Restaurant E24

Rapport Général

Itération 3

1ère Année, Département Sciences du Numérique

Younes Saoudi
Hamza Mouddene
Issam Habibi
Mehdi Wissad
Hatim Meskine
Chaimaa Lotfi

2019 - 2020

Contents

Rapport

1.1 Rappel du sujet

Nous devons réaliser une application de gestion de restaurant. Le modèle à mettre en oeuvre contiendra deux interfaces :

- Interface client : visualisation du menu, prises de commandes, disponibilités des plats, réservation, personnalisation des menus; etc.
- Interface administrateur : gestion de l'état du restaurant, gestion du menu et des commandes, interaction avec les clients; etc.

L'application sera pourvue d'une interface graphique et accessible à la fois par les clients et le personnel du restaurant.

1.2 Les principales fonctionnalités

- L'accès aux fonctionnalités de l'application se fait par un système de Log In obligatoire pour les employés et optionnel pour les clients. La création d'un compte peut se faire au sein de l'application même et un email de vérification sera envoyé.
- Les employés peuvent visualiser l'état de leur stock et le changer , ils peu- vent aussi changer le contenu du menu (prix , plats , offres exceptionnelles, etc...).
- Les employés ont accès à un espace pour consulter l'état financier du restaurant et aussi pour le mettre à jour (les gains en cours , relevés de dépôts , comparaison des gains mensuels, etc...).
- Les employés ont accès à un "espace client" où ils peuvent visualiser les commandes des clients , mettre à jour l'état des commandes et communiquer avec les clients.
- Les clients peuvent accéder au menu et commander leurs plats à partir du menu en précisant l'état de la commande(sur place, emportée). Le prix total de la commande doit être visible et dynamique.
- Les clients qui ont créé un compte auront accès aux plusieurs autres fonctionnalités: un historique d'achats , un système de points de fidélité avec possibilité de parrainage , un outil de communication avec le personnel du restaurant, etc...
- La bonne utilisation de la programmation par contrat en utilisant JML et de la programmation défensive,
- Les informations des utilisateurs et leurs progressions seront stockées dans un fichier et sauvées à la fermeture

1.3 Le découpage de l'application en sous-systèmes

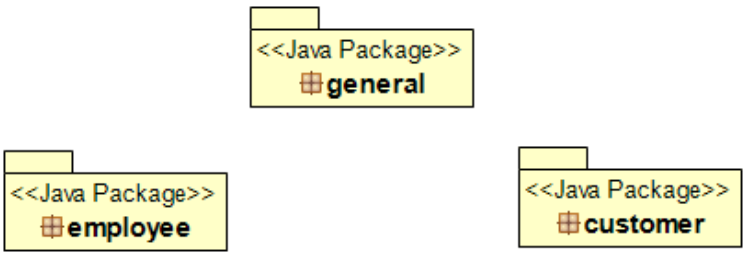

Figure 1.1: Packetage de l'Application

L'application est découpée en trois *packages*:

- `general`: Contient les classes `Sign Up, Sign In, Security, Home, Account Details` ainsi que la classe des utilisateurs `User`.
- `customer`: Contient tout ce qui est relatif aux clients comme leur interface, les menus, l'historique d'achat; etc.
- `employee`: Contient tout ce qui est relatif aux employés comme la création de compte d'un nouvel employé, la gestion des commandes et du stock ainsi que l'état financier du restaurant.

