# RAPPORT CONCEPTION LOGICIELLE

## The Cookie Factory

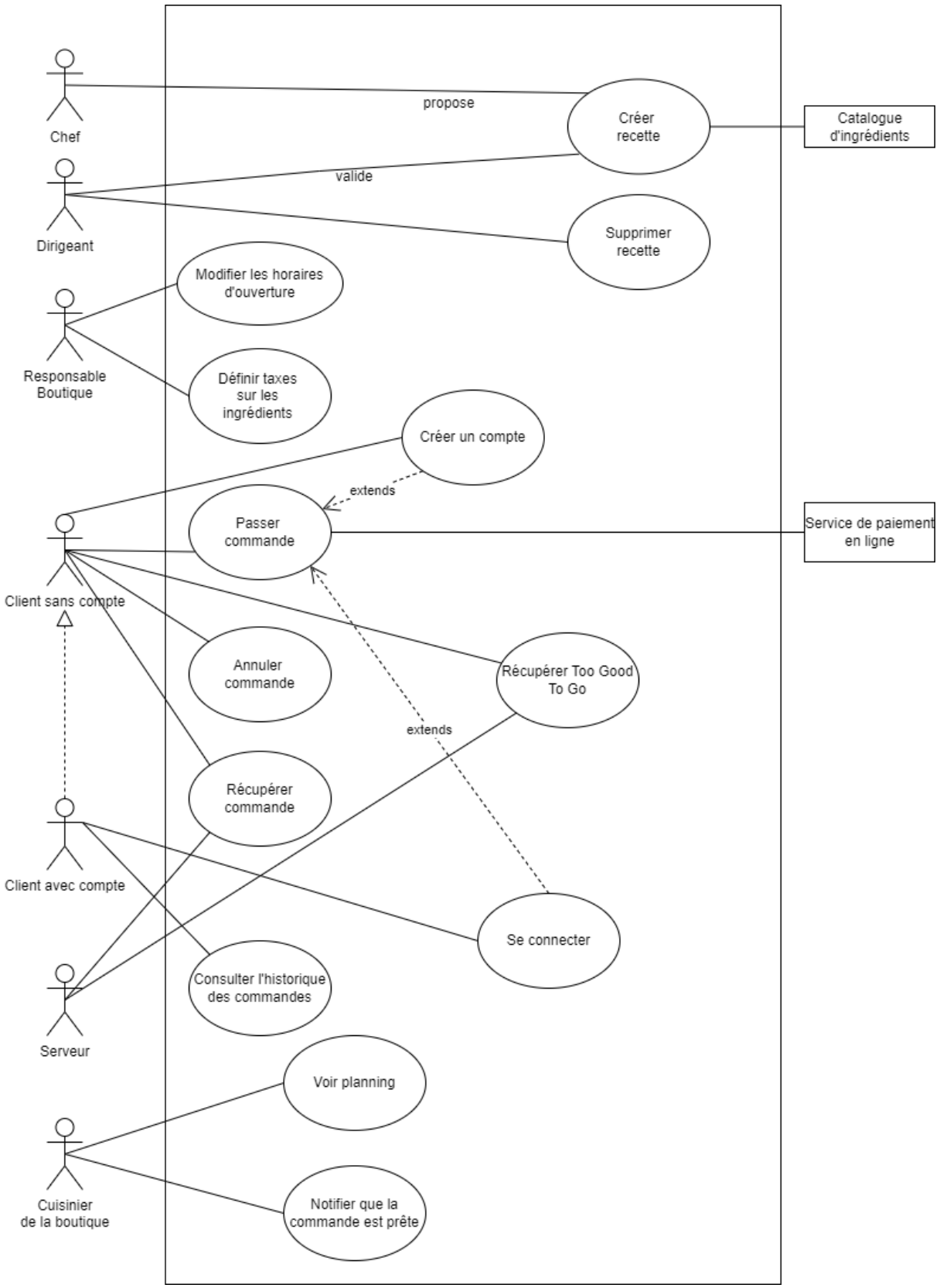
## Team N

Bernigard Benjamin - Bevan Tom - Boudreau Quentin
Correia Ambre - Faucher Vinh

