CentraleSupélec

université PARIS-SACLAY

Rapport de projet

Génie Logiciel Orienté Objet

MyFoodora

Groupe 19

Aubin Ollivier
François Petit

9 juin 2024

1. Introduction

L'objectif de ce projet est de concevoir une application de livraison de repas sur le modèle d'*Uber Eats* ou *Deliveroo*, nommée *myFoodora*, dans le langage de programmation Java. L'application est accessible grâce à une interface en ligne de commande où les utilisateurs tels que les managers, restaurants, clients et coursiers peuvent interagir avec l'application en se connectant. Nous avons conçu l'application avec l'IDE Eclipse.

Le présent rapport a pour but de donner une vision d'ensemble du projet, le fonctionnement de l'application, deux scénarii de test, les choix effectués lors de l'implémentation, ainsi que l'organisation au sein du groupe.

2. Fonctionnement de l'application

2.1 Accessibilité et lancement de l'application

L'application fonctionne grâce à une interface en ligne de commande (CLUI). L'utilisateur interagit avec le système en entrant des commandes dans le terminal, par exemple `login <username> <password>` pour se connecter, ou `showMenuItem <restaurant-name>` pour afficher le menu d'un restaurant.

Afin d'accéder à cette interface, on exécute le programme `Main.java` qui ouvre le terminal. On peut ensuite générer un environnement ayant un manager, r restaurants, c clients et l livreurs grâce à la commande `setup r c l`. La commande `setup` permet de créer un manager et autant de restaurants, de clients et de livreurs que nous le désirons. Pour chacun de ces utilisateurs, les attributs sont initialisés de la manière suivante :

- **manager** : le manager est initialisé avec l'identifiant `"ceo"`, le mot de passe `"123"`, le prénom `"John"` et le nom de famille `"Doe"`.

- **restaurants** : les restaurants ont comme noms `"Restaurant1"`, ..., `"Restaurantn"`, comme identifiants `"rest1"`, ..., `"restn"`, comme mots de passe `"restpass1"`, ..., `"restpassn"` et des adresses choisies aléatoirement dans un carré de côté de longueur 100. Chaque restaurant se voit attribué trois entrées, trois plats et trois desserts. Nous avons effectué ce choix de sorte à ce que chaque catégorie de plat, à savoir standard, végétarien ou sans gluten, soit représentée. Les noms donnés aux entrées sont `starter_1`, `starter_2` et `starter_3`, ceux donnés aux plats `main_1`, `main_2` et `main_3`, et, de la même manière, les noms donnés aux desserts sont `dess_1`, `dess_2` et `dess_3`. Ensuite, pour avoir une carte qui soit la plus réaliste possible, nous avons attribué à ces plats des prix générés aléatoirement, avec des fourchettes de prix étant différentes pour les entrées, pour les plats et pour les desserts. Aussi, chaque restaurant se voit attribué 3 menus, nommés selon la même logique que précédemment `meal_1`, `meal_2` et `meal_3`. Chaque menu a une catégorie différente : l'un est standard, un autre végétarien et le dernier sans-gluten. Enfin, nous attribuons à tous ces menus les plats de la même catégorie.

- **clients** : les clients ont les prénoms `"Customer1"`., ... `"Customern"`, comme noms de famille `"Lastname1"`, ... `"Lastnamen"`, comme identifiants `"cust1"`, ... `"custn"`, comme mot de passes `"custpass1"`, ... `"custpassn"`, comme localisation une position générée aléatoirement, de la même façon que pour les restaurants, puis une adresse mail et un numéro de téléphone inutiles dans la suite du projet. Pour initialiser la base de données des commandes, chacun de ces clients a effectué un nombre compris entre 0 et 20

commandes à une date choisie au hasard dans les trois mois précédents la date actuelle, dans les restaurants (eux aussi choisis aléatoirement dans la liste des restaurants déjà établie). De plus, les clients peuvent choisir de recevoir ou non des notifications relatives aux offres proposées par les restaurants. Dans cette initialisation, cela est également déterminé aléatoirement, donc certains peuvent les recevoir et d'autres non.

— **livreurs** : les livreurs ont comme noms `"Courier1"`, ... `"Couriern"`, comme identifiants `"courier1"`, ... `"couriern"`, comme mots de passe `"courpass1"`, ... `"courpassn"` et de même que précédemment, une localisation aléatoire. Les livreurs sont aussi caractérisés par un état « occupé » ou « libre », cet état étant initialisé au hasard pour chacun des livreurs.

