voulais faire une classe générique mais je ne connaissais pas les outils à ma disposition fournis par Java pour le faire, de plus, je n’ai pas la « Java’s way of thinking ». Je pense que sans l’aide du professeur, j’aurais mis un bon bout de temps avant de trouver ceci :

Json.decodeValue(r.encode(), clazz)).collect(Collectors.toList())

En soit c’est à peu près tout. Je pense que j’aurais eu beaucoup plus de questions si j’avais travaillé sur Spring.

Une petite difficulté à noter : Joëlle et moi n’avons pas réussi à envoyer le token à Vert.x. Comment se fait-ce ? Nous avons utilisé l’API fetch() avec le polyfill pour faire nos appels Ajax. Pour passer le token dans les headers, l’API vient avec une nouvelle entité : Headers. Sauf que je n’ai jamais réussi à recevoir des headers customs settés par Joëlle. Donc la logique de mon code est bien là, mais nous n’avons pas réussi à l’exploiter.

### **Bénéfices**

- Java. Bien que je sois mauvaise langue et que j’aime bien tacler les langages que je ne pratique pas couramment, Java était sur ma liste de langages à approfondir. Ce projet m’a permis de faire mes premier vrais pas avec ce langage et depuis j’ai quelques side-projects en Java, donc ça m’a permis de me lancer et ça, c’est chouette. Indirectement, c’est aussi Java qui m’a un peu poussé vers TypeScript dans mes projets professionnels et personnels et cela me donne accès à pleins de possibilités d’architecture que je n’envisageais pas forcément (eh oui, les interfaces n’existent pas – encore – en javascript).

- Vert.x. Pareil que pour Java. J’ai un google sheet avec une liste de technos que je veux connaître, Vert.x en faisait partie et je suis bien content d’en avoir eu l’occasion.

- Conception. Je pense que ce projet (et les cours qui vont avec) m’ont permis de faire un petit pas au niveau de la conception d’applications.

## **Joëlle Ferrari**

### **Difficultés rencontrées**

J'ai mis du temps à comprendre le fonctionnement de Spring Security et le Java de façon générale. L'ajout du token dans les requêtes AJAX vers l'API VertX n'a pas pu être finalisée. Lors de l'ajout d'un paramètre dans le header de la requête, une erreur de configuration de la requête n'a pas été résolue.

### **Bénéfices**

Le projet m'a permis de mieux comprendre le fonctionnement de Java et des langages typés de façon générale. J'ai par ailleurs trouvé intéressant d'apprendre à avoir le réflexe à regarder dans les classes natives afin de comprendre comment utiliser un objet.

## **Nicolas Melin**

### **Difficultés rencontrées**

Apprentissage d’un nouveau langage. Même si le terme difficulté me paraît un peu excessif, le fait de devoir apprendre un langage notamment via un framework nécessite un temps d’adaptation pour bien comprendre le mécanisme et donc être performant.

### **Bénéfices**

Ce projet m’a permis d’apprendre un langage que je voulais travailler depuis quelques temps déjà ainsi que son framework Spring. Cela m’a également permis de travailler sur un nouvel IDE “IntelliJ” qui est après plusieurs jours d’utilisation est selon moi plus qu’agréable à utiliser.

## **Blaise Cador**

### **Difficultés rencontrées**

90% du temps j'ai galéré parce que je ne comprenais pas les tenants et aboutissants du framework, mais étonnamment le dernier jour, pendant les dernières 3heures, tout est devenu plutôt clair et j'ai pu travailler sereinement.

Le problème ne vient pas particulièrement de Java, mais plutôt de la visualisation des multiples liens qui relient entre eux tous les services/modèles/entités/controllers/jsp etc.

### **Bénéfices**

400€ .. euh non pardon, gros apprentissage du travail en équipe et de la coordination des push/pull/merge avec Monsieur Teub, Guy Teub.

Amélioration de la compétence de prise de recul pour mieux résoudre un problème.

J'ai également fait la connaissance d'intellij sous linux et sous windows, avec malheureusement une préférence pour la version Windows (moins buguée).

# **Bilan sur l’organisation**

Commençons par s’enlever une épine du pied en parlant de ce qui n’a pas marché.

Git. Alors que ce n’était pas trop mal parti, sur les 2 derniers jours la gestion du versionning est devenue beaucoup moins cadrée. Avec des dev directement sur master, des commit pushés sur master qui introduisent des bugs.

Bien que nous ayons également mis en Kanban en ligne, nous ne l’avons pas vraiment exploité et il était plus là pour la forme que pour son utilité.

Bien que les petites réunions en début de journées aient pu paraître dérisoire, elles nous ont permis d’essayer de se donner des objectifs à courts terme et donc de se motiver et bien avancer (même si les objectifs n’étaient pas toujours remplis, dans les temps).