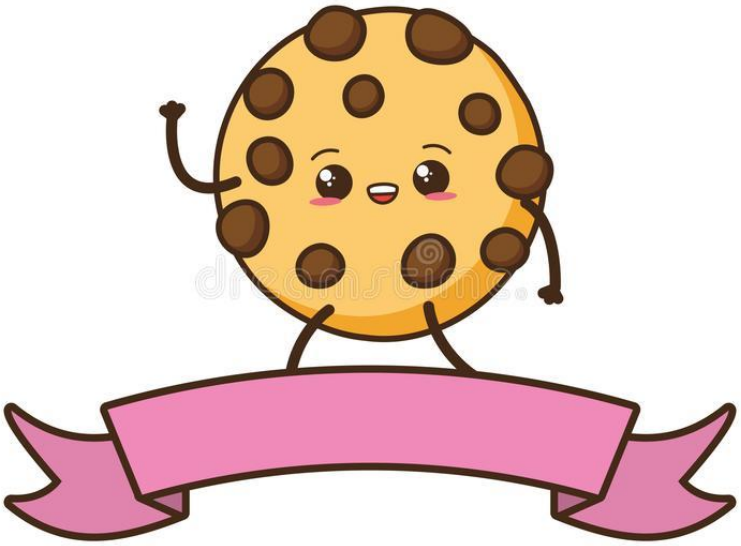
Conception Avancée en UML

The Cookie Factory



Equipe 0

Yahiaoui Imène - Gazzeh Sourour - Paris Floriane

Devictor Pauline - Dibasso Epouta Joel

Sommaire

Introduction

Dans le cadre de ce cours nous avons été chargées d'élaborer une application qui permet de commander des cookies. Le présent rapport a pour objectif de vous présenter dans un premier temps notre conception logicielle. Dans un second, de vous exposer les patrons de conception mises en œuvre dans l'application. Puis enfin de mettre en lumière la façon dont nous avons travaillé le long de ce projet, la cohérence du résultat final avec nos attentes et les aspects non traités du sujet avec les possibles améliorations qui pourraient être apportées à notre travail

Section UML

Diagramme de cas d'utilisation

Suite à la lecture du sujet, nous avons émis les hypothèses suivantes :

- Un client peut passer commande sans être connecté,
- Les responsables des magasins (Store Managers) sont chargés de :
    - Définir les horaires d'ouvertures de leurs magasins,
    - Déterminer le montant des taxes.
- Les chefs peuvent proposer de nouvelles recettes aux responsables de la marque en fonction des ingrédients disponibles dans les catalogues électroniques des partenaires,
- Les responsables de la marque (Brand Leaders) peuvent approuver ou refuser de nouvelles recettes,
- Chaque magasin (Store) a son inventaire (Inventory),
- Si un magasin n'a pas les ingrédients nécessaires pour une recette, la recette n'est pas affichée pour ce magasin.,
- Les responsables d'un magasin peuvent choisir de proposer des cookies pour différentes occasions,
- Les thèmes d'un magasin dépendent des compétences des cuisiniers de ce magasin. Si dans un magasin aucun cuisinier n'a de compétence sur le thème "Musique" alors le thème musique ne sera pas proposé pour ce dit magasin