Tous les utilisateurs possèdent aussi un identifiant qui lui est propre, pour faciliter son identification. Enfin, la commande setup initialise aussi les paramètres de l'application, en fixant à 20 pourcent le pourcentage de marge, à 3 euros les frais de livraison et à 2,5 euros les frais de fonctionnement de l'application.

La liste des commandes est spécifiée à la section ...

2.2 Structure de l'application

L'application est conçue en programmation orientée objet. Le projet *myFoodora* est subdivisé en plusieurs packages, lesquels contiennent différentes classes. Voici un récapitulatif des différents packages et classes du projet *myFoodora* :

— Le package **main** contient la classe `Main` permettant de lancer l'interface en ligne de commande.
— Le package **core** contient les classes nécessaires au lancement du système *myFoodora*, telles que `MyFoodoraSystem` et `CommandLineInterface` pour l'interface en ligne de commande.
— Le package **user** spécifie les différents types d'utilisateurs. Il définit une classe abstraite `User`, implémentée par les quatre différents types d'utilisateurs de l'application : `Manager`, `Restaurant`, `Courier`, et `Customer`.
— Le package **menu** contient les différentes classes relatives aux repas, telles que `Dish`, `Meal`, et `Menu`.
— Le package **orders** contient la classe `order` relative aux commandes.
— Le package **fidelitycard** contient les classes relatives aux programmes de fidélité. Il définit une interface `FidelityCard` implémentée par les trois programmes de fidélité `BasicFidelityCard`, `LotteryFidelityCard`, et `PointFidelityCard`.
— Le package **utils** contient la classe `Location` définissant l'adresse d'un utilisateur.
— Enfin, le package **tests** contient tous les tests JUnit importants du projet.

2.3 Explication des différentes commandes de base

Nous rappelons ici les commandes spécifiées dans le sujet. (modifiées ?)

— `login <username> <password>` : permet à un utilisateur de se connecter (un manager avec pour identifiant `"ceo"` et mot de passe `"123"` existe par défaut)

— `logout <>` : permet à l'utilisateur connecté de se déconnecter

— `registerRestaurant <name> <address> <username> <password>` : permet à un manager connecté d'ajouter un restaurant avec le nom, l'adresse (des coordonnées bidimensionnelles), l'identifiant et le mot de passe spécifiés

— `registerCustomer <firstName> <lastName> <username> <address> <password>` : permet à un manager connecté d'inscrire un client à l'application

— `registerCourier <firstName> <lastName> <username> <position> <password>` : permet à un manager connecté d'ajouter un livreur à l'application

— `addDishRestaurantMenu <dishName> <dishCategory> <foodCategory> <unitPrice>` : permet à un restaurant connecté d'ajouter un plat avec le nom, la catégorie (entrée, plat principal, dessert), le type d'aliment (standard, végétarien, sans-gluten) et le prix spécifiés au menu d'un restaurant

— `createMeal <mealName> <foodCategory>` : permet à un restaurant connecté de créer un repas avec un nom spécifié et une catégorie

— `addDish2Meal <dishName> <mealName>` : permet à un restaurant connecté d'ajouter un plat à un repas

— `showMeal <mealName>` : permet à un restaurant connecté de montrer les plats d'un repas

— `saveMeal <mealName>` : permet à un restaurant connecté d'enregistrer un repas avec le nom spécifié

— `setSpecialOffer <mealName>` : permet à un restaurant connecté d'ajouter un repas en tant qu'offre spéciale de la semaine

— `removeFromSpecialOffer <mealName>` : permet à un restaurant connecté de retirer son offre spéciale de la semaine

— `createOrder <restaurantName> <orderName>` : permet à un client connecté de créer une commande dans un restaurant

— `addItem2Order <orderName> <itemName>` : permet à un client connecté d'ajouter un article (soit un élément de menu, soit un repas) à une commande existante

— `endOrder <orderName> <date>` : permet à un client connecté de finaliser une commande à une date donnée et de la régler

— `onDuty <username>` : permet à un livreur connecté de définir son état comme étant en service

— `offDuty <username>` : permet à un livreur connecté de définir son état comme étant en congé