1.4 Quelques diagrammes de classe

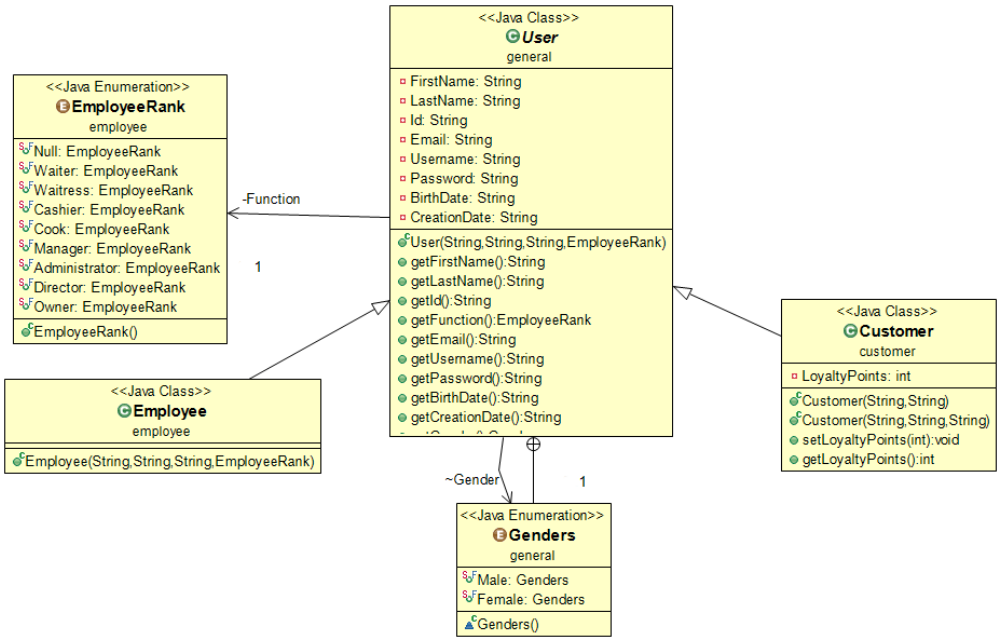

Figure 1.2: Les Utilisateurs

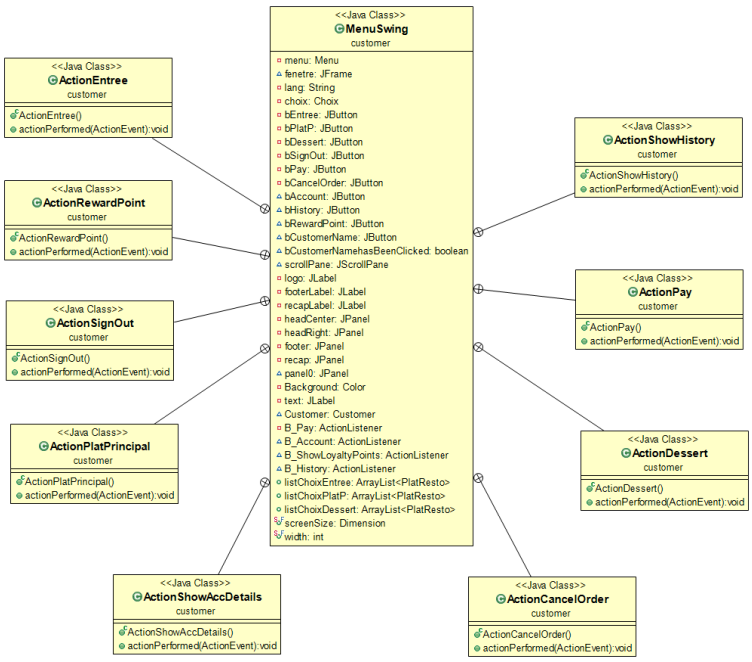

Figure 1.3: L'Interface Graphique des Clients

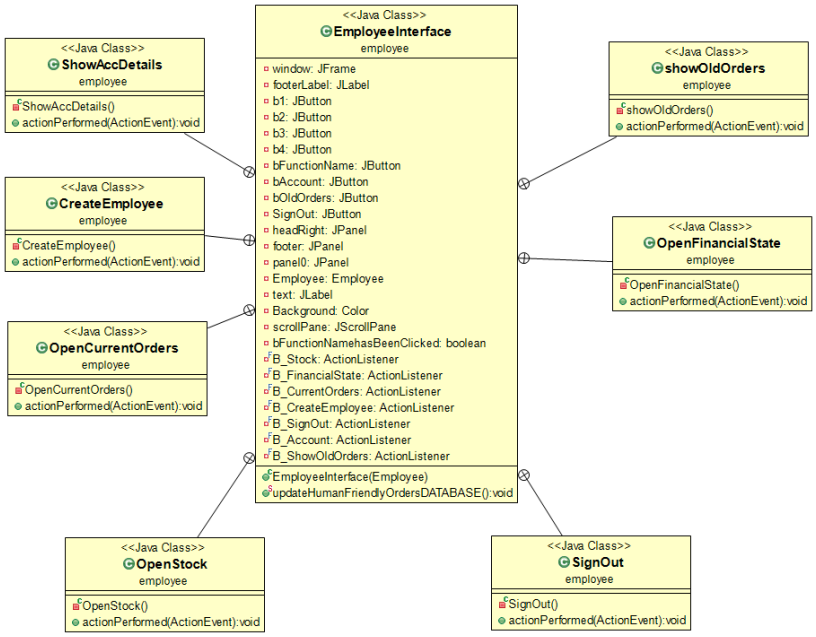

Figure 1.4: L' Interface Graphique des Employés

Conception

2.1 Les principaux choix de conception et réalisation

2.1.0 Code en Anglais:

Comme vous l'avez certainement remarqué, la quasi-totalité du code de l'application est en anglais. Ceci est une décision que nous avons prise pendant la 1ère itération. La raison pour laquelle nous avons décidé de coder en anglais au lieu de suivre les consignes vues normalement en TD/TP est le fait que la majorité des langages de programmation utilisés aujourd'hui dans l'industrie sont en anglais et contiennent des mots réservés en anglais pour que le code se lise de façon naturelle, intuitive et plus ou moins comme la langue anglaise.

Choisir dans son code des identifiants qui ne sont pas composés de mots anglais va complètement contre ce design universel de programmation en anglais et rend le code difficile à lire. Sans oublier que publier du code contenant des identifiants ou commentaires en une autre langue que l'anglais sur un forum comme stackoverflow, serait choisir de présenter à la majorité des lecteurs un code qui, pour eux, n'a aucun sens, est illisible et de basse qualité; cela reduit ainsi la qualité de leurs réponses.

Finalement, la raison la plus importante derrière ce choix est le fait que pendant cette année, plusieurs boîtes où nous voulions faire un stage en fin d'année demandait un projet de programmation que nous avions effectués durant nos études, exigeant qu'il soit commenté en anglais. Ceci est d'autant plus vrai quand nous essayons de postuler à l'étranger. Nous espérons ainsi avoir été assez clairs et transparents et vous remercions pour votre compréhension.

2.1.1 Internationalisation

L'internationalisation d'une application consiste à construire l'application avec une architecture qui permet de l'adapter facilement à différentes langues ou pays. Dans notre cas, nous avons choisi de donner aux clients la liberté de choisir entre les deux langues qui sont disponibles à savoir le français et l'anglais. Ce choix de la langue ne concerne que l'interface dédiée au client. Tout ce qui est en rapport avec l'interface de l'employé est écrit en anglais.