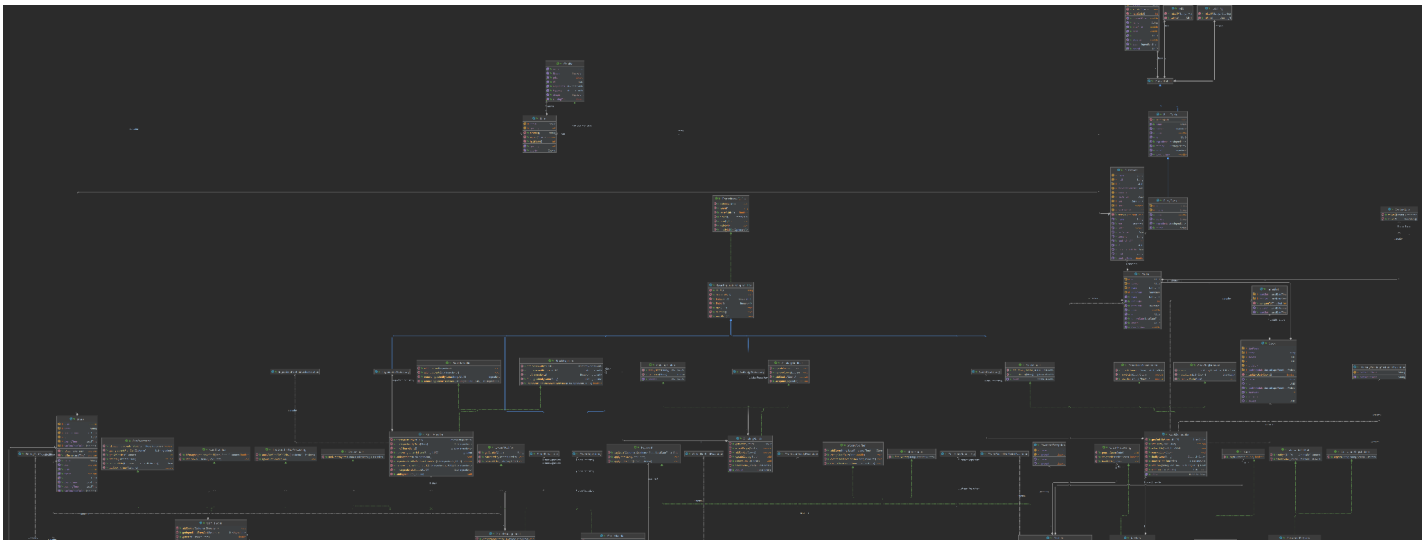
# SOMMAIRE

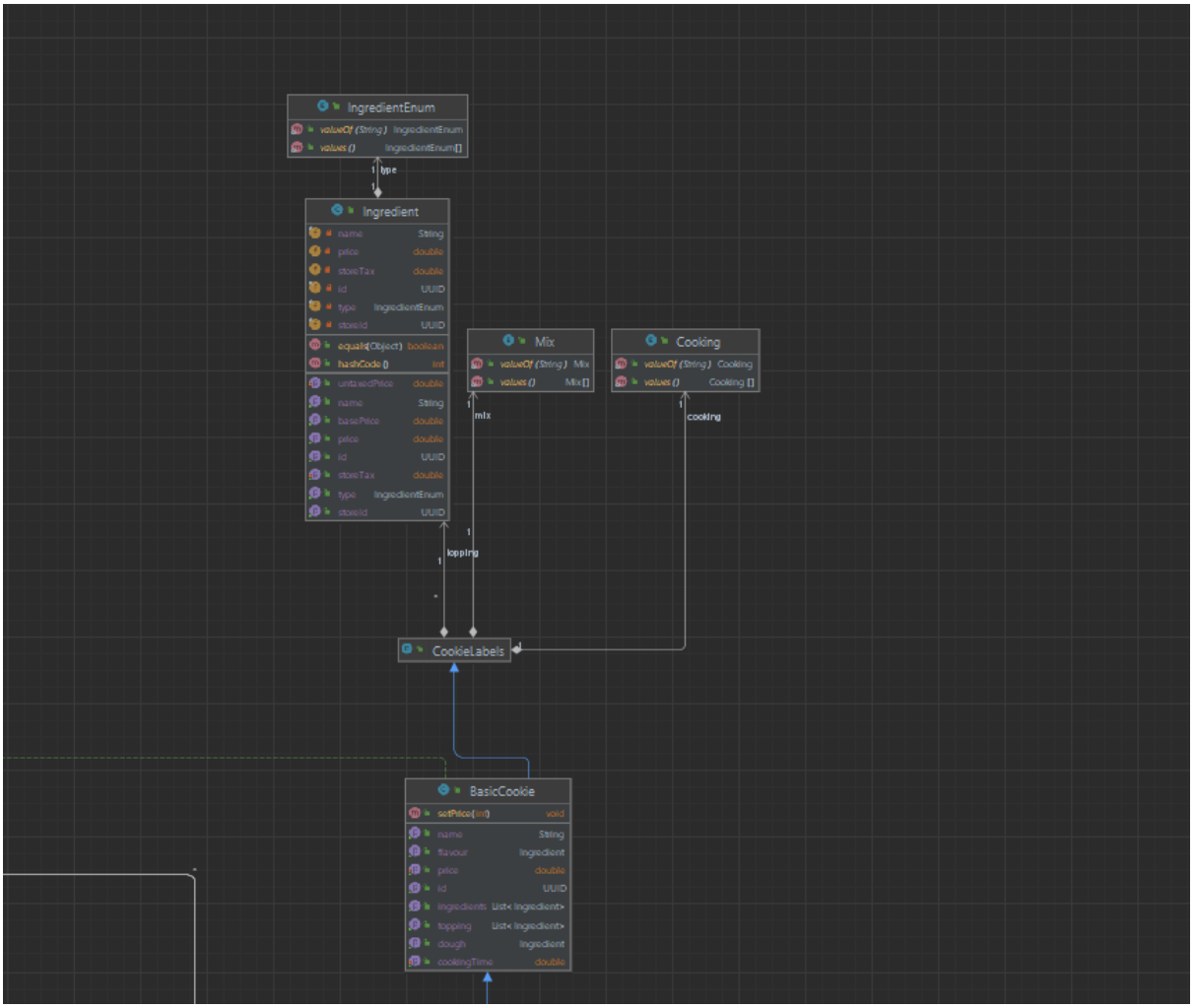
# 1. UML

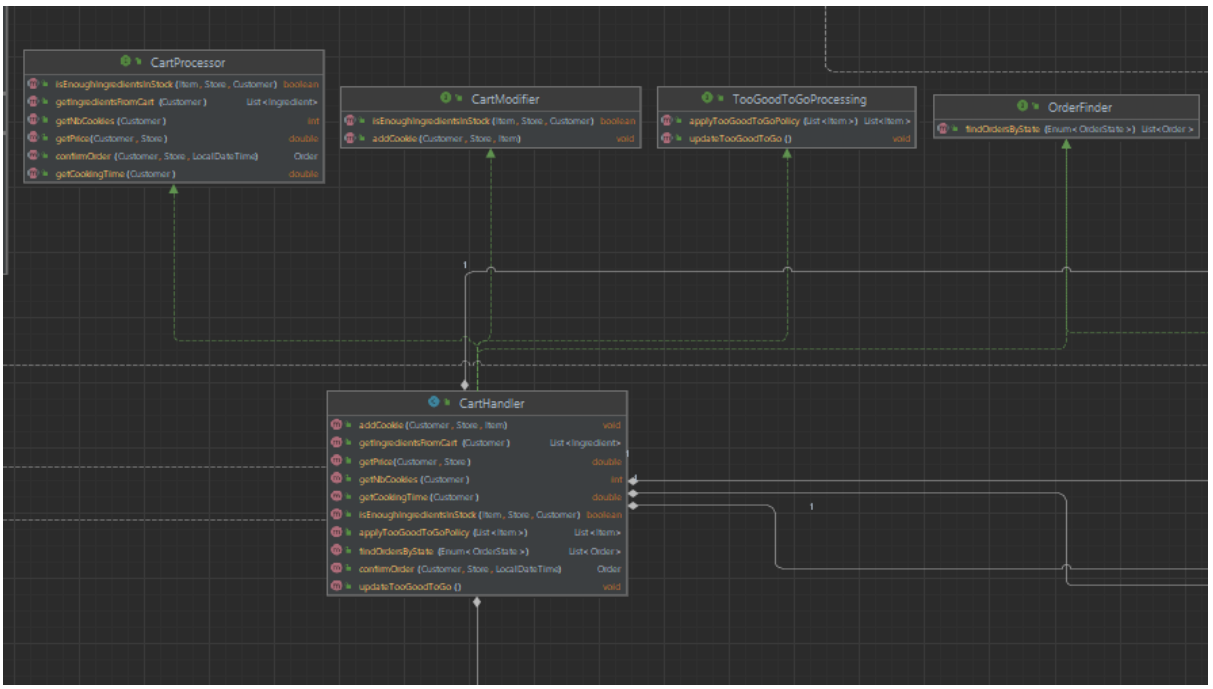
## 1.1. Diagramme de Use Case



## 1.2. Diagramme de classe








Afin de pouvoir parcourir notre diagramme de classe avec plus de lisibilité, le diagramme au format png est disponible dans le dossier doc présent à la racine de notre projet.