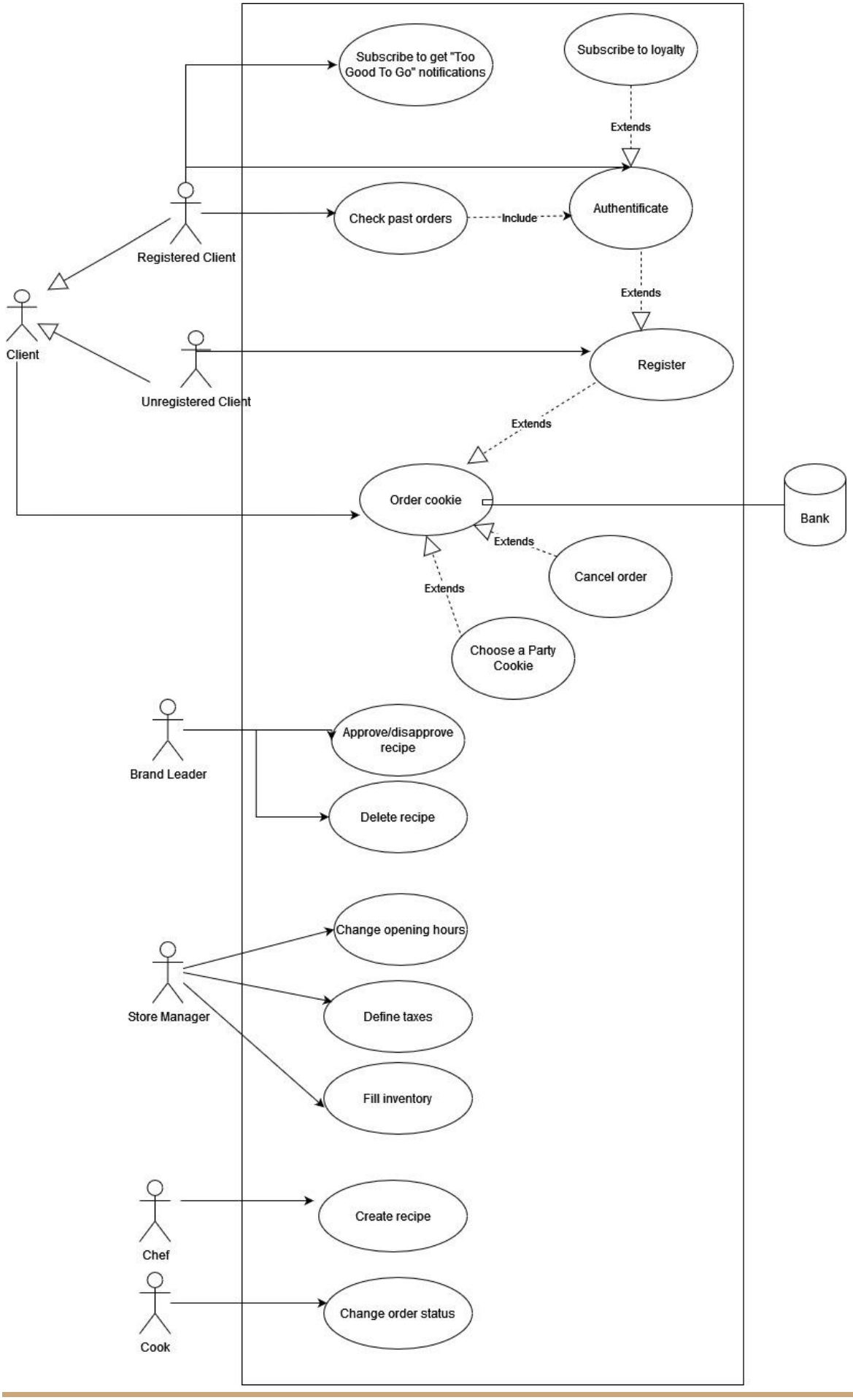

Subscribe to get "Too Good To Go" notifications
Subscribe to loyalty
Extends
Registered Client
Check past orders
Include
Authentificate
Extends
Client
Unregistered Client
Register
Extends
Order cookie
Bank
Extends
Cancel order
Extends
Choose a Party Cookie
Brand Leader
Approve/disapprove recipe
Delete recipe
Store Manager
Change opening hours
Define taxes
Fill inventory
Chef
Create recipe
Cook
Change order status

