RAPPORT DE PROJET
QUALITÉ LOGICIELLE

CRÉATION D'UNE PLATEFORME D'EMPRUNT DE MATÉRIEL INFORMATIQUE

Encadrants :
François SENIS
Andrey PIVIDORI

Réalisé par :
Albin CALAIS
Ellyn LAFONTAINE
Louis LECOEUR
Pauline PEREIRA MARTINS
Mallory TAFFONNEAU
Joshua VAN HOLLEBEKE

Introduction :

L'objectif de ce projet, au-delà des contraintes de réalisations, est avant tout de mettre en œuvre les principes fondamentaux de la gestion de projet, afin de nous familiariser avec ces pratiques qui garantissent la qualité du code produit, son bon fonctionnement, et surtout que le projet correspond bien aux attentes du client. Nous expliquerons donc ici les choix que nous avons fait, autant dans le choix des outils utilisés que dans les adaptations que nous avons fait par rapport à la demande d'origine. Nous ferons également un retour sur le travail accompli, notre organisation, ainsi que les difficultés rencontrées et leurs solutions.

Nos choix :

Le projet se compose de deux parties : un client web et un serveur. Pour le client web, nous avons décidé de l'implémenter en html/css. Le serveur quant à lui a été implémenté en Java, car c'est un langage que nous maîtrisons bien, et qui offre un très grand nombre de librairies, notamment pour toute la partie tests, mais aussi pour la gestion des stockeurs de données.

Les données de l'application sont stockées en fichiers texte au format JSON, car ces données sont peu volumineuses, et leur quantité ne justifiait pas selon nous la mise en place d'un système de gestion de bases de données, plus coûteux en temps. De plus, Java offre une bibliothèque de fonctions permettant la lecture et l'écriture de fichiers JSON très facilement. Enfin, en cas de problème, un administrateur pourra éditer les données manuellement.

Suivre le sujet original et les étapes indiquées nous a aidé. Le fait de rédiger les spécifications en amont a été une grande aide pendant la réalisation, nous avons travaillé chacun en parallèle sur le front/back et les tests.

Pour le backend nous avons utilisé la librairie Takes, elle permet de servir un serveur web simplement, elle offre de quoi faire du routage et un peu de SSR mais tout reste très manuel. Pour l'échange front-back nous avons créé plusieurs points d'entrée d'api et les communications se font par échange de texte/json. Nous avons mis en œuvre le pattern MVC, le modèle découlant directement des spécifications.

Pour le frontend nous avons utilisé js, les requêtes se font par xhr/fetch ce qui permet de ne pas rafraichir la page et d'afficher des messages d'erreurs lorsqu'une requête n'aboutit pas. Une importante partie des données affichées est récupérée par xhr et pas par SSR, c'était plus simple à réaliser et permettrait d'implémenter un hot-reload si le projet devait évoluer.

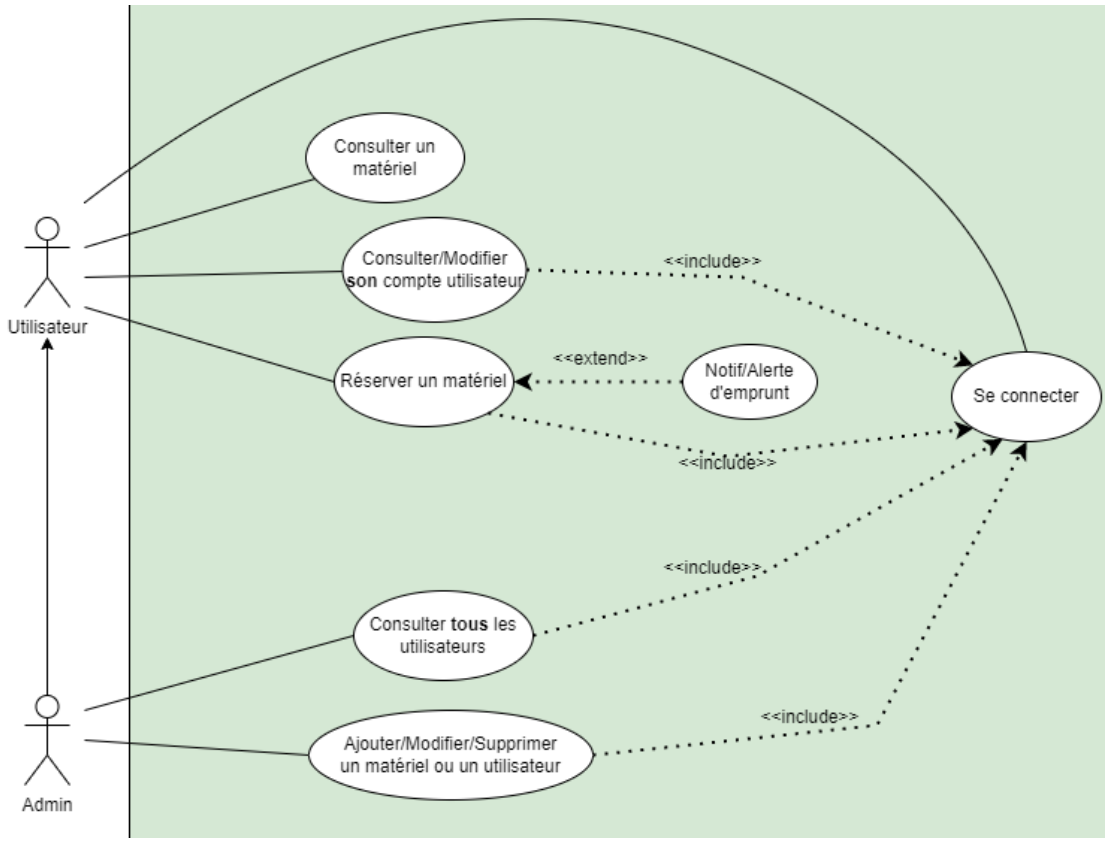
Organisation de l'équipe :