En ce qui concerne l'implémentation, nous avons procédé à travers le format de données JSON. Deux base de données `langueEn.json` et `langueFr.json` ont été ainsi créées et contiennent tous les messages qui s'afficheront sur les différentes pages auxquelles le client peut accéder.

2.1.2 Log In/Sign Up

Tout compte créé est enregistré dans une base de données JSON `users.json`. Cette base de donnée contient, pour chaque individu créé, les attributs suivants:

- `Function:` La fonction de l'utilisateur (`Null` si client).
- `First name`
- `Last name`

- `Gender`
- `Date of Birth`
- `E-Mail:` C'est l'e-mail d'inscription auquel un message de confirmation est envoyé pour chaque Sign Up
- `Username:` Il est choisi par les clients mais généré automatiquement pour les employés ($1^{ère}$ lettre du prénom + 6 premières lettres du nom + occurrences). Il est unique.
- `Password:` Il est crypté avant d'être enregistré dans la base de données.
- `Created:` La date de création du compte
- `UserID:` Un ID utilisateur unique généré automatiquement à la création.

ATTENTION! Contrairement aux clients qui créent leur compte eux-même, seul un employé de rang supérieur à `Manager` peut créer un compte pour un nouvel employé (sinon n'importe qui pourra créer un compte `Employee`).

2.1.3 Cryptage des mots-de-passe

Afin d'augmenter la sécurité, les mots-de-passe des utilisateurs ne sont pas stockés tels quels dans la base `orders.json` mais ils sont d'abord cryptés grâce à classe `Security` ainsi que les bibliothèques `java.util.Base64` et `javax.crypto`. Cette classe prend le mot de passe et, en utilisant un *salt* généré aléatoirement, le code en base 64 et le crypte puis le sauvegarde dans la base de données. Quand un utilisateur veut se connecter, on compare la version décryptée du mot de passe sauvegardé avec ce que l'utilisateur a saisi pour vérifier si les informations sont correctes et le rediriger vers son compte le cas échéant.