Diagramme de classe

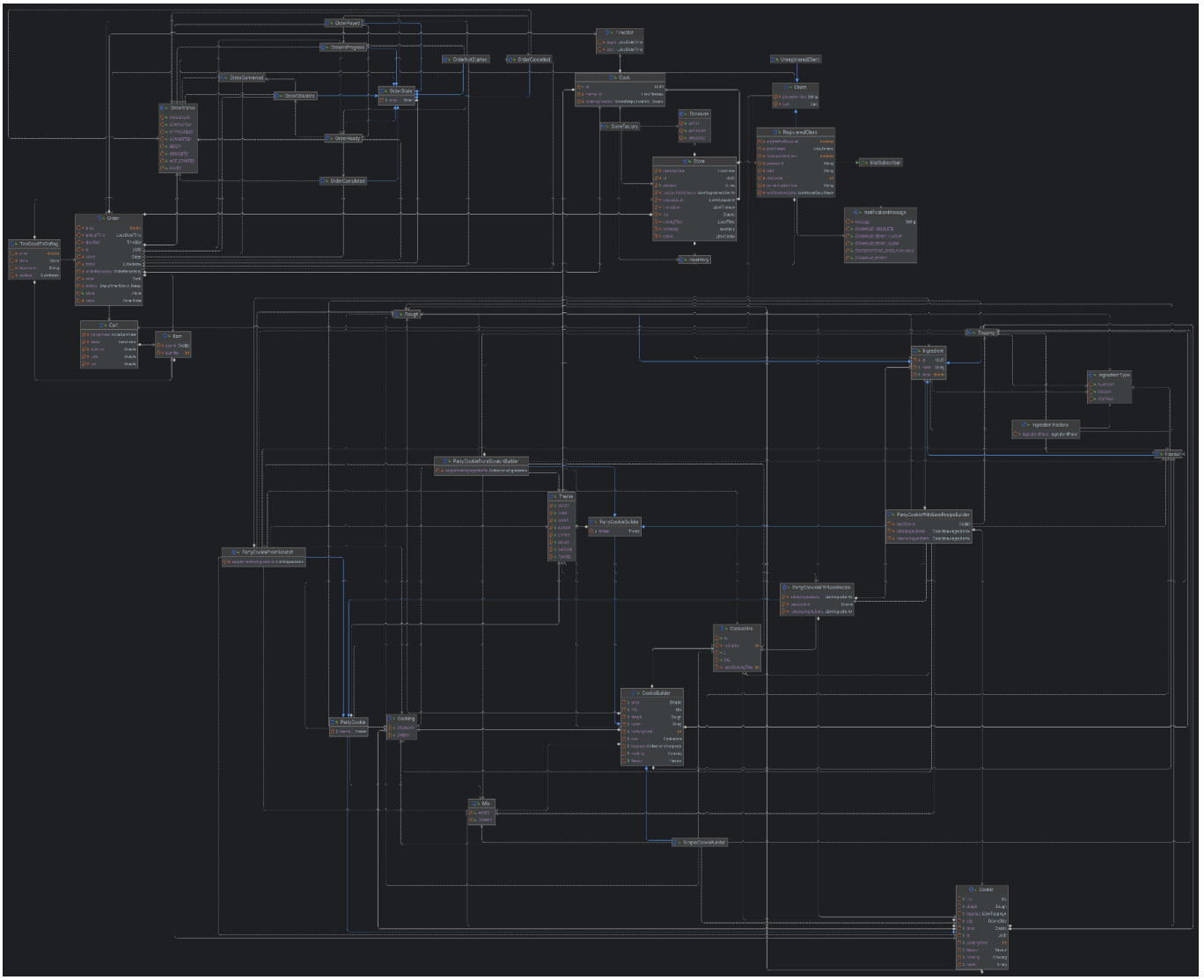
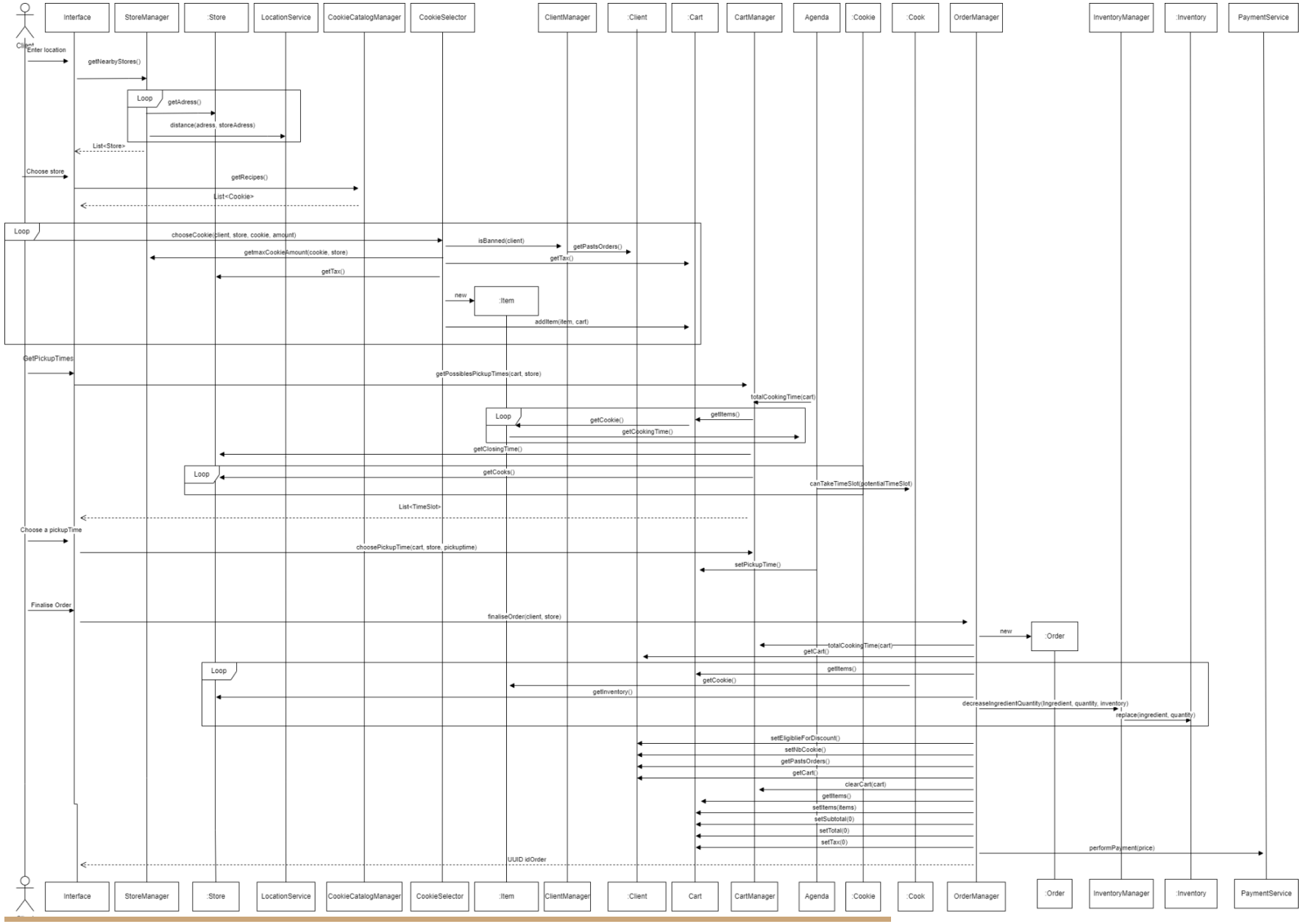