Notre équipe s'est organisée autour des tâches à accomplir et des affinités de chacun. Nous étions donc séparés en deux "sous équipes" s'occupant respectivement du front-end et du back-end. La coordination entre les membres de ces sous équipes a été principalement faite grâce aux colonnes d'avancement du tableau des spécifications. Nous nous rendons compte à posteriori qu'il aurait été utile de nommer un chef de projet, qui aurait été toujours au courant de l'avancement de chacun sur ses tâches, et qui aurait pu renforcer la coordination dans l'équipe, en attribuant à chacun une liste de tâches afin de s'assurer que tous les éléments du projets seront réalisés avant la date limite.

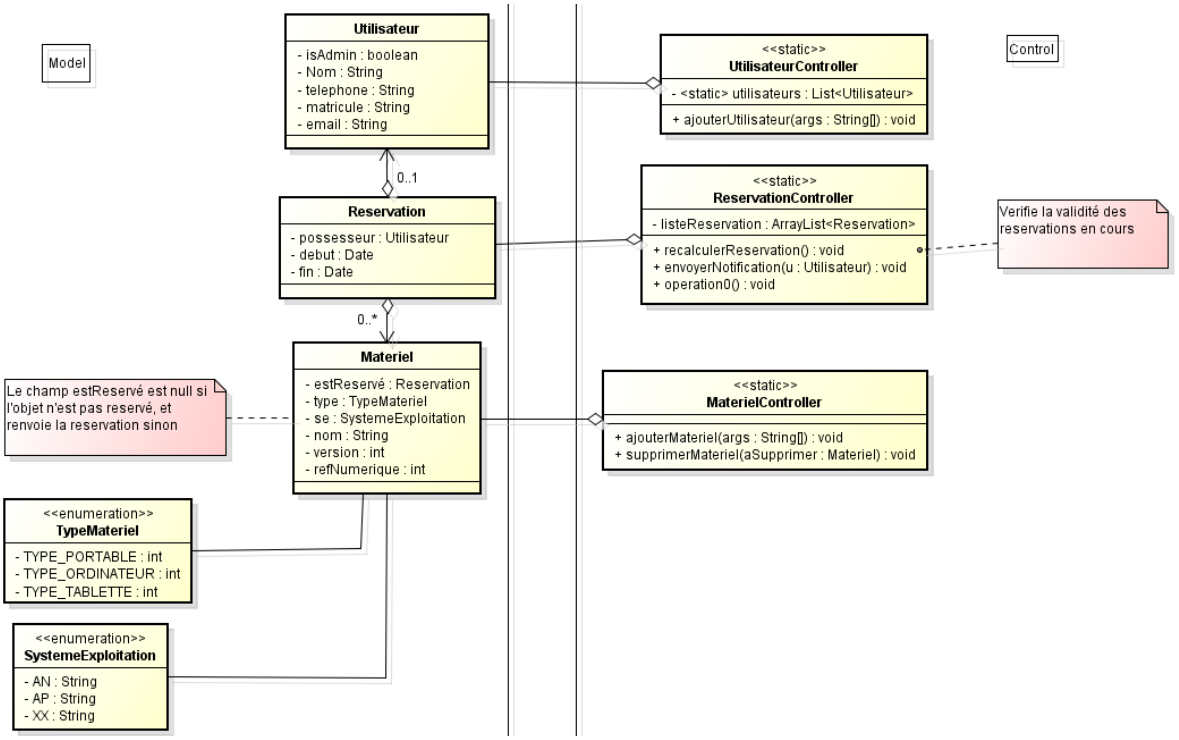
Concernant la mise en place des tests unitaires, d'intégration, ainsi que la réalisation des livrables, nous nous sommes également répartis les tâches afin de pouvoir progresser simultanément sur plusieurs points.