2.1.4 Interface des Clients

Le Menu

Le menu permet les clients de faire leurs commandes et les payer à partir de divers choix . Sa version graphique est représentée par la classe `MenuSwing` qui se base sur la bibliothèque `javax.swing` , qui implémente l'interface `Observer` pour assurer la dynamicité et l'interactivité entre les différents parties de l'application et qui possède 4 attributs :

- `menu:` parvenant de la classe Menu qui modélise un menu abstrait.
- `choix:` parvenant de la classe Choix qui modélise la commande du client
- `customer:` parvenant de la classe Customer
- `lg:` Un string caractérisant la langue du menu.

La classe Menu permet la construction du menu à partir des `ArrayList` des plats. chacune de ces liste caractérise un sous menu des 3 catégories suivantes : les entrées , les plats principaux et les desserts. Cette construction se fait d'une manière statique et prédéfinie et offre plusieurs fonctionnalités : les `getters` pour récupérer les listes des plats de chaque catégorie , les `setters` pour les changer et l'ajout d'un plat à chacune des catégories.

Quant aux plats , ils parviennet d'une classe `PlatResto` qui modélisent les plats d'une manière générale , cette classe est l'implémentation de l'interface `Plat`, elle permet de construire un plat à partir des 4 attributs suivants et de manipuler ces attributs :

- `type:` Une énumération = ENTREE, PLATPRINCIPAL, DESSERT

- `nomPlat:` Le nom du plat
- `prixPlat:` le prix du plat
- `disponibilité:` Un booléan caractérisant la disponibilité d'un plat

Chaque catégorie de plats possède une version swing `EntreeSWing, PlatPrincipauxSwing, DessertSwing` , le but est de faire "pop up" un sous-menu des plats de chaque catégorie une fois le client appui sur le bouton correspondant dans la version graphique du menu .

Pour la classe `Choix` , le but est de conserver la commande du client dans un modèle qui permet de la manipuler, la représenter graphiquement, faire des calculs..etc. Elle hérite de la classe `Observable` et elle se base sur un seul attrbut:

- `map:` Une map de la forme `Map<String, ArrayList<PlatResto>` qui associe à chaque `Clé` des 3 catégorie possibles: Entrées, plat principaux et desserts une liste des plats choisits par le client dans cette catégorie.

Ensuite, cette classe offre plusieurs fonctionnalités:

- `getPrixTotal:` Récupérer le prix totale de la commande du client .
- `toStringCommande:` Une représentation string de la commande du client dans la quelle figurent les noms des plats choisits et leurs quantités.
- `AjouterEntree/AjouterPlatPrincipal/AjouterDessert:` Ajoute un plat à son catégorie, et notifie les observateurs de ce changement, on envisage ainsi de construire la commande du client à partir de ces méthodes d'une façon dynamique , si le client appui sur le bouton choisir dans le sous-menu d'une catégorie des plats, son choix est bien conservé .

Payment:

Le paiement des commandes les enregistre immédiatement dans la base de données `orders.json` avec toutes les informations nécessaires telles que le contenu, le prix, le nom du client et son UUID s'il est connecté, le numéro de la table, la date de création, le statut de la commande (qui est initialement configuré sur "In Progress") ainsi qu'un UUID associé à la commande pour faire la différence entre une commande et une autre.

Loyality Points:

En ce qui concerne les points de fidélité, chaque client se voit attribué lors de son inscription un compteur de points qui est initialisé à 0. On trouve donc dans la base de données users.json la donnée relative aux "RewardPoint". Pour pouvoir récupérer les points de fidélité , on fait appel au constructeur de la classe RewardPoint qui prend comme paramètre l'ID du client. On peut également faire appel à d'autres méthodes définies dans cette classe comme le fait d'ajouter des points ou de savoir si le client peut bénéficier d'une réduction.