— `findDeliverer <orderName>` : permet à un restaurant connecté d'attribuer une commande à un livreur en appliquant la politique de livraison actuelle

— `setDeliveryPolicy <PolicyName>` : permet à un manager connecté de définir la politique de livraison

— `setProfitPolicy <ProfitPolicyName>` : permet à un manager connecté de définir la politique de profit

— `associateCard <userName> <cardType>` : permet à un manager connecté d'associer une carte de fidélité à un client

— `showCourierDeliveries <>` : permet à un manager connecté d'afficher la liste des livreurs triés par ordre décroissant par rapport au nombre de livraisons effectuées

— `showRestaurantTop <>` : permet à un manager connecté d'afficher la liste des restaurants triés par ordre décroissant par rapport au nombre de commandes livrées

— `showCustomers <>` : permet à un manager connecté d'afficher la liste des clients

— `showMenuItem <restaurant-name>` : permet à utilisateur connecté d'afficher le menu d'un restaurant

— `showTotalProfit <>` : permet à un manager connecté d'afficher le profit total de l'application depuis le début

— `showTotalProfit <startDate> <endDate>` : permet à un manager connecté d'afficher le profit total de l'application sur une période spécifiée

— `runtest <testScenario-file>` : permet à un utilisateur de l'interface en ligne de commande d'exécuter la liste des commandes contenues dans le fichier de scénario de test passé en argument

— `help <>` : permet à un utilisateur de l'interface en ligne de commande (pas besoin de se connecter) d'afficher la liste des commandes disponibles avec une indication de leur syntaxe

2.4 Explication des commandes ajoutées

En plus des commandes établies dans le sujet, nous avons implémenté de nouvelles commandes, de sorte à augmenter le nombre de fonctionnalités de l'application. Ces fonctionnalités étant néanmoins présentées la majorité du temps dans le sujet. Parmi celles-ci, nous trouvons les commandes :

— `showProfile` : cette commande permet à un utilisateur, quel que soit son rôle dans l'application (restaurant, client, livreur ou manager) d'avoir un aperçu de l'ensemble de ses attributs (ceux détaillés dans la liste en partie 2.1). Ainsi, un utilisateur peut par exemple voir si ses notifications sont activées ou non, de quelle carte de fidélité il dispose, le nombre de points dans le cas d'une carte de fidélité à points ou encore le nombre de commandes passées. Un restaurant peut par exemple voir combien de commandes ont été reçues. Enfin, un livreur peut voir s'il s'est déclaré occupé ou libre, combien de commandes ont été livrées par ses soins...

— `notifOn` et `notifOff` : cette commande permet, comme son nom l'indique, à un client

d'activer ou non les notifications qui lui permettront d'être averti des potentielles offres

— `showNotifs` : cette commande permet à un utilisateur et un livreur d'accéder à l'ensemble des notifications reçues, dans leur ordre d'apparition. Pour les clients, cela concerne comme expliqué ci-dessus les offres promotionnelles, pour un livreur, les notifications servent à recevoir l'ensemble des informations relatives à une livraison lui ayant été attribuée.

— `histoOrders` : cette commande permet à un client ou un restaurant d'avoir accès à l'historique des commandes passées/reçues. Ces commandes sont classées par date.

— `showMenu` : cette commande permet à un restaurant d'avoir accès à l'ensemble de sa carte.

— `setServiceFee <serviceFee>` : cette commande permet au manager de modifier la valeur des frais de fonctionnement de l'application par commande. Cette commande reçoit en argument un nombre, à considérer comme une valeur en euros.

— `setMarkupPerc <markupPerc>` : cette commande permet comme précédemment au manager de modifier le pourcentage de marge. la commande prend en argument une valeur comprise entre 0 et 1.

— `setDeliveryCost <deliveryCost>` : cette commande permet au manager de modifier les frais de livraison d'une commande. Cette commande reçoit en argument un nombre à considérer comme une valeur en euros.

— `showOrders` : cette commande permet au manager d'accéder à l'ensemble des commandes ayant été passées sur le site depuis sa création, triées par date.

— `defaultFactorDiscount <defaultFactor>` : cette commande permet à un restaurant de modifier le pourcentage de réduction accordé lorsque les plats sont arrangés dans un menu.

— `specialFactorDiscount <specialFactor>` : cette commande permet de même à un restaurant de modifier le pourcentage de réduction qui s'applique au menu qualifié de menu de la semaine.