Conception :

Lors de la découverte du sujet, nous avons utilisé plusieurs outils pour poser le périmètre du projet. Le premier était de synthétiser les users stories dans un document texte afin de simplifier la compréhension du sujet. (cf Users_Stories.pdf).
Puis nous avons élaboré un use case pour mieux visualiser les cas d'utilisation :



Ainsi qu'un diagramme de classe :

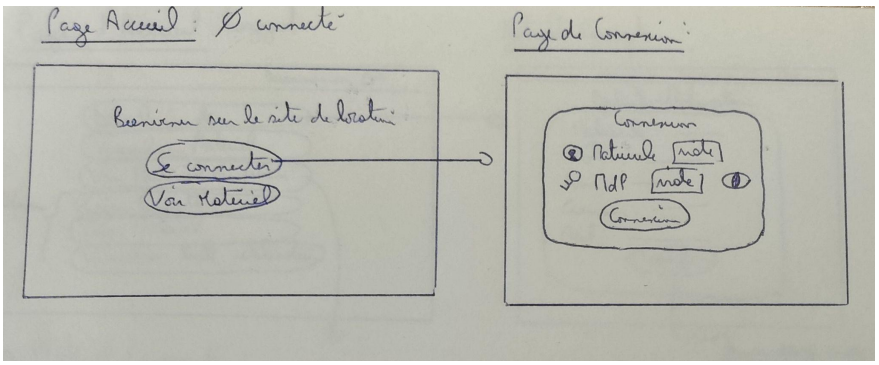


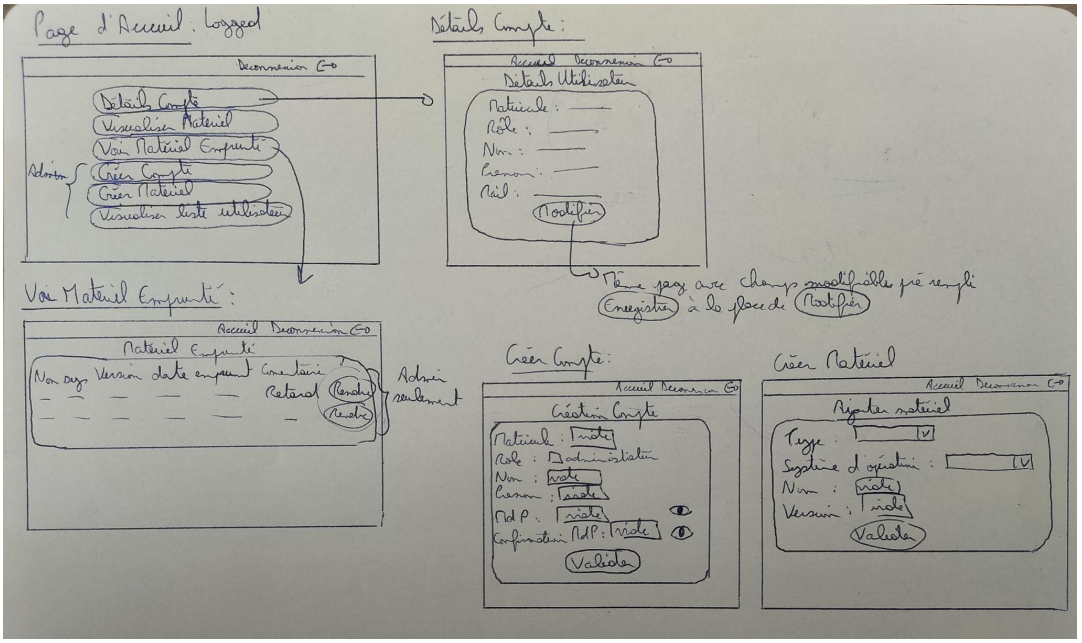
Spécifications :

Nous avons choisi de rédiger nos spécifications sous forme d'un tableau excel, dans un format assez proche de celui du cahier des recettes. Cela permet une meilleure lisibilité, une séparation nette entre les éléments logiciels, les fonctionnalités et leur appartenance à un bloc d'éléments interdépendants. De plus, cela nous a aidé à les rédiger, car nous avons pu procéder par bloc au début, puis décomposer chaque partie jusqu'à avoir un niveau de détail suffisant. Enfin, cela nous a permis de visualiser précisément l'avancement du projet grâce à une colonne indiquant si chaque élément était fait ou non, et une colonne de commentaires additionnels.