2.1.5 Interface des Employés

- `Current Orders:` Les commandes en cours de préparation (i.e., qui ne sont pas marquées comme `Done`) sont affichées sous la forme d'une table JTable équippé de filtres `RowFilter` configurés en `and` pour qu'ils se complémentarisent. Les commandes sont extraites d'une base de données JSON `orders.json` qui contient, pour chaque commande, les attributes suivants:
  - `Status:` L'état de la commande. Il ne peut être que `Done/ In Progress/ Delayed/ Need Assistance.`

- `Order:` Le contenu de la commande
- `Payment:` La méthode de paiement `Credit Card/ Cash`
- `TableNm:` Le numéro de table (de 1 à 50) ou éventuellement `TAKEOUT` pour signifier que la commande est emportée.
- `Price:` Le prix à payer.
- `Customer:` Le nom du client qui a commandé (`NULL` s'il n'est pas connecté)
- `UserID:` L'ID unique du client
- `OrderNm:` L'ID unique de la commande (sert à différencier les commandes)
- `Created:` La date de création de la commande
- `Edits:` Toutes les modifications faites aux commandes par les employés, le nom de l'employé correspondant, la date de la modification et le contenu de cette dernière.

La table JTable est aussi munie d'un `Listener` pour enregister toutes les modifications faites aux commandes dans la base de données. `users.json`
L'import des commandes se fait par la méthode `String[][] importOrders(boolean OnlyOrderID)`. Si `OnlyOrderID` est `false`, la méthode retourne la matrice contenant dans chaque ligne une commande pas encore finie et chaque colonne l'un des attributs de ces commandes dans la base JSON. Sinon, elle retourne un vecteur contenant l'ID unique de chaque commande.

Ce vecteur est utilisé pour l'édition des commandes: Le `Listener` fait appel à la méthode `void editOrder(Employee Employee, String OrderID, int column, String NewEdit)` qui applique les modifications qu'a effectué l'employé `Employee` sur la colonne `column` de la ligne associée à la commande `OrderID` en remplaçant la valeur ancienne par la valeur nouvelle `NewEdit`.
Cette méthode marque alors les modifications faites à cette commande par cet employé, le nom de l'employé correspondant, la date de la modification et le contenu de cette dernière.
Il faut aussi noter que toutes les modifications dans la table JTable sont faites au travers d'un `ComboBox` pour contrôler les saisies des employés et harmoniser le contenu de la base de données JSON.

- `Old Orders:` Comme pour `Current Orders`, les commandes anciennes (i.e., celles marquées comme `Done`), sont affichées dans une JTable et montrent les modifications qu'elles ont subites et par quel employé. Un employé peut alors relancer cette commande en la marquant comme `In Progress` ou modifier une autre colonne s'il le veut; modification qui sera aussi bien sûr enregistré dans la base de données (Cela permet aux superviseurs de savoir qui a effectué quelle modification sur quelle commande et quand).

- `User:` User est une classe dont héritent `Customer` et `Employee`.

- `Financial State:`

  - L'état financier du restaurant, où on gère les revenus et les dépenses du restaurant. Les employés ont accès à un espace pour consulter l'état financier du restaurant et aussi pour le mettre à jour (les revenus en cours, relevés de dépôts, comparaison des gains mensuels..). On a réalisé les trois fonctionnalités suivantes : relevé de dépôt, l'historique, comparaison des gains mensuels.

  - L'historique est affiché sous forme d'une table JTable et les opérations sont extraites d'un fichier csv "comma separated values". Le but de l'utilisation de ce type de fichier est de permettre de garder une sauvegarde du fichier même en cas de panne électrique, ce qui peut bel et bien arriver dans un restaurant. Ce fichier contient toutes les opérations effectuées, la date, le montant ainsi que la balance associés à chaque opération.

- Pour donner le relevé de dépôt pour un mois précis ainsi que la comparaison des gains mensuels, on a ajouté deux classes intermédiaires afin d'interagir avec l'utilisateur en lui demandant le mois et l'année pour lesquels il souhaite avoir un relevé de dépôt, ou bien deux mois dont il souhaite comparer les revenus et les dépenses.

- Pour l'affichage de relevé de dépôt on a utilisé JTable en se limitant à l'affichage d'opérations effectuées seulement dans le mois choisi par l'utilisateur.

- Pour la comparaison des revenus mensuels, on les a sommés pour chaque mois choisi puis on a affiché chaque somme, avec un message qui annonce le mois qui a généré le plus de revenus.

2.1.6 Gestion de stock

- Classe DishIngredient : qui a pour but de rassembler les informations des plats.

- Classe Ingredient : Donne la possibilité d'ajouter, avoir son nom, voir s'il est valable, ajouter et enlever les ingrédients du système, ajouter une quantité de cet ingrédient dans le stock; etc.

- Classe Quantity : c'est une classe qui représente le stock des aliments (ingredients) que le chef utilise .