— `lastMonthIncome` : cette commande permet au manager de visualiser les revenus obtenus sur le dernier mois glissant. La date de référence étant celle du jour même.

— `setup <numRestaurant> <numCustomers> <numCouriers>` : cette commande permet une initialisation du système, son fonctionnement est détaillé en section 2.1.

3. Scénarios de test

Nous avons élaboré deux scénarios de test, un premier avec une base de données vierge, et un second avec une base de donnée remplie grâce à la fonction `setup`.

Afin de lancer le premier scénario (respectivement le second scénario), il suffit d'exécuter le fichier `Main.java` et de rentrer dans le terminal

```
runtest testScenario1.txt.
```

(respectivement `runtest testScenario2.txt.`)

Nous n'avons pas réussi à configurer un fichier .ini et en sommes désolés.

3.1 Avec une base de données vierge

Nous allons premièrement nous intéresser à un scénario où la base de données est vierge. Nous allons ajouter un restaurant, un client, ainsi que deux livreurs.

Tout d'abord, nous allons configurer une base de données vierge avec la commande `setup 0 0 0` et nous connecter en tant que CEO. Le manager y ajoute le restaurant `Tour_d'Argent`, un utilisateur, ainsi que deux coursiers, `Max Verstappen` et `Lewis Hamilton`. De plus, le manager définit la politique de livraison en tant que « la livraison la plus rapide ». Notons que le livreur le plus proche du client est Max Verstappen.

Commençons par ajouter des plats et menus à la carte de la Tour d'Argent. Nous y ajoutons deux menus :

- Un premier menu *Renommées* "standard" composé d'une entrée (Foie gras des trois empereurs), un plat (caneton Frédéric Delair) et un dessert (crêpe Mademoiselle)
- Un second menu *Temps et saisons* végétarien également composé d'une entrée (Asperges vertes et blanches), un plat (gnocchis au comté) et un dessert (soufflé Valtesse)

Nous enregistrons ensuite les menus et les affichons.

```
> showMeal Renommées
Renommées (standard):
Starter: Foie_gras_des_trois_empereurs (starter, standard) - 135,00
Main Dish: Caneton_Frédéric_Delair (main, standard) - 185,00
Dessert: Crêpe_Mademoiselle (dessert, standard) - 58,00
Price: 359.1

> showMeal Temps_et_saisons
Temps_et_saisons (vegetarian):
Starter: Asperges_vertes_et_blanches (starter, vegetarian) - 86,00
Main Dish: Gnocchis_au_comté (main, vegetarian) - 116,00
Dessert: Soufflé_Valtesse (dessert, vegetarian) - 42,00
Price: 231.8
```

On peut ensuite essayer de modifier le taux de réduction par défaut à 4 % et le taux de réduction de l'offre de la semaine à 20 %. On définit ensuite le menu *Temps et saisons* comme offre de la semaine. On peut voir que les prix ont bien été modifiés :

```
> showMeal Renommées
Renommées (standard):
Starter: Foie_gras_des_trois_empereurs (starter, standard) - 135,00
Main Dish: Caneton_Frédéric_Delair (main, standard) - 185,00
Dessert: Crêpe_Mademoiselle (dessert, standard) - 58,00
Price: 362.88

> showMeal Temps_et_saisons
Temps_et_saisons (vegetarian):
```

```
Starter: Asperges_vertes_et_blanches (starter, vegetarian) - 86,00
Main Dish: Gnocchis_au_comté (main, vegetarian) - 116,00
Dessert: Soufflé_Valtesse (dessert, vegetarian) - 42,00
Price: 195.2
This is the meal of the week!
```

On observe que les prix ont bien changé. Nous pouvons ensuite nous connecter en tant qu'utilisateur et faire une commande au restaurant. Nous commandons un menu et un plat, puis affichons la commande.

```
    > endOrder commande 2024-06-09
Order ID: commande
Restaurant: Tour_d'Argent
Date: 2024-06-09
Dishes:
Foie_gras_des_trois_empereurs (starter, standard) - 135,00
Meals:
Temps_et_saisons (vegetarian):
Starter: Asperges_vertes_et_blanches (starter, vegetarian) - 86,00
Main Dish: Gnocchis_au_comté (main, vegetarian) - 116,00
Dessert: Soufflé_Valtesse (dessert, vegetarian) - 42,00
Price: 195.2
This is the meal of the week!

Total Price: 330.2

You benefit from BasicFidelityCard
Final Price after discount: 330,20
Order finalized on: 2024-06-09
Courier mv allocated to deliver the order.
Order finalized successfully.
```