Diagramme de séquences


Client
Interface
StoreManager
:Store
LocationService
CookieCatalogManager
CookieSelector
ClientManager
:Client
:Cart
CartManager
Agenda
:Cookie
:Cook
OrderManager
InventoryManager
:Inventory
PaymentService
Enter location
getNearbyStores()
Loop
getAdress()
distance(adress, storeAdress)
List<Store>
Choose store
getRecipes()
List<Cookie>
Loop
chooseCookie(client, store, cookie, amount)
isBanned(client)
getPastsOrders()
getmaxCookieAmount(cookie, store)
getTax()
getTax()
new
:Item
addItem(item, cart)
GetPickupTimes
getPossiblesPickupTimes(cart, store)
totalCookingTime(cart)
Loop
getCookie()
getItems()
getCookingTime()
getClosingTime()
Loop
getCooks()
canTakeTimeSlot(potentialTimeSlot)
List<TimeSlot>
Choose a pickupTime
choosePickupTime(cart, store, pickuptime)
setPickupTime()
Finalise Order
finaliseOrder(client, store)
new
:Order
totalCookingTime(cart)
getCart()
Loop
getItems()
getCookie()
getInventory()
decreaseIngredientQuantity(Ingredient, quantity, inventory)
replace(ingredient, quantity)
setEligiblieForDiscount()
setNbCookie()
getPastsOrders()
getCart()
clearCart(cart)
getItems()
setItems(items)
setSubtotal(0)
setTotal(0)
setTax(0)
performPayment(price)
UUID idOrder
Interface
StoreManager
:Store
LocationService
CookieCatalogManager
CookieSelector
:Item
ClientManager
:Client
Cart
CartManager
Agenda
:Cookie
:Cook
OrderManager
:Order
InventoryManager
:Inventory
PaymentService