Front (html/css) :

Suite à la définition des spécifications, nous devions concevoir le visuel de notre site de location. Le cahier de spécification étant assez long, nous avons eu l'idée de créer une maquette de notre site pour éviter d'oublier un ou plusieurs éléments et rendre le développement du front plus simple. Bien que le site ne correspond pas à 100% à ces schémas, ces derniers ont beaucoup aidé lors du développement. Ci dessous, un exemple de maquette :





Recettes :

La rédaction du cahier de recettes a été facilitée par le format des spécifications, assez proche de celui-ci. Nous avons donc pris la liste des fonctionnalités, et avons extrait tous les tests qui nous ont semblé pertinents, puis rempli tous les champs nécessaires. Nous avons utilisé l'aspect dynamique des tableurs pour permettre une évolution visuelle de l'avancement des tests du logiciel.

Ceci dit, beaucoup des tests sont purement visuels, il est possible de regarder les valeurs CSS des éléments, mais ces tests sont peu pertinents dans le contexte du projet, puisque nous avons une grande liberté dans l'interprétation et la présentation de l'outil numérique.
Il est possible de faire des tests automatisés pour s'assurer de petits détails non-fonctionnels, qui sont importants dans le cas d'un contrat signé par les deux partis pour éviter des amendes sur des détails clairement prédéfinis. Ce n'est pas le cas ici.

Tests Back :

Pour les tests, et plus particulièrement les tests unitaires, nous avons opté pour JUnit 5.8.2.
La grande partie des tests se font sur les controllers, car le modèle ne contient que des getters / setters.
Les tests des fonctions des controllers commencent par vérifier si les bonnes exceptions sont bien levées lorsque l'on passe des mauvais paramètres via `Assertions.assertThrows()`, puis on test ensuite que les fonctions ont bien le comportement attendu : par exemple, si une méthode doit ajouter un objet dans une liste, on crée d'abord un objet fictif, on appelle la méthode, puis on vérifie si l'objet est dans la liste (ca parait bête et ça l'est).

Tests front:

Pour le front, nous avons utilisé Selenium 4.6.0. Après une rapide prise en main, et l'installation du driver de Chrome, nous avons pu commencer les tests.
Les tests du front sont beaucoup plus longs et compliqués à mettre en place.
Comme il est assez peu pertinent de vérifier les erreurs de saisie de l'utilisateur, nos tests portent principalement sur le bon passage des données du front au back : est-ce que, si j'ajoute un nouveau matériel sur la page, la liste du matériel du back est bien incrémentée d'un objet ?

Certains tests peuvent demander des petites astuces / ristournes / loufoqueries / zoiveries pour fonctionner : pour vérifier si le système de login fonctionne, on doit par exemple vérifier l'url de redirection (/connexion/ → /accueil/ = OK), ou compter le nombre de balises d'une certaine classe html pour s'assurer que l'affichage des utilisateurs est complet.

Parmi les choses que nous avons dû mettre en place pour faire fonctionner ces tests, on trouve notamment de lancer un serveur uniquement pour les tests. Cela a l'inconvénient de devoir le faire tourner sur un thread à part, sinon le serveur est bloquant.
Ainsi, on peut se retrouver avec des problèmes de concordance : bien souvent, après avoir fait une série d'appels sélénium, on doit faire un petit `Thread.sleep(100)` pour s'assurer que les données transitent bien à temps, et que le test passe.

Enfin, certaines pages, principalement *l'accueil*, contiennent des boutons et redirections supplémentaires si on est non-connecté, connecté, ou admin. Nous avons pu utiliser l'annotation `@Nested` de JUnit, qui permet de créer des sous-classes de tests pour la classe de test de l'accueil. Cependant, il faut tout de même se connecter / déconnecter entrer les sous-classes, et cela nécessite d'appeler des fonctions de setup/undo avec `@BeforeClass` `/@AfterClass`, mais ces annotations marchent sur des méthodes statiques, et Java ne supporte pas les méthodes statiques dans des classes nested. On doit ajouter l'annotation `@TestInstance(TestInstance.Lifecycle.PER_CLASS)` qui contourne cette restriction.

Conclusion :

Ce projet a été très formateur pour nous, car c'est la première fois que nous travaillons en "grosse" équipe. Il aura permis de mieux nous rendre compte de l'utilité des principes de qualité logicielle, dans le travail en entreprise notamment, que nous avons abordés dans le cours, ainsi que celles des documents tels que les spécifications, les diagrammes UML etc.

Même si notre gestion du projet n'a pas été parfaite, cela nous aura permis de situer les difficultés et les erreurs d'un tel exercice, et permettra de les anticiper et de les éviter lors des prochains projets. Cependant, nous sommes tout de même satisfaits du résultat. Grâce aux tests, nous pouvons avoir une certaine confiance dans le code produit, et nous avons essayé de le rendre le plus maintenable et évolutif possible.