- Classe Order : une classe qui définit les plats qui sortent de la cuisine afin de soustraire du stock les aliments utilisés .

- Une interface Order qui va nous permettre de définir trois types de size qui sont une réalisation de la classe menu, pour pourvoir retirer du stock la quantité nécessaire dépendant de la taille du plat pris par le client.

- FullSize pour la taille M
- HalfSize pour la taille S
- SuperSize pour la taille L

- Un API qui affiche le menu concernant la gestion du stock et qui permettra de faire la gestion du stock en utilisant un affichage de menu ainsi que récupérer le choix de l'utilisateur (ici les employés du restaurant qui ont accès). Les bibliothèques utilisées sont :
`java.util.ArrayList`; (pour l'utilisation des listes)
`java.util.Date;` (pour l'utilisation de la date)
`java.util.Calendar;` (pour utiliser le calendrier)

- Pour la gestion des ingrédients on a favorisé l'utilisation d'une liste avec `arraylist` de java et grâce aux notions vue dans le cours des patrons on a pu utiliser le `foreach` qui facilite le parcours.

- La Classe Client gérera toute la logique d'interface avec l'utilisateur.
inventoryManager : est une classe qui manipule une liste des ingrédients que le système gérera tels que tomate, oignon , pâtes ,viande etc.

- Observer : est une classe qui a pour but de gérer une liste de plat avec des fonctions déjà définis dans les listes en java. L'utilisateur dans ce cas pourra ajouter, initialiser, enlever un plat et la consulter .

- Ordering system : gère la relation entre les ingrédients et les plats ainsi toute commande de plat agira sur les ingrédients en les diminuant dans le stock avec la fonction `orderDish()`.

2.2 Les problèmes rencontrés et les solutions apportées

`Mehdi WISSAD` : J'ai eu des problèmes lors de l'utilisation des listes et aussi dans le choix de la structure de données a utilisée mais grâce à la communication et le débat avec les membres du groupe on a pu relever cette difficulté . Un autre problème que j'ai rencontré et celui de la notification des autres objets de ma classe du changement d'état d'un objet , mais grâce au cours des patrons , j'ai compris la notion d'observer , et aussi d'iterator qui on été très utile dans la conception de la gestion de stock .

`Chaimaa LOTFI` : J'ai eu le problème de l'écriture et la lecture de fichier csv "comma separe values", qui stocke les dépenses et revenus du restaurant, puisque c'est un type de fichier que j'utilise pour la première fois .
J'ai trouvé des problèmes au niveau graphique afin d'utiliser des images qui conviennent le plus avec le thème couleur de notre application .

D'autre part, en ce qui concerne les interfaces graphiques, j'ai eu plusieurs problème comme écrire sur une image ou placer un bouton sur elle mais avec la pratique j'ai pu surmonter ce problème.
L'affichage de l'historique a été conçu avec JTable ce qui était un peu difficile , mais grâce à Younes j'ai pu le faire . (Merci Younes). J'ai utilisé ComboBox pour afficher tout les mois et les années et j'avais un problème pour détecter le choix de la date entrer par l'utilisateur la solution était d'utiliser la commande "combox.getItemAt(combobox.getSelectedIndex())" ainsi que les getters et les setters (Merci Hamza pour l'aide).

`Hatim MESKINE` : Lors de de l'étape de l'internationalisation, j'ai commencé par utiliser la classe ResourceBundle, la classe qui gère les fichiers .property et qui contient les données spécifiques aux paramètres régionaux. Après avoir fait quelques essais sur les premières pages (`Home`,`SignIn`), tout semblait fonctionner. Cependant, lors de l'essai de l'étape d'inscription, le mail de vérification n'a pas pu être renvoyé. Effectivement, une exception a été levée. N'ayant pas participé à l'implémentation du système de mail, j'ai mis du temps à décerner le lien entre les deux. Ce n'est qu'après concertation avec mon camarade Younes que j'ai su que le code concernant le mail utilisait aussi les formats de fichier .properties. J'ai fini alors par changer de méthode en utilisant les formats JSON qui ont l'avantage d'être simple à mettre en œuvre et plus pratiques.