Patrons de conception

Patrons de conception utilisés

- Factory pour la création des Store,
- Abstract Factory pour la création des ingrédients du fait des nombreux sous types possible de la classe Ingrédient,
- Singleton pour le Store factory et le IngredientFactory,
- Builder pour les Cookies.
- State pour gérer les statuts de la commande.

Patrons pertinents non utilisés (ou utilisés dans le passé)

Le Pattern Façade a été appliqué au début de notre projet avec notre classe COD, néanmoins lors du passage sur Spring nous l'avons remplacé par les repositories.

Pattern Observer pourrait être pertinent notifier les cuisiniers d'un magasin quand une commande est validée mais cela n'a été pas fait car c'est le composant Agenda qui attribue les commandes aux cuisiniers, On a pas besoin de notifier tous les cuisiniers.

Le patron Abstract Factory était un patron pertinent pour le projet pour la gestion des Cookies mais nous avons choisi de le remplacer par un patron Builder qui nous semblait plus pertinent.

Approche composant

On a commencé par grossièrement sortir toutes les fonctions des entités et de Cod (notre classe façade), afin de les mettre dans des composants “[Entité]Manager” et des interfaces “I[Entité]”. Puis en nous inspirant de l’exemple donné, nous avons divisé les interfaces par rôle, pour qu’un composant n’ait accès qu’aux fonctions qui lui sont nécessaires. Des composants ont été rajoutés lorsque cela nous semblait logique, comme Agenda qui gère les dates de récupération de commande en fonction des cuisiniers.

Le composant **CookieCatalogManager** gère la liste de Cookie dans le répertoire CookieRepository, qui stocke toutes les recettes validées, et le répertoire SuggestedCookieRepository, qui stocke les recettes proposées mais pas encore validées. Il implémente les interfaces CookieRegistration avec les fonctions permettant de proposer, accepter ou refuser une recette, et CookieFinder qui permet de récupérer les recettes des répertoires. Le composant appelle l'interface IngredientFinder afin de pouvoir vérifier l’existence des ingrédients des recettes dans le répertoire d’ingrédients.

Le composant **IngredientManager** gère les ingrédients du système, stockés dans le répertoire IngredientRepository. L’interface IngredientRegistration permet d’ajouter des ingrédients au répertoire et l’interface IngredientFinder permet aux autres composants de rechercher des ingrédients.

Le composant **InventoryManager** gère les inventaires des stores. Il utilise l’interface IngrédientFinder afin de vérifier l’existence dans le catalogue des ingrédients qu’il ajoute aux inventaires. Il implémente InventoryFiller, utilisé par StoreManager pour remplir les inventaires, et InventoryUpdater, utilisé par OrderManager pour décrémenter ou incrémenter les ingrédients d’un inventaire lorsqu’une commande est passée ou annulée.

StoreManager gère les stores du système, et donc le répertoire des stores. Il implémente StoreRegistration qui permet d’ajouter des magasins, mais égalements des cuisiniers, des occasions et des thèmes. Il implémente également StoreFinder afin de retrouver les magasins, et StoreProcessor utilisé par CookieSelector, nécessaire afin de pouvoir décider si on peut ou non commander un cookie dans ce magasin.

