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

1 Rapport **2**
1.1 Rappel du sujet . . . 2
1.2 Les principales fonctionnalités . . . 2
1.3 Le découpage de l'application en sous-systèmes . . . 3
1.4 Les diagrammes de classe . . . 3

2 Conception **4**
2.1 Les principaux choix de conception et réalisation . . . 4
2.1.1 Log In/Sign Up . . . 4
2.1.2 Cryptage des mots-de-passe . . . 4
2.1.3 Interface des Clients . . . 4
2.1.4 Interface des Employés . . . 5
2.1.5 Gestion de stock . . . 6
2.2 Les problèmes rencontrés et les solutions apportées . . . 7
2.3 Organisation de l'équipe et la mise en œuvre des méthodes agiles . . . 7

3 Bibliographie **9**

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

1.4 Les diagrammes de classe

Conception

2.1 Les principaux choix de conception et réalisation

2.1.1 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`).

2.1.2 Cryptage des mots-de-passe

2.1.3 Interface des Clients

- `Restaurant Menu:`
  - `Choix:`
  - `etc`
- `Payment:`
- `Loyalty Points:` Description

- Loyalty Points stuff 1
- Loyalty Points stuff 2

Final Description

2.1.4 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 abstraite qui dont héritent `Customer` et `Employee`.
- `Financial State:` Description
  - Finiancial stuff 1
  - Finiancial stuff 2

  Final Description

2.1.5 Gestion de stock

- Classe DishIngredient : qui a pour but de rassembler les informations des plats .
- Classe Ingredient :Donne la possibilité d'ajouter , avoir sont nom , voir si il est valable , ajouter et enlever les ingrédients du système , ajouter une quantité de cette ingrédient dans le stock ....
- 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 sorte 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 et cela dépend de la taille du plat pris par le client .
- FullSize pour la taille M
- HalfSize pour la taille S
- SuperSize pour la taille L pour pourvoir retirer du stock la quantité nécessaire et cela dépend de la taille du plat pris par le client .
- 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 et aussi récupérer le choix de l'utilisateur(ici les employer du restaurant qui ont accès) . les bibliothèques utilisées sont : import java.util.ArrayList; (pour l'utilisation des listes) import java.util.Date;(pour l'utilisation de la date) import java.util.Calendar;(pour utiliser le calendrier)
- Pour la gestion des ingrédients on a favorisé l'utilisation d'une liste avec arrayliste de java et cela grâce au notion 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 défini 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 .

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.

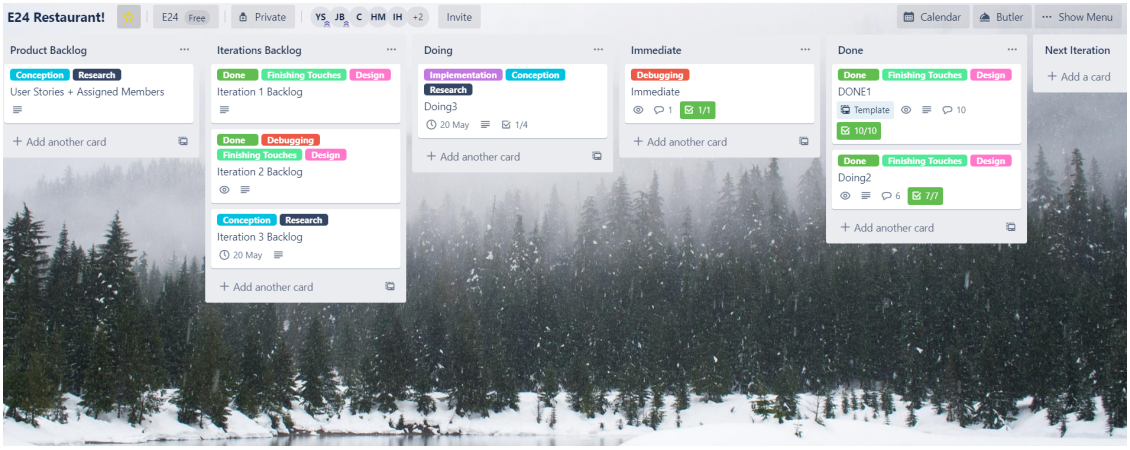

Figure 2.1: Mise en Oeuvre des Méthodes Agiles sur *Trello*

Remarque de l'un de nos 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

Bibliographie