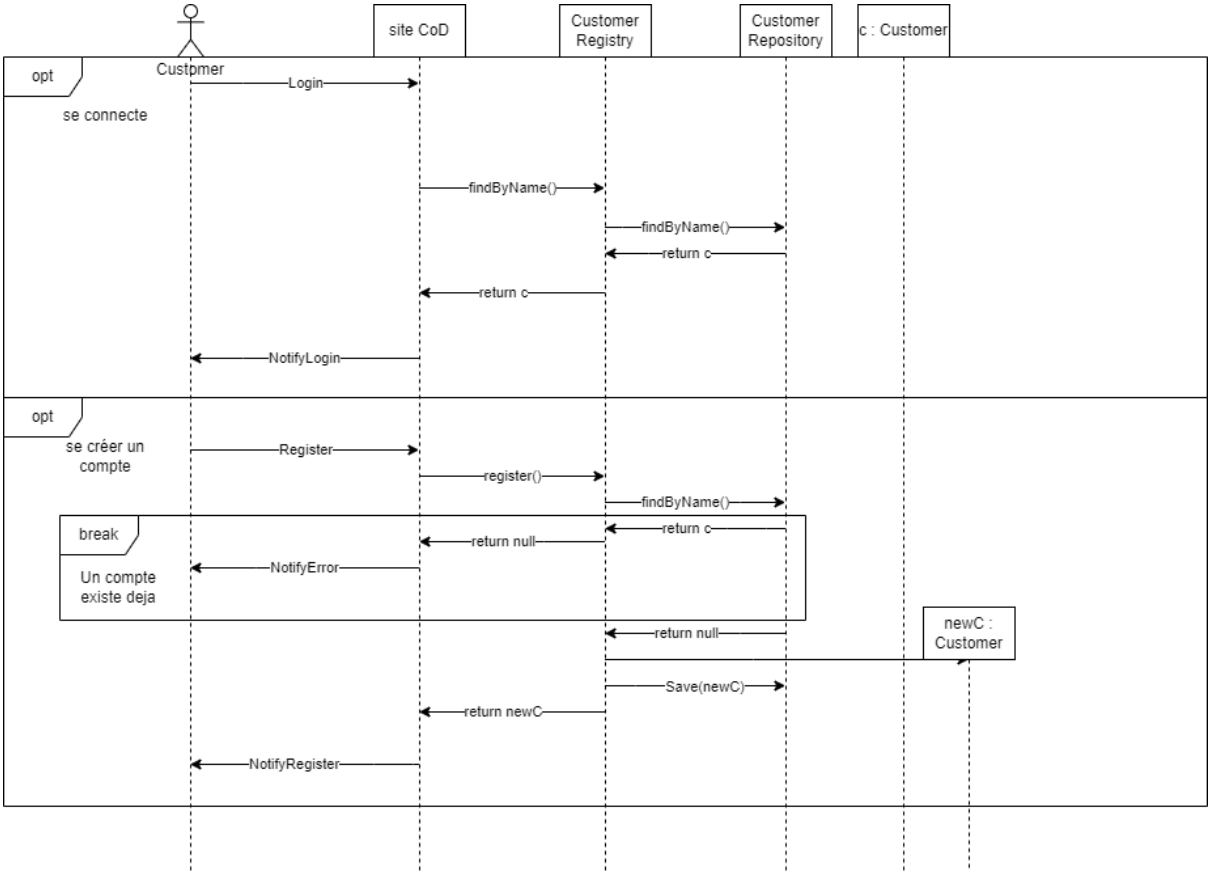
### 1.3. Diagrammes de Séquence

Pour plus de clarté, nous avons découpé notre diagramme de séquence en trois parties :