CookieSelector implémente l’interface CookieChoice, et contient les méthodes permettant à un client de remplir son panier. Il utilise CookieFinder pour vérifier l’existence de la recette, StoreProcessor afin de savoir si le cookie peut être fait dans ce

magasin, ClientHandler pour vérifier si le client a l'autorisation de commander, et CartHandler afin d'ajouter des items au panier.

Le composant **Agenda** implémente AgendaProcessor, et gère les dates et heures de récupération de commande en fonction d'une commande, un magasin, et les heures d'ouverture et cuisiniers de celui-ci. CartHandler lui permet de récupérer le temps que prendra une commande en cuisine.

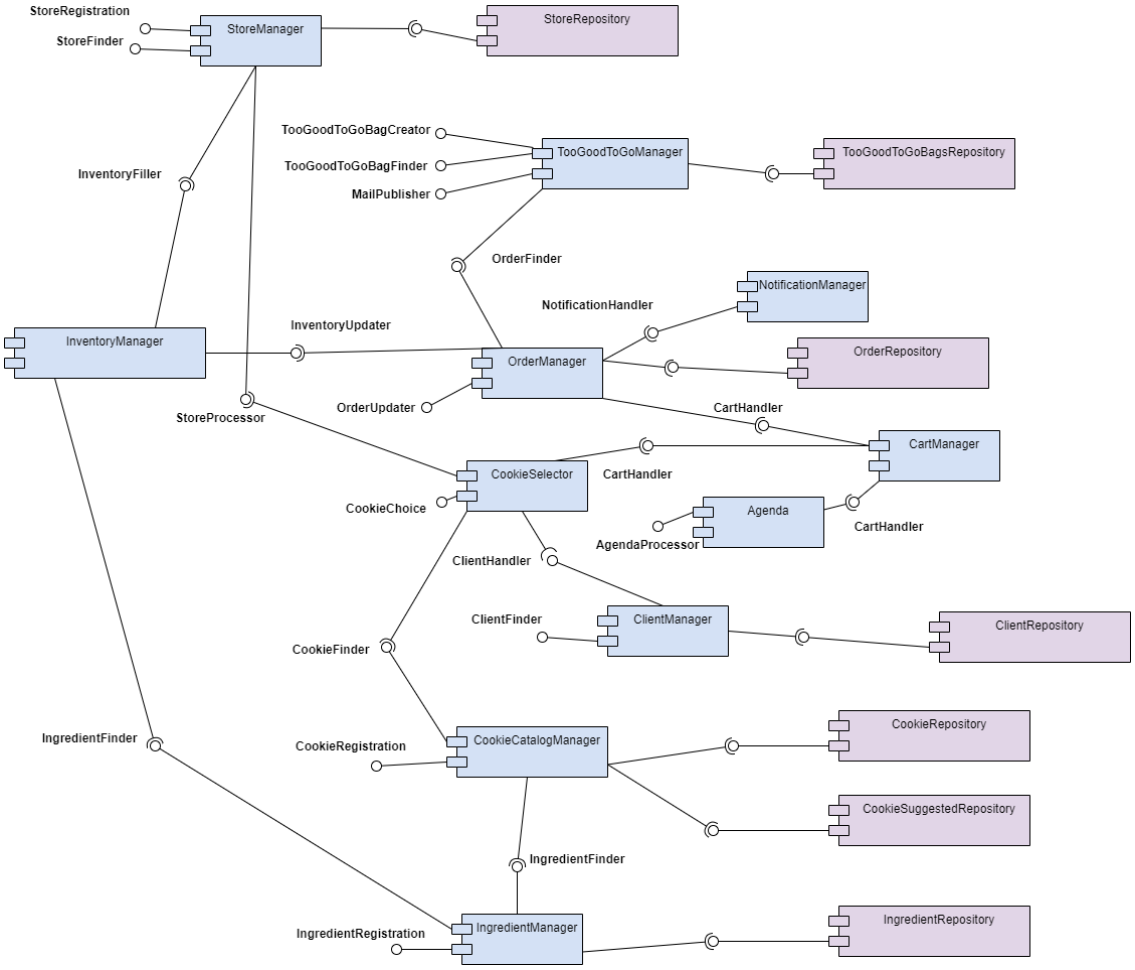
Le composant **OrderManager** implémente OrderUpdater qui se charge de finaliser et de mettre à jour les commandes, et OrderFinder afin de les retrouver dans le répertoire de commande. Il utilise InventoryUpdater qui met à jour l'inventaire, NotificationHandler pour prévénir le client du changement d'état de sa commande et CartManager afin de remettre à zéro le panier.

Le composant **TooGoodToGooManager** implémente TooGoodToGoBagCreator qui s'occupe de la création des paniers TGTG et de leurs publication, TooGoodToGooBagFinder afin de les retrouver dans le répertoire des paniers TGTG, et MailPublisher qui se charge de la notification des clients abonnés à TooGoodToGo.

L'interface CartHandler, implémentée par le composant **CartManager**, gère les opérations sur les paniers des clients.

L'interface **NotificationHandler**, implémentée par le composant **NotificationManager**, gère les notifications à envoyer aux clients.

Diagramme de composant:



Auto-évaluation

Répartition des points

Prénoms NOM	Points accordés (sur 100)
Floriane PARIS	100
Imène YAHIAOUI	100
Joël DIBASSO EPOUTA	100
Pauline DEVICTOR	100
Sourour GAZZEH	100

Conclusion

Avec du recul, nous sommes fiers du travail que nous avons réalisé et de la façon dont nous l'avons fait. Tout au long du projet nous avons bien communiqué, d'autant plus que la parole de chacun a pu être entendue et prise en compte, ce qui nous a permis d'être efficace dans l'implémentation de nouvelles fonctionnalités.
Nous nous sommes bien organisés pour finir d'implémenter toutes les fonctionnalités du sujet avant le passage Spring. Nous avons divisé le projet en 6 milestones. Chaque semaine nous avons travaillé sur une nouvelle milestone, ce qui nous a permis de presque finir l'intégralité des fonctionnalités pour la soutenance intermédiaire.
Nous avons eu des difficultés lors du passage en Spring du projet ce qui nous a grandement ralenti. De plus, nous avons longtemps débattu sur comment découper le projet pour le passage en Spring afin d'éviter de casser le code existant.
Malheureusement, lors de la période des examens, il nous a été moins facile de nous organiser et de diviser les tâches comme nous l'avons fait tout au long du semestre.
Parmi les difficultés que nous avons rencontré, il y a eu :

- La mise en place du catalogue des ingrédients qui a par la suite été remplacé par un répertoire,
- Le Pattern Observer que nous n'avons pas pu mettre en place sur TooGoodToGo,

- La mise à jour des tests avec les composants : ce dernier point a été le plus bloquant de tous pour nous. Nous considérions beaucoup de nos tests comme pertinents, et que nous souhaitions les garder. Néanmoins, les avoir corrigés lors du passage en Spring nous a aussi aidé à détecter puis corriger des problèmes de liens dans nos composants.

Par manque de temps, certaines fonctionnalités que nous avions prévu lors de la première phase de création de notre projet sont manquantes.

Nous avions prévu d'implémenter les fonctionnalités suivantes:

- Le client choisit la date auquelle il souhaite être notifié pour récupérer les commandes TooGoodToGo

-

Annexes

Le glossaire

Mot	Alias	Description
Dough	Type de pâte	
Flavour	Arôme optionnel	
Topping	Garniture	
Mix	Type de mélange	
Cooking	Type de cuisson	
Client		Classe dans notre système
Loyalty	“Loyalty program”	Programme de fidélité du magasin
TCF	The Cookie Factory	
Price	Prix	
Discount	Réduction	
Order	Commande	
Store manager	Responsable des Magasins	
Order Cookie	Passer commande (comprend le choix du cookie & lieu & date & heure)	
Cook	Cuisinier	
Chef	Chef cuisinier	
Brand Leaders	Dirigeants de la marque / Chefs de la marque	
Catalog	Catalogue	
User	Client (derrière son ordinateur), Utilisateur	