On observe que le livreur alloué à la commande est bien le livreur le plus proche, Max Verstappen. On peut ensuite, en se connectant en tant que CEO, afficher des statistiques telles que la liste des restaurants les plus sollicités, les utilisateurs, les livraisons par coursiers ainsi que le profit total. Nous pouvons également changer le mode de fidélité de l'utilisateur :

```
> showRestaurantTop
Tour_d'Argent - Orders: 1
> showCustomers
aubin - Name: Aubin Ollivier
> showCourierDeliveries
mv - Deliveries: 1
lh - Deliveries: 0
> showTotalProfit
Total Profit: 65.54
> associateCard aubin point
Fidelity card associated successfully.
```

Enfin, on peut commander une deuxième fois, cette fois-ci en plaçant le livreur Max Verstappen "off-duty". La commande est livrée par le deuxième livreur, Lewis Hamilton, car bien

qu'il soit plus loin, c'est le seul qui est en service (plus exactement le plus proche qui soit en service).

```
Courier lh allocated to deliver the order.
```

3.2 Avec une base de données remplie

Dans cette partie, nous allons nous intéresser à un use-case se basant sur une base de données déjà bien remplies. En effet, le fichier testScenario2.txt débute par la commande `setup 25 50 15`, signifiant qu'il y a 25 restaurants, 50 clients et 15 livreurs. Chacun des clients réalise entre 2 et 5 commandes dans les 3 mois précédents la date d'initialisation. Pour chaque commande, des menus et plats sont choisis au hasard dans la liste des restaurants proposés. Puis un livreur est assigné à la commande, incrémentant leur compteur de livraison. La politique de livraison est initialisée sur "fastest", donc pour chacune de ces livraisons, le livreur le plus proche du restaurant est choisi.

La première partie de ce use-case concerne le manager. Nous commençons par montrer son profil, puis il sélectionne la politique de livraison "fair-occupation". Les livreurs ayant effectué le moins de livraisons sont maintenant priorisés vis-à-vis de la distance. Le manager accède ensuite à la liste de l'ensemble des commandes depuis la création de l'application, puis il observe la carte du restaurant n°7, montrant son contenu et donnant un exemple de carte de restaurant générée. Ensuite, le manager exploite les données récoltées, en regardant successivement le chiffre d'affaire depuis la première commande, le chiffre d'affaire sur le dernier mois glissant, le profit depuis le début, puis le profit entre deux dates choisies par le manager. Le manager modifie ensuite les frais de fonctionnement de l'application, les faisant passe de 2.5 euros à 3.5. Puis il change la politique de profit en la passant sur deliverycost. Cette commande peut par ailleurs prendre un second argument : un profit cible sur un mois. Cependant, ne pouvant connaître le nombre précis de commandes contenues dans la liste "Orders" suite à l'initialisation, nous ne sommes pas en mesure d'estimer un profit cible. Nous précisons tout de même que la commande doit renvoyer un nouveau prix pour les frais de livraison, ce n'est ici pas le cas, mais cela provoque notre incompréhension vis-à-vis du code que nous pensions correctement implémenté. Enfin, il prend un client aléatoire (Customer35) puis lui attribue la carte de fidélité à points. Ce client servira par la suite.

Nous nous connectons justement au profil de ce clientdans le but d'activer les notifications, utilisées plus tard dans la démonstration. Nous affichons son profil. Cela permet éventuellement (avec une chance sur deux) de constater que les notifications sont déjà activées ou non. En effet, à la création du client, ce paramètre est choisie aléatoirement. Dans le doute, la commande suivante permet d'activer les notifications. Nous affichons ensuite l'historique de ses commandes.