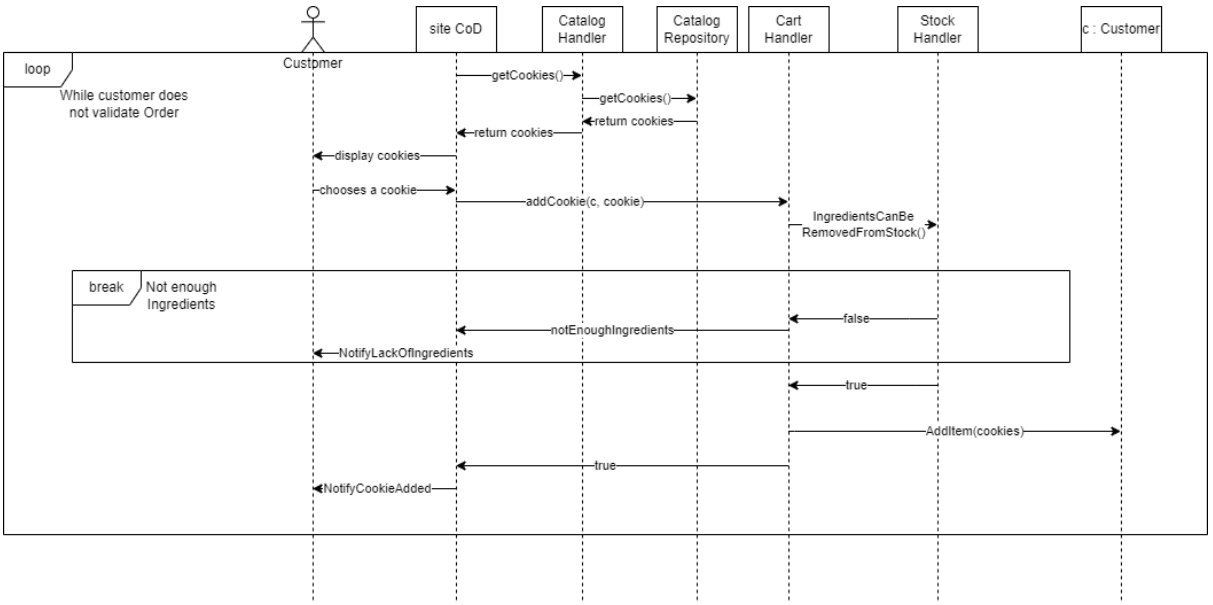
- Se connecter
- Ajouter des cookies au panier
- Valider commande

La connexion au compte et l'ajout de cookie dans le panier peuvent être faits en parallèle.