2.3 Organisation de l'équipe et la mise en œuvre des méthodes agiles

Agilité:

Depuis le début, une Burndown Chart a été implémentée par notre Facilitator et Scrum Master sur *Trello*. Cette Burndown Chart contient le Product Backlog général ainsi que 3 Iteration Backlogs qui spécifient les fonctionnalités *features* à implémenter pour chaque itération. Ces Backlogs ont été bien sûr mis à jour à chaque fois qu'une fonctionnalité était retirée ou ajoutée (ex: Historique d'achat au lieu de Photo de profile pour les clients). La présence de ce Chart facilitait la supervision du produit et la capacité de toujours savoir qui fait quelle tâche et qu'est-ce qui doit être fait en général.
Vu l'impossibilité de se rencontrer dans la vrai vie, nous avons mis en place un serveur Discord pour que les

membre de l'équipe puissent montrer en live les modifications qu'ils ont effectuées mais surtout pour parler quotidiennement du projet en présence du facilitateur durant des Scrum Meetings.
Puisque le projet est assez grand, nous avons opté pour l'application du Pair Programming pour plusieurs features, ce qui était très bénéfique et cruciale dans la préservation de la clarté et propreté du code ainsi que l'efficacité de la conception. Nous avons aussi testé pratiquement l'application presque quotidiennement pour être sûr de bien livrer de la valeur matière à chaque itération. Ces itérations ont bien entendu été toujours respectées par notre groupe.

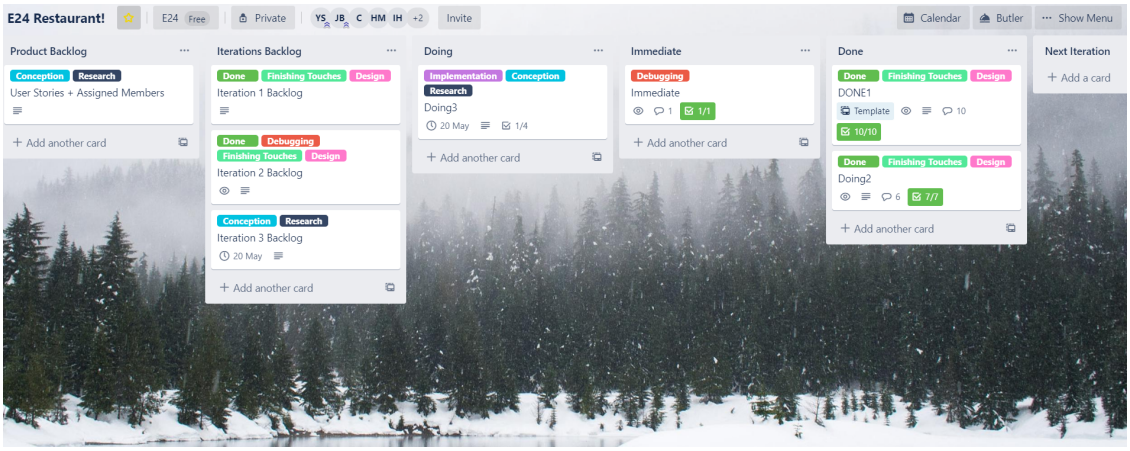

Figure 2.1: Mise en Oeuvre des Méthodes Agiles sur *Trello*

Remarque de quelques collègues:

Quand nous avons décidé de choisir le sujet tout au début, et aussi face à la gestion de notre temps afin d'organiser des réunions et trouver un terrain d'entente sur les différents choix l'implantation, nous avons eu du mal, surtout au début, vu l'ampleur du projet. Ce dernier nous a pris du temps dans l'organisation et la division des taches, mais grâce à mes collègues qui ont été très réactifs mais aussi présents quand un problème apparaissait. Nous avons été unis dès le début et cela nous a permis de mieux communiquer, et d'être dynamique.

-Mehdi Wissad

Parmi les méthodes agiles que nous avons utilisé dans notre projet, l'organisation de la charge de travaille de notre équipe d'où l'utilisation de Trello et la création d'un groupe sur Discord et Messenger était l'une des plus importantes. Nous avons posé des dates fixe de rendu et des objectifs à atteindre à la fin. Nous avons aussi utilisé des méthodes de coordination entre nous afin de lier les différentes parties de notre application et, comme je l'ai dit, de fixer des deadlines pour permettre au membres de mon groupe de pouvoir gérer cette charge de travail sans causer de difficultés à la gestion de projet .

-Chaimaa Lotfi