Par la suite, nous nous connectons à un restaurant (Restaurant17), nous exposons sa carte, ses menus meal_1 et meal_2, nous le faisons modifier le discount accordé sur les menus classiques (passant d'une valeur par défaut de 5 pourcent à une valeur de 7 pourcent) puis le discount accordé au menu de la semaine (passant quant à lui de 10 à 20 pourcent). Nous réaffichons ensuite le meal_1, et constatons que le prix à bien évolué. Nous déclarons ensuite le meal_2 comme étant le menu de la semaine, puis de même nous l'affichons. Encore une fois, nous observons que le prix à évolué en conséquence. De plus, dès qu'un restaurant déclare un menu de la semaine, une notification est envoyé à tous les utilisateurs le permettant. Pour compléter cet exemple, nous nous connectons sur le profil d'un autre restaurant, nous lui faisons déclarer un menu de la semaine aussi.

Puis nous passons sur le profil d'un livreur, nous exposons ses informations. De même que pour les notifications pour les clients, à l'initialisation, les livreurs sont déclarés comme étant en service ou en pause de manière aléatoire. Encore une fois, nous pouvons donc éventuellement voir que le livreur est déclaré comme étant prêt à effectuer des livraisons. Pour ce livreur, nous exposons aussi la liste des notifications reçues comportant toutes les informations nécessaires au traitement d'une commande dont la livraison lui est confiée. Si cette liste est vide et si vous désirez observer le contenu d'une telle notification, nous vous invitons à relancer le fichier pour que le livreur considéré au hasard soit initialement déclaré comme étant prêt à effectuer des livraisons.

Nous revenons ensuite sur le profil d'un client (Customer35) et nous lui faisons exposer la liste des notifications reçues. Nous constatons que comme attendu, deux notifications relatives à des déclaration de menus de la semaine par des restaurants sont présentes. Il n'est pas nécessaire pour lui de relancer le fichier, la liste ne pouvant être vide puisque précédemment, nous avions activé les notifications de cet utilisateur. Nous lui faisons passer une commande dans le restaurant Restaurant17, puis le client finalise sa commande à la date du 20 juin 2024. Nous rappelons que cet utilisateur bénéficie de la carte de fidélité à points. Dans ce cadre, la commande affiche le total de points sur sa carte. Comme cette carte vient de lui être attribué, le total de point ne s'incrémente que sur cette commande.

Nous nous reconnectons ensuite au manager, qui montre la liste des restaurants, classés dans un ordre décroissant du nombre de commandes reçues depuis le création de l'application. De même, nous exposons le nombre de livraisons réalisées par chacun des livreurs dans un ordre décroissant des livraisons faites.

Pour tester la politique de de livraison sélectionnée en début de processus par le manager, nous nous connectons au restaurant ayant reçu la commande du Customer35 et nous lui faisons assigner manuellement un livreur. Nous voyons que le livreur choisi est bien celui ayant le moins de livraisons, et étant déclaré comme disponible. Néanmoins, cette commande ne sert que d'exemple puisque dès que la commande est finalisée par le client, un livreur est attribué automatiquement. Si vous affichez à nouveau la liste des restaurants et leur nombre de commandes, ainsi que la liste des livreurs et leur nombre de livraisons, nous constatons que le total de commande livrée et supérieur d'une unité au total de commandes passées dans les restaurants. Cela est normal compte tenu de la remarque précédente.

4. Choix effectués lors du développement

Lors du développement, nous avons essayé au maximum de respecter le principe "open-close". Ainsi, par exemple, nous avons crée une classe abstraite `User` pour les utilisateurs implémentée par les différents types d'utilisateurs (managers, clients, restaurants et livreurs). Cela permet de pouvoir étendre facilement à d'autres types d'utilisateurs tout en pouvant créer un utilisateur générique. Également, nous avons crée une interface `FidelityCard` implémentée par les trois types de cartes de fidélité. Ainsi, il est très aisé de pouvoir ajouter de nouvelles cartes de fidélité.

Par ailleurs, nous avons décidé d'organiser la structure du code en différents packages correspondant à plusieurs sections de l'application. Cela permet un gain d'agilité lors du développement, ainsi qu'une meilleure lisibilité du code.

5. Organisation au sein du groupe

Nous avons en grande majorité fait le projet ensemble, nous nous sommes retrouvés à plusieurs reprises dans des salles afin de réfléchir ensemble sur le projet. Une fois la structure globale de l'application construite, nous nous sommes répartis les tâches et les implémentations équitablement et nous retrouvions au moins une fois par semaine pour coder ensemble et pour échanger. La conception des scénarios de test et l'écriture du rapport ont également bien été réparties entre nous deux. Ce fut un réel plaisir de travailler en binôme.