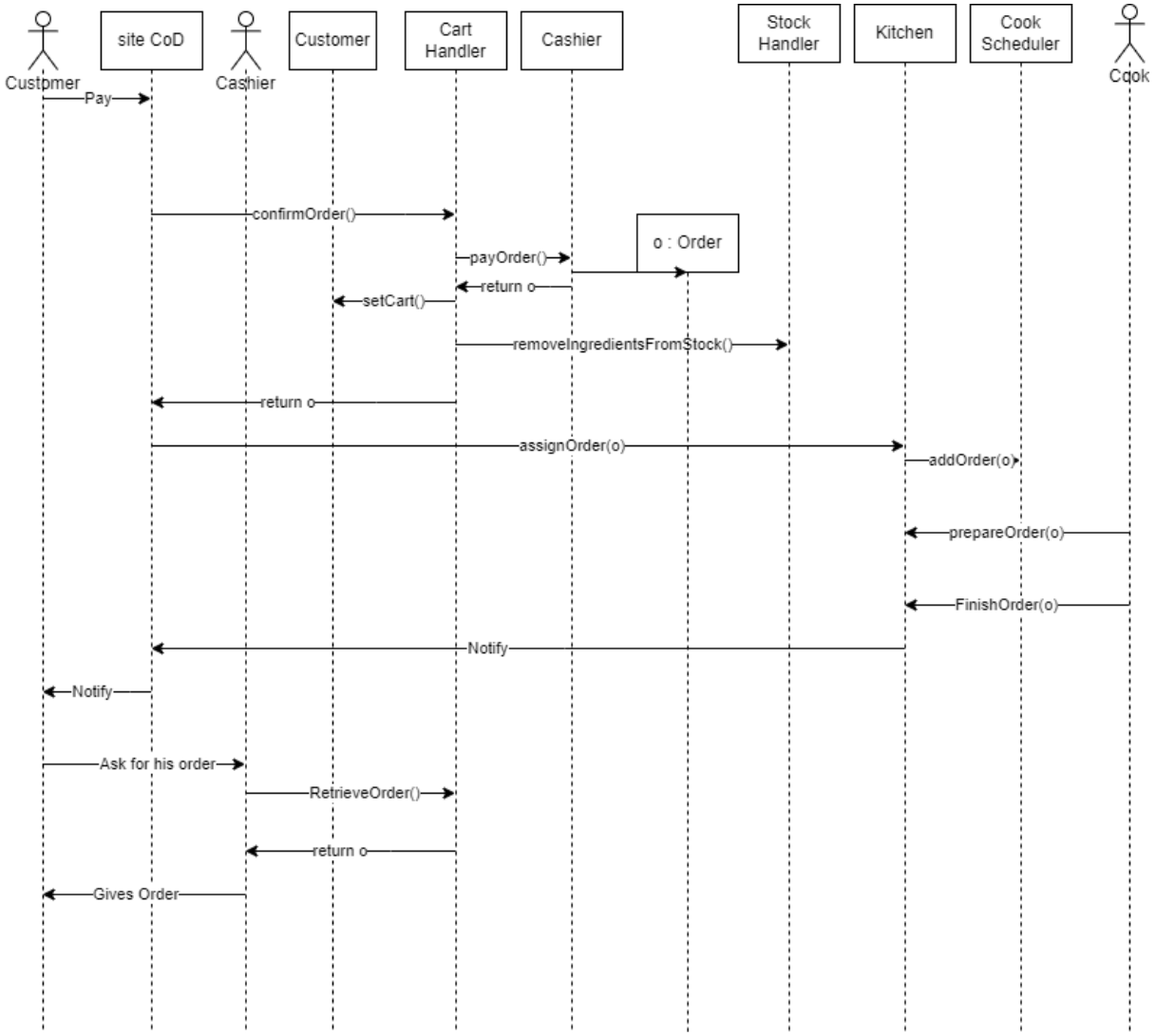
### 1.3.1. Se connecter




### 1.3.2. Ajouter des cookies au panier



### 1.3.3. Valider commande



# 2. Patrons de Conception

## 2.1. Proxy - BankProxy

On a créé un proxy qui est là pour simuler un service de paiement car il est nécessaire d'avoir un service de paiement dans la logique métier. Une idée pour remplacer le proxy serait de faire un appel à un service sécurisé comme Paypal par exemple afin de permettre au client de payer en toute sécurité.

## 2.2. Iterator - Repositories

Les différents repositories ont été implémentés avec comme idée d'utiliser le design pattern iterator. En effet, l'idée était de pouvoir avoir des "fausses" bases de données afin de sauvegarder nos différents éléments et de pouvoir y accéder facilement et rapidement sans que les autres services aient connaissance de la structure qui se cache derrière. Nous n'avons pas utilisé la partie séquentielle parce qu'elle ne nous paraissait pas pertinente dans l'utilisation, mais son implémentation est possible et peu compliquée à effectuer. Nous avons privilégié la recherche à partir d'id, ou de nom selon le besoin, qui est beaucoup plus rapide et pertinente dans notre utilisation.

## 2.3. Singleton - Catalog

Nous avons pensé à utiliser un singleton pour le catalogue au début, mais puisqu'il doit être différent en fonction du magasin nous avons abandonné cette idée. De plus, il est possible que créer un singleton rende les tests plus fastidieux pour un apport minime d'un point de vue logique.

## 2.4. Builder - Cookie

L'idée d'un builder pour les cookies est aussi apparu lors de l'ajout des party cookies mais puisque l'on a toujours toutes les informations qui constituent le cookie à l'instant ou il est généré cela nous a paru être de l'over-engineering qui résoudrait un problème hors du scope du projet.

## 2.5. State - TooGoodToGo

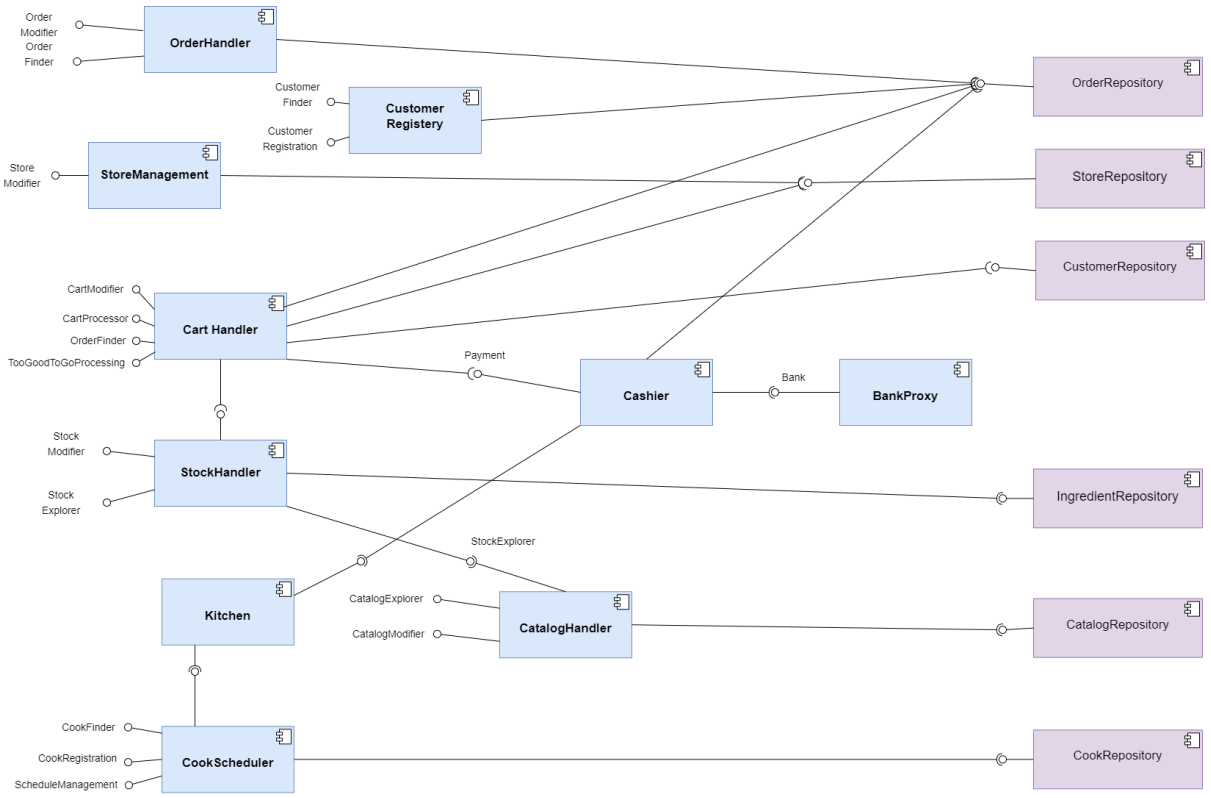
Notre première idée pour implémenter TooGoodToGo était d'utiliser un pattern State qui fait changer le comportement de retrait du panier s'il rentre dans l'état TooGoodToGo. Cependant ce changement de comportement était trop grand pour cacher les différences derrière une interface. En effet pour ce qui est de modifier le calcul du prix ce n'est pas une mauvaise idée, mais pour modifier le comportement que doit avoir le client avec la commande (pas besoin de passer de commande pour récupérer un TooGoodToGo par exemple) cela ne semblait pas adapté à notre code.

### 2.6. Façade - Composants

La majorité de nos composants, notamment CartHandler et CookScheduler, servent en réalité de façades. En effet, ils servent “d’interface” unifiée regroupant différentes sous interfaces sous une seule classe. L’idée est de séparer au maximum les responsabilités, par exemple les méthodes qui modifient des entités et celles qui vont simplement chercher des éléments. Ainsi, en tant qu’utilisateur nous appelons simplement les méthodes, sans savoir qu’en réalité elles sont séparées en différentes interfaces, car nous n’avons pas besoin de le savoir. L’utilisateur va donc simplement utiliser la façade et n’aura pas connaissance des réelles intéractions derrière.

# 3. Rétrospective Spring

## 3.1. Diagramme de composants




## 3.2. Explications

Pour passer de notre première conception objet à notre conception spring nous avons d'abord extrait toutes les interfaces nécessaires. La plupart des extractions ont été assez naturelles comme la capacité d'enregistrer un cuisinier et de trouver un cuisinier dans un magasin donné (respectivement CookRegistration et CookFinder). La classe qui a demandé plus de réflexion à extraire a été notre classe Store qui contient beaucoup d'informations différentes et avait donc, dans notre première conception, beaucoup de responsabilités différentes. En effet Store permettait à la fois de relier le stock, le catalogue ,le planning des cuisiniers et la gestion de too good to go en plus de définir le concept de magasin.

On a donc pu séparer une partie du fonctionnement de notre store dans les cinq composants suivants : CookScheduler, StockHandler, StoreManagement ,CatalogHandler et TooGoodToGoProcessing.

StoreManagement permet tout simplement de gérer les informations liées uniquement au store comme les horaires d'ouverture et les taxes. La logique a changé dans le sens ou la logique du store a été déportée sur les autres composants, ainsi le storeManagement n'a aucune visibilité sur la gestion du stock et la gestion du stock n'a aucune visibilité sur la gestion du magasin. Cependant le StockHandler lui peut trouver un

stock spécifique à partir d'un magasin donné puisque cela reste nécessaire à la logique métier de l'application.

# 4. Auto-évaluation de l'équipe

Dans notre implémentation, nous couvrons tous les use case que nous avions identifiés durant notre phase de conception.

Le groupe a été dans l'ensemble plutôt équilibré sur le projet mais nous avons eu du mal à paralléliser le travail lors de la migration Spring.

Notre manque de compréhension de la conception Spring et le fort niveau de couplage au détriment d'un faible niveau de cohésion au début de la migration nous a causé du tort dans l'organisation de l'équipe et nous a fait prendre du retard.

| Nom Prénom | 500 points |
|---|---|
| Bernigaud Benjamin | 100 |
| Bevan Tom | 100 |
| Bourdeau Quentin | 100 |
| Correia Ambre | 100 |
| Faucher Vinh